Capítulo 3. Funcionamiento e Interconexión del Computador

El ciclo de la instrucción consiste en la obtención de una instrucción, seguida de cero o más obtenciones de operando, seguidas de cero o más almacenamientos de operando, seguidos de un cheque de interrupciones (si están habilitadas).

Los componentes principales de un sistema computacional (procesador, memoria principal, módulos I/O) necesitan estar interconectados para poder intercambiar datos y señales de control. El medio más común de interconexión es el uso de un bus de sistema con múltiples lineas. En los sistemas actuales existe una jerarquía de buses para mejorar el desempeño.

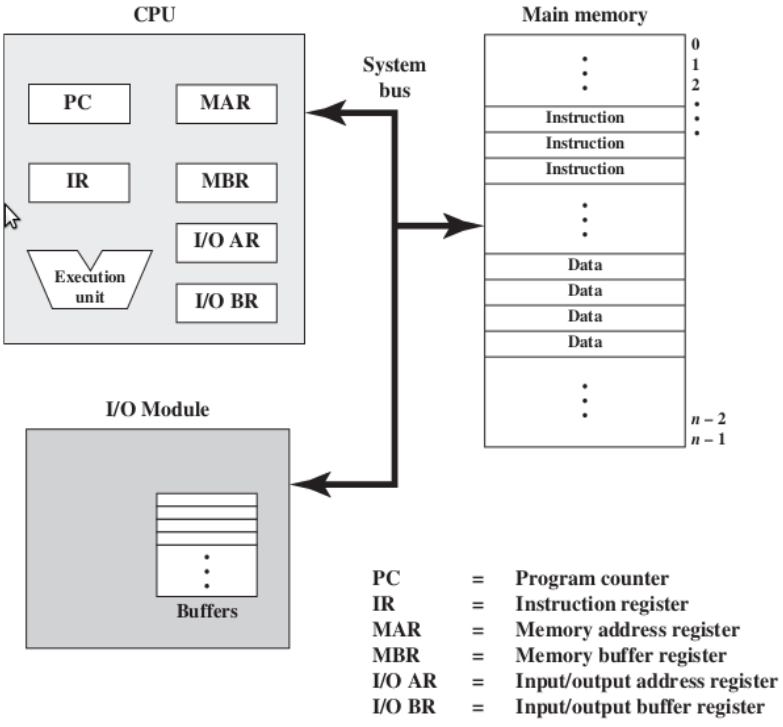
Elementos claves del diseño de buses incluyen el arbitraje (centralizado o descentralizado), la temporización y ancho (número de líneas de datos y número de líneas de dirección).

En el nivel más alto, una computadora se compone de un CPU, memoria y componentes de entrada y salida. Estos elementos se interconectan para lograr la función básica de una computadora, ejecutar programas. Así, se puede describir una computadora describiendo el comportamiento de cada componente, es decir, las señales de datos y control que intercambia con otros componentes; y describiendo la estructura de interconexión y los controles empleados para usar dicha estructura. Éste enfoque favorece la comprensión de cuestiones complejas de desempeño como son los cuellos de botella, la magnitud de algunas fallas, la facilidad para agregar mejoras, caminos alternativos, etc. En muchos casos, los requerimientos de mayor desempeño se logran cambiando el diseño sin tener que incrementar las características de los componentes individuales.

3.1 Componentes del computador

La arquitectura de von Newumann se basa en tres conceptos clave:

- Los datos y las instrucciones se almacenan en la misma memoria.
- Los contenidos de dicha memoria son direccionables por localidad, sin importar el tipo de información que contienen.
- La ejecución ocurre de manera secuencial de una instrucción a la siguiente.



El razonamiento es el siguiente: existe un conjunto pequeño de componentes que pueden ser combinados de diversas maneras para almacenar datos o efectuar operaciones lógicas y aritméticas. Si se quiere efectuar una operación determinada se construye la configuración correspondiente. Si se considera el proceso de interconectar los componentes como una forma de programación, entonces el "programa" resultante es la estructura que tiene el hardware y se le conoce como *hardwired*.

Considere la alternativa, se construye una configuración de propósito general que es capaz de efectuar diferentes operaciones dependiendo de ciertas señales de control que se le apliquen. Con este esquema, el sistema toma señales de control y de datos y produce resultados. Así, en lugar de tener que reconectar los componentes para cada programa, el programador simplemente debe

de proveerlo con un nuevo conjunto de señales de control. Para cada operación a realizar se necesita un conjunto de señales de control en específico, si se identifica cada operación mediante un código podemos agregar al hardware un módulo que acepte dichos códigos y genere las señales de control necesarias.

Así para programar, en lugar de reconectar los componentes para cada programa, únicamente se proveé una nueva secuencia de códigos, cada código representa una instrucción y el hardware interpreta cada instrucción y genera señales de control. A los programas generados con este método se les llamó *software*.

Debería ser claro se han definido dos componenetes importantes del sistema: un intérprete de instrucciones y un módulo de propósito general que efectúa operaciones aritméticas y lógicas. Estos constituyen al CPU. Sin embargo aún son necesarios componentes que nos permitan ingresar instrucciones al sistema (módulo de entrada) así como desplegar los resultados (módulo de salida), a éstos módulos se les llama **componentes I/O**.

Adicionalmente, se requiere un módulo que permita almacenar temporalmente las instrucciones y los datos. El **módulo de memoria** consiste de un conjunto de localidades, definidas por direcciones numeradas secuencialmente. Cada localidad contiene un número binario que puede ser interpretado como dato o como instrucción.

El CPU intercambia datos con la memoria, y los módulos I/O transfieren datos entre los dispositivos externos y el CPU y la memoria. Para ello utilizan registros internos: el CPU utiliza el registro de dirección de memoria (MAR) y el registro de búfer de memoria (MBR); los módulos I/O utilizan el registro de dirección de I/O (I/OAR) y el registro de búfer de I/O (I/OBR).

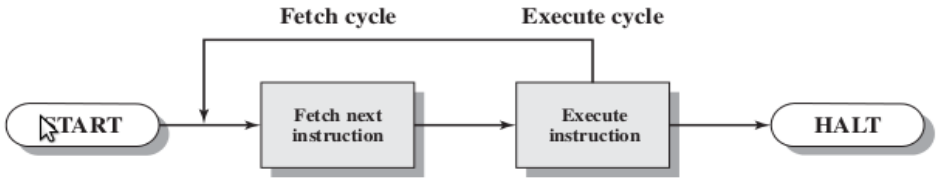
- MAR – especifica la dirección de memoria para la siguiente operación de lectura o escritura.
- MBR – contiene los datos a ser escritos en memoria o recibe los datos leídos de la memoria.
- I/OAR – espedifica un dispositivo I/O en particular.
- I/OBR – contiene los datos a ser enviados al dispositivo I/O o los datos leídos del dispositivo I/O.

3.2 Funcionamiento del Computador

La función básica de una computadora es la ejecución de un programa, el cual consiste de un conjunto de direcciones almacenadas en memoria. En su forma más simple, el procesamiento de una instrucción consiste de dos pasos: el procesador lee (obtiene, *fetches*) las instrucciones de la memoria una a la vez y ejecuta cada instrucción. La ejecución de un programa consiste en la repetición de estos dos pasos *fetch instruction – execute instruction*. La ejecución de programas únicamente se detiene si la máquina se apaga, algún tipo de error irrecuperable ocurre o una instrucción indica a la máquina que se detenga.

El procesamiento requerido para una instrucción se conoce como **ciclo de instrucción**. Usando la descripción simplificada de dos pasos definida previamente, el ciclo de la instrucción consiste de el ciclo de obtención (*fetch cycle)* y el ciclo de ejecución (*execute cycle*).

Ciclos Fetch y Execute



Al inicio de cada ciclo de instrucción, el procesador trae de la memoria una instrucción. Típicamente, un registro llamado contador de programa (PC, *program counter*) tiene la dirección de la instrucción a traer. A menos de que se indique lo contrario, PC es incrementado después de cada ciclo fetch, de esta manera, cada instrucción es traída en secuencia (se trae la instrucción ubicada en la siguiente localidad de memoria), por ejemplo, si PC se encuentra apuntado a la dirección 300, los siguientes ciclos de instrucción traerán las instrucciones de las localidades 301, 302, ... , aunque esta secuencia puede ser alterada.

La instrucción obtenida se carga en un registro llamado registro de instrucción (IR, *instruction register*). La instrucción contiene bits que especifican la acción a tomar. El procesador interpreta la instrucción y efectúa la acción. En general, estas acciones entran en alguna de las siguientes 4 categorías;

- Procesador-memoria: transferir datos del procesador a la memoria o viceversa.
- Procesador-I/O: transferir datos entre el procesador y un dispositivo I/O o viceversa.
- Procesamiento de datos: realizar algún tipo de operación aritmética o lógica sobre los datos.
- Control: instrucciones que modifican la secuencia de ejecución.

Considere el ejemplo presentado a continuación. Una máquina hipotética que contiene un único registro de datos AC *(accumulator)*. Las instrucciones y los datos son de 16 bits por lo que se utilizan palabras de ese tamaño. El formato de instrucción dedica 4 bits para el código de operación *(opcode)* por lo que existen hasta 16 códigos diferentes, y hasta 4096 (2^12) localidades de memoria pueden ser direccionadas directamente.

(a) Instruction format

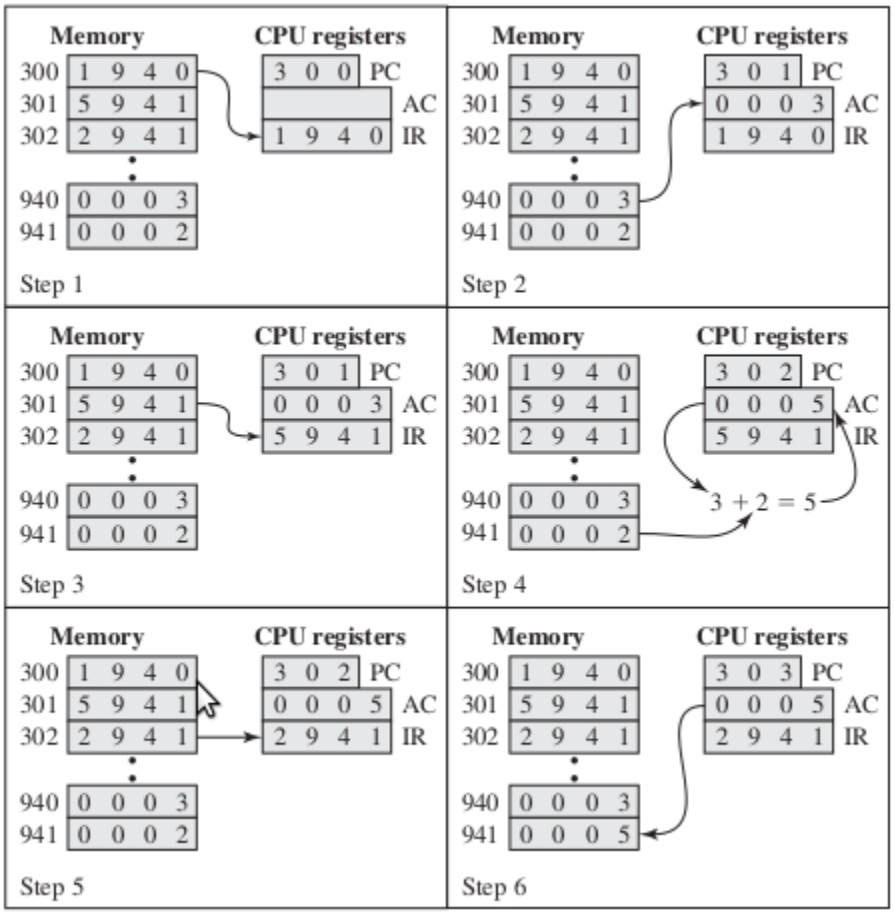
(b) Integer format

Program counter (PC) = Address of instruction
Instruction register (IR) = Instruction being executed
Accumulator (AC) = Temporary storage

0001 = Load AC from memory
0010 = Store AC to memory
0101 = Add to AC from memory

(d) Partial list of opcodes

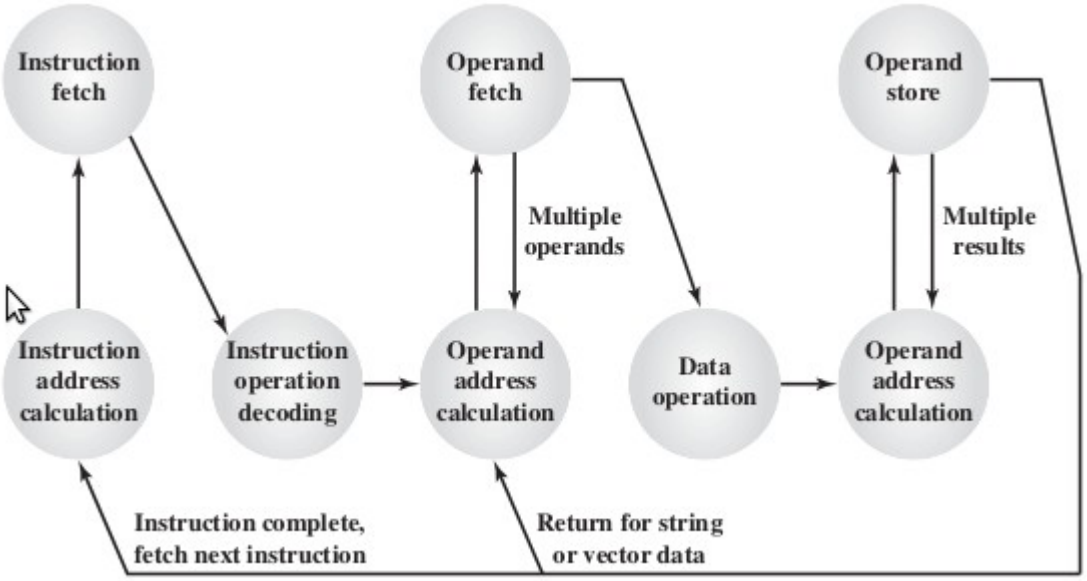
Capítulo 2: Evolución de las Computadoras



La figura ilustra parte de la ejecución de un programa. Se utiliza notación hexadecimal donde cada dígito representa 4 bits. Esta notación es conveniente cuando el tamaño de palabra es múltiplo de 4. En el ejemplo, se utilizan 3 ciclos de instrucción, cada uno con sus subciclos fetch y execute, para sumar los contenidos de la localidad 940 a los contenidos de la localidad 941. Si se tuvieran instrucciones más complejas, se requerirían menos ciclos de instrucciones. Por ejemplo, existen procesadores con instrucciones como ADD A, B donde se almacena la suma de los contenidos de las locaciones de memoria B y A en la locación de memoria A. En éste caso 1 solo ciclo de instrucción tendría los siguientes pasos:

- Obtener la instrucción ADD.
- Leer los contenidos de la localidad de memoria A hacia el procesador.
- Leer los contenidos de la localidad de memoria B hacia el procesador. (Por lo tanto se necesitan 2 registros al menos).
- Sumar ambos valores
- Escribir el resultado desde el procesador a la locación de memoria A.

De ésta forma, el ciclo de execución para una instrucción en particular puede requerir más de una referencia a memoria, o especificar una operacion de I/O. La figura siguiente describe a mayor detalle el ciclo de instrucción. No todos los estados son válidos para todas las instrucciones, algunas de ellas únicamente utilizan un subconjunto de los estados presentados.



- **Cálculo de la dirección de instrucción:** determina la dirección de la siguiente instrucción a ser ejecutada; esto usualmente consiste en sumar un número fijo a la dirección de la instrucción previa. Por ejemplo, si cada instrucción es de 16 bits y la memoria está organizada en palabras de 16-bits, entonces se le suma 1 a la dirección previa. Si, en cambio, la memoria está organizada como bytes de 8-bits direccionables individualmente, entonces sumar 2 a la dirección previa.
- **Obtener la instrucción:** leer la instrucción desde su localidad de memoria hacia el procesador.
- **Decodificar la instrucción:** analizar la instrucción para determinar el tipo de operación y los operandos a utilizar.
- **Cálculo de la(s) dirección(es) de operando(s):** si la operación requiere referencias a operandos en memoria o a través de I/O, entonces se determina la dirección del operando.
- **Obtener operando:** traer el operando de la memoria o leerlo desde I/O.
- **Procesamiento de datos:** efectuar la operación indicada por la instrucción.
- **Almacenar resultado:** escribir el resultado a memoria o a I/O.

Los estados superiores de la imagen requieren intercambios entre el procesador y la memoria o I/O. Los estados inferiores únicamente involucran operaciones internas del procesador. El estado OAC aparece dos veces puesto que una instrucción puede incluír una operación de lectura, de escritura o ambas. ¿Cuál es la secuencia de estados para la instrucción ADD A,B descrita previamente?

3.2 Interrupciones

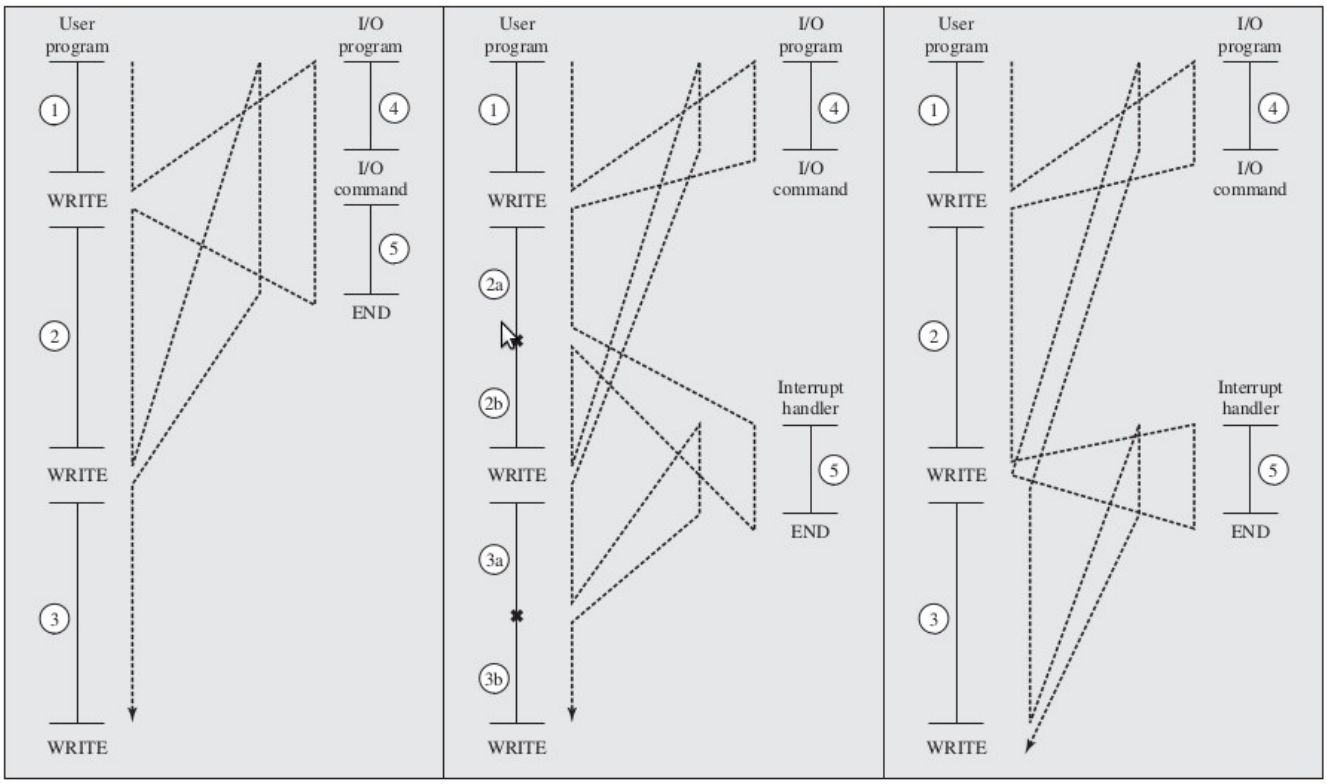
Todas las computadors proveen un mecanismo mediante el cual un módulo (I/O, memoria) pueden interrumpir el funcionamiento normal de un procesador. Las interrupciones se introdujeron como una manera de mejorar la eficiencia del procesamiento. Las principales clases de interrupciones son las siguientes:

- **De programa:** generadas por alguna condición que ocurre como resultado de la ejecución de una instrucción como puede ser un desbordamiento aritmético, división por cero, intentar ejecutar una instrucción ilegal o hacer una referencia a memoria fuera del espacio permitido para el usuario.
- **De temporizador:** generadas por un temporizador dentro del procesador. Esto permite al sistema operativo realizar ciertas funciones de manera regular.
- **De entrada y salida:** generadas por un controlador I/O, para señalar la terminación de una operación, solicitar servicios del procesador o señalar una variedad de condiciones de error.
- **Fallas de hardware:** generadas por fallas como fallas de suministro o errores de paridad.

La mayoría de los dispositivos externos son mucho más lentos que el procesador. Por ejemplo, imagine que el procesador se encuentra transfiriendo información a una impresora, cada vez que se solicite a la impresora escribir (WRITE) el procesador deberá detenerse y mantenerse inactivo hasta que la impresora termine. La longitud de esta pausa puede estar en los cientos o miles de ciclos de instrucciones. Claramente, ésto representa un desperdicio del uso del procesador. En la figura siguiente (izquierda) se observa la situación anterior, los segmentos de código 1, 2 y 3 se refieren a secuencias de instrucciones que no involucran I/O. Las llamadas a WRITE son programas I/O que realizan la operación de escritura. Éste programa consiste de 3 partes:

- Una secuencia de instrucciones (4) que preparan la operación I/O. Ésto puede incluir copiar datos a un buffer especial o preparar los parámetros del dispositivo.
- El comando I/O. Sin el uso de interrupciones, una vez que se ejecuta este comando, el programa deberá a que el dispositivo I/O termine de realizar la acción.
- Una secuencia de instrucciones (5) que completarán la operación, como puede ser establecer una bandera indicando si la operación fue exitosa.

Debido a que la operación I/O puede tomar mucho tiempo para completarse, el programa de usuario se detiene al momento de hacer la llamada a WRITE por un tiempo considerable.

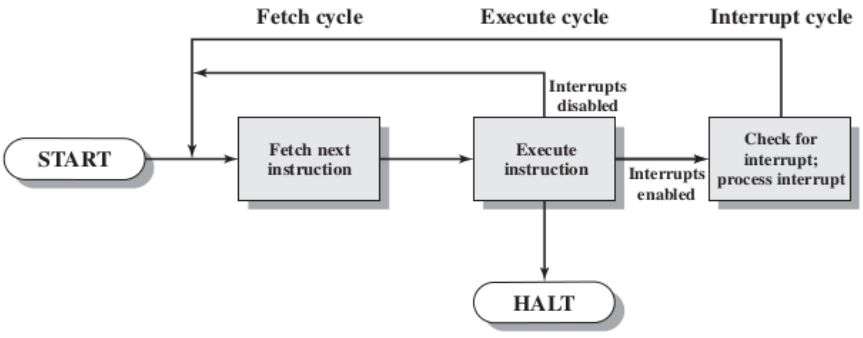


(a) No interrupts (b) Interrupts; short I/O wait (c) Interrupts; long I/O wait

Utilizando interrupciones, el procesador puede continuar ejecutando otras instrucciones mientras se efectúa la operación I/O. En la figura anterior (centro), se observa el resultado de utilizar interrupciones: cuando se realiza una llamada a la subrutina WRITE el programa invocado sólamente consiste del código de preparación (4) y el comando I/O; una vez que se ha ejecutado éstas instrucciones el control se regresa al programa de usuario. Mientras tanto, el dispositivo externo se encuentra trabajando, la operación I/O se realiza de manera concurrrente con la ejecución de instrucciones del programa de usuario.

Cuando el dispositivo externo ha terminado y requiere nuevos datos del procesador, el módulo I/O envía una *solicitud de interrupción.* El procesador suspenderá la operación del programa de usuario y dará inicio a un programa manejador de interrupciones que se encargará de dar servidio al dispositivo externo y continuará con la ejecución original cuando haya terminado. En la figura anterior (centro), las interrupciones se indican con un asterisco.

El manejo de interrupciones es responsabilidad de el sistema operativo, por lo tanto, el procedimiento es transparente para un programa de usuario. El SO se encarga de interrumpir la ejecución del programa de usuario y reanudarlo en el mismo punto.

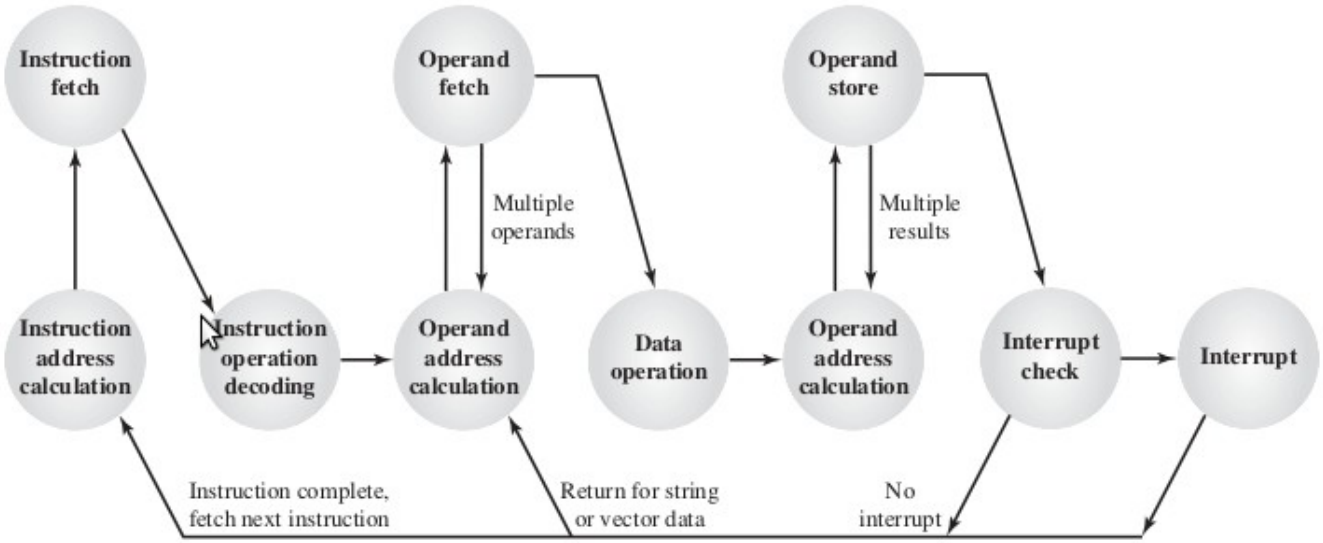


Es necesario entonces agregar un subciclo de interrupción al ciclo de instrucción. En el subciclo de interrupción, el procesador revisa si han ocurrido interrupciones. Si no hay interrupciones pendientes, el ciclo de instrucción continúa normalmente. Si hay alguna interrupción pendiente:

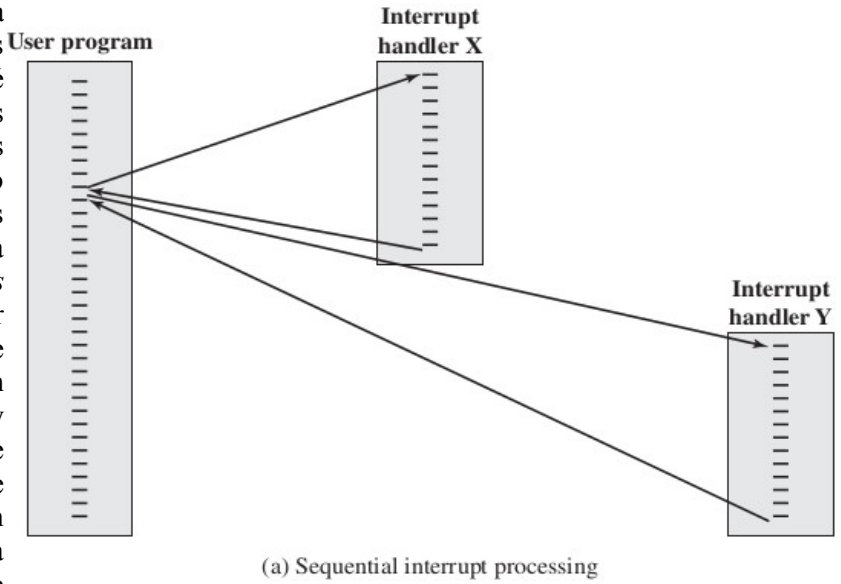
- El procesador detiene la ejecución actual y guarda su contexto, es decir, la dirección de la siguiente instrucción a ejecutarse (PC) y cualquier otra información relevante.
- Establece PC con la dirección inicial de la subrutina de manejo de interrupciones.
- El manejador de interrupciones generalmente es parte del SO. Determina la naturaleza de la interrupción

y efectúa las acciones que sean necesarias.

Es claro que al utilizar interrupciones es necesario ejecutar instrucciones adicionales, por ejemplo, para determinar la naturaleza de la instrucción y decidir la acción a tomar. Sin embargo, debido a la gran cantidad de tiempo que se desperdicia al esperar por operaciones I/O, el procesador se emplea de manera más eficiente con el uso de interrupciones.



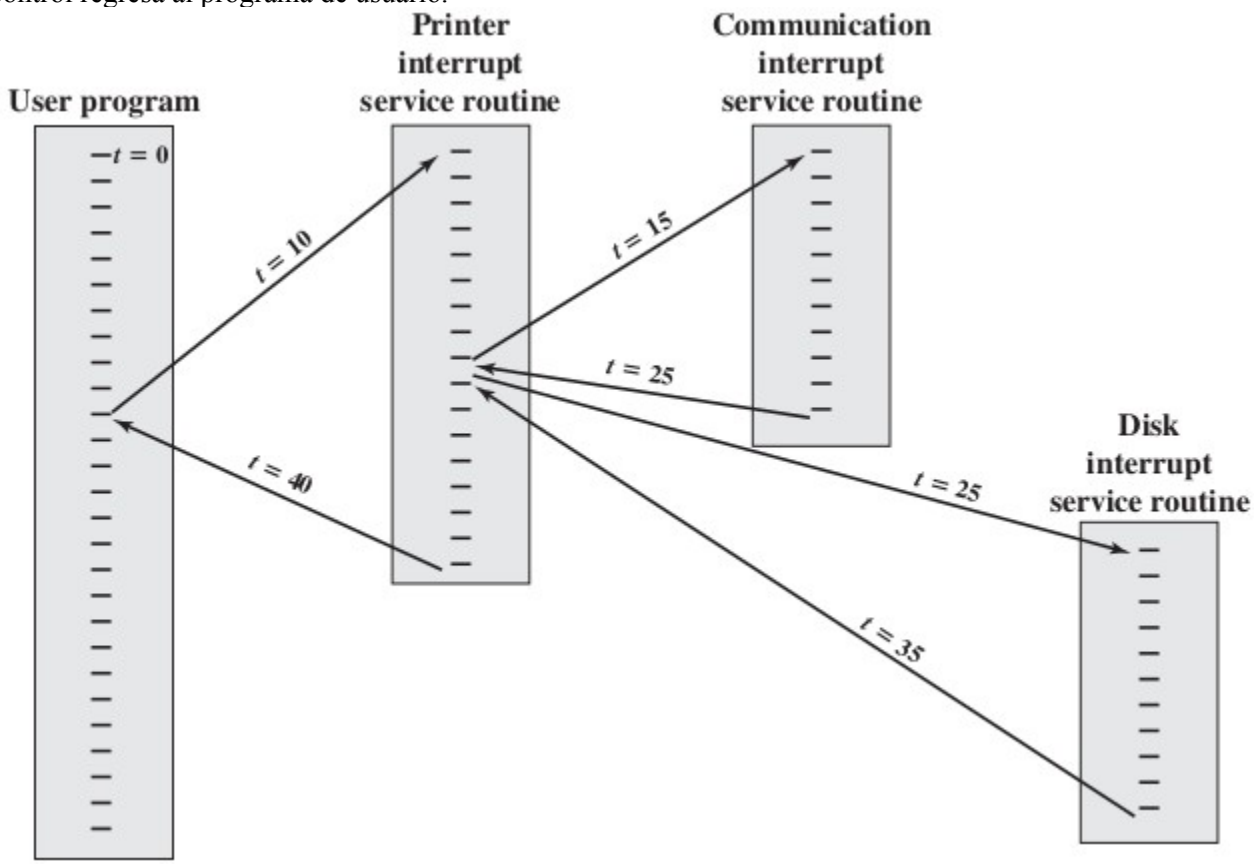
Una pregunta que podría surgir es, ¿qué pasa si ocurren dos o más interrupciones al mismo tiempo? O ¿qué pasa si ocurre una interrupción mientras se ejecuta el manejador de interrupciónes de otra? Existen dos enfoques. El primero consiste en deshabilitar las interrupciones cuando se está ya procesando una interrupción. *Deshabilitar interrupciones* simplemente significa que el procesador ignorará la señar de petición de interrupción. Si ocurre una interrupción adicional, ésta se mantendrá pendiente y será atendida cuando termine de ejecutarse el primer manejador de interrupciones. Así, cuando se ejecuta un programa de usuario y ocurre una interrupción, las interrupciones se deshabilitan inmediatamente. Este enfoque es bastante sencillo y las interrupciones se atienden en orden secuencial, el problema es que no toma en cuenta la prioridad de las interrupciones y, usualmente, los dispositivos externos tienen diferentes prioridades de acuerdo a su función.



(a) Sequential interrupt processing

Un segundo enfoque consiste en definir prioridades para las interrupciones y permitir que una interrupción de alta prioridad cause que el manejador de interrupciones de una interrupción de baja prioridad sea, a su vez, interrumpido. Considere el ejemplo de la figura siguiente en el cual se tienen 3 dispositivos: una impresora (prioridad 2), un disco (prioridad 4) y una línea de comunicaciones (prioridad 5). Un programa de usuario comienza en *t=0* y en *t=10* ocurre una interrupción de la impresora; se guarda la información del programa de usuario en el stack y la ejecución continúa con el ISR (rutina de servicio de interrupción, *Interruption Service Routine*) de la impresora. Mientras se ejecuta esta rutina, en *t=15,* ocurre una interrupción

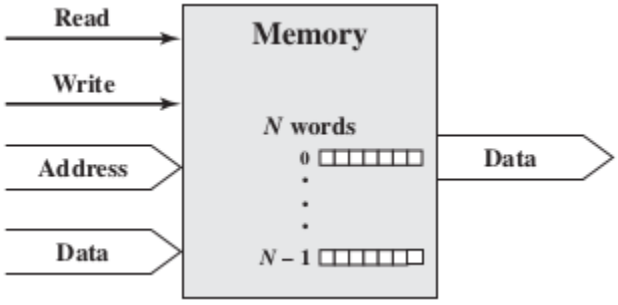
de la línea de comunicación, como este dispositivo tiene una prioridad mayor a la impresora, entonces se atiende la interrupción. Se interrumpe el ISR de la impresora (se guarda su contexto) y comienza el ISR de la línea de comunicaciones. Posteriormente, en *t=20* ocurre una interrupción de disco, al tener una menor prioridad que la línea de comunicaciones, la interrupción se ignora y se mantiene pendiente. Cuando termina el ISR de comunicación *(t=25),* se reestablece el último contexto guardado (ISR de la impresora). Sin embargo, antes de que incluso una simple instrucción se ejecute, el procesador atiende la interrupción generada por el disco que tiene una prioridad mayor a la de la impresora. Así que el contexto se vuelve a guardar y se inicia el ISR del disco. Es hasta que rutina termina en *t=35* que se reinicia el ISR de la impresora. Cuando éste termina en *t=40*, el control regresa al programa de usuario.

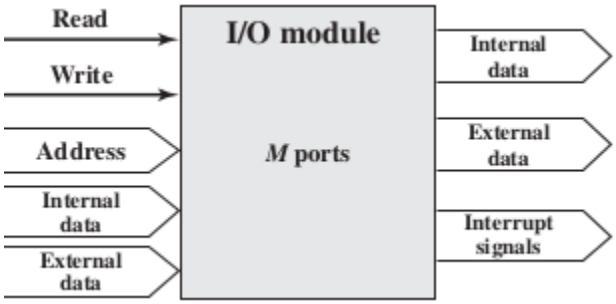


3.3 Estructuras de Interconexión

Los caminos que conectan los diversos módulos que componen una computadora se conocen como la *estructura de interconexión*. El diseño de esta estructura dependerá de los intercambios que deban ocurrir entre los módulos:

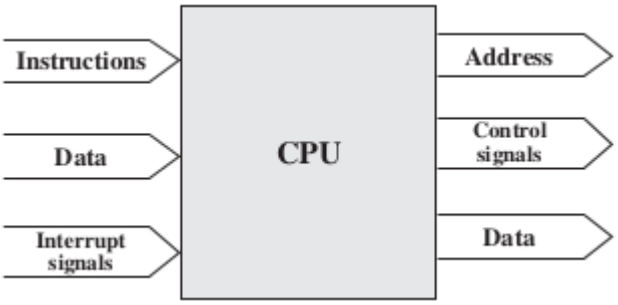
- **Memoria:** típicamente la memoria consiste de N palabras (localidades del tamaño de la palabra) de la misma longitud. A cada localidad se le asigna una dirección numérica (0, ... , N-1). Una palabra de datos puede ser leída o escribida en la memoria. La naturaleza de la operación se indica por una señal de control leer *(read)* o escribir *(write)*. La localización de la operación se define por una dirección.





- **Módulos I/O:** visto desde una perspectiva interna, un módulo I/O es similar a la memoria. Existen dos operaciones, leer y escribir. Además, un módulo I/O puede controlar más de un dispositivo externo. Así que podemos identificar cada dispositivo externo con un *puerto* y asignarle una dirección única (0, ... , M-1). Adicionalmente, existen caminos de datos externos para la entrada y salida de datos. Finalmente, los dispositivos I/O son capaces de enviar interrupciones al procesador.

- **Procesador:** el procesador lee instrucciones y datos, escribe datos después de procesarlos y utiliza señales de control para administrar la operación global del sistema. También recibe señales de interrupción.



En las figuras, las flechas gruesas representan líneas múltiples de señal que llevan múltiples bits de información en paralelo; las flechas delgadas representan una línea simple de señal.

De esta manera, la estructura de interconexión deberá de soportar los siguientes tipos de transferencias: memoria a procesador, procesador a memoria, I/O a procesador, procesador a I/O, I/O a memoria y memoria a I/O. A través de los años se han implementado diversas estructuras, llas más comúnes son la estructura de bus y bus múltiple y la interconexión punto a punto con transferencia de datos mediante paquetes.

3.4 Interconexión por bus

Un bus es un camino de interconexión que conecta dos o más dispositivos. Una característica clave es que es un medio de transmisión compartido: múltiples dispositivos se conectan al bus y una señal transmitida por uno de ellos está disponible para ser recibida por todos los demás. Si dos dispositivos intentan transmitir al mismo tiempo, las señales se mezclarán y dañarán. Así, sólo un dispositivo puede transmitir a la vez.

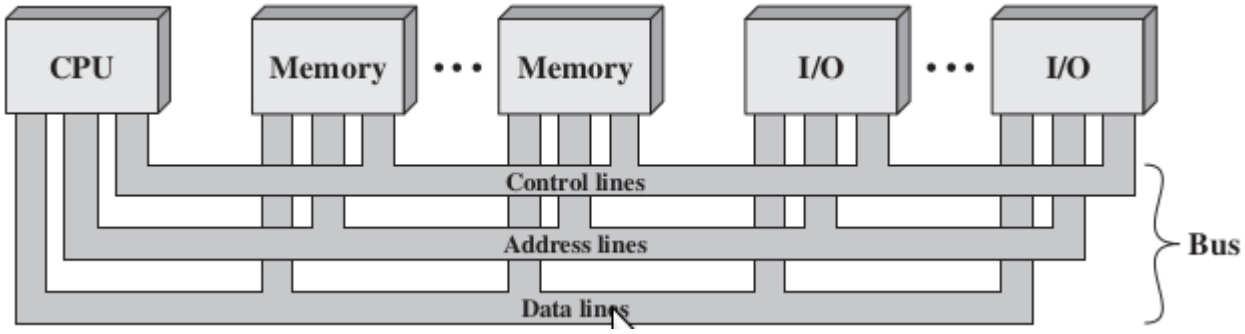
Típicamente, un bus consiste de múltiples caminos de comunicación o líneas, cada línea puede transmitir señales que representan 0 o 1. Si se utiliza una sola línea, una serie de dígitos binarios puede ser transmitida en cierto tiempo; pero si se utilizan varias líneas un bus puede transmitir la misma serie en paralelo. Por ejemplo, una palabra de 8 bits puede ser transmitida en paralelo en un bus de 8 líneas.

Las computadoras contienen un número diferente de buses que proveen caminos entre los componentes a varios niveles de la jerarquía del sistema. El bus que conecta los componentes principales (procesador, memory, I/O) se conoce como bus de sistema.

Estructura del bus

Un bus consiste de entre 50 o hasta cientos de líneas individuales donde cada una tiene asignada una función en particular. Los 3 grupos funcionales principales son: líneas de datos, de direcciones y de control. Adicionalmente, pueden existir líneas de poder que energizan a los módulos.

Las **líneas de datos** proveen un camino para mover los datos entre los componentes, colectivamente se les conoce como *bus de datos*. El bus de datos consiste de 32, 64, 128 o incluso más lineas separadas, a este número se le conoce como el *ancho de bus*. El ancho de bus es un factor clave para determinar el desempeño del sistema, por ejemplo, si el bus de datos tiene un ancho de 32 bits y cada instrucción es de 64 bits, entonces el procesador necesitará accesar al módulo de memoria 2 veces para obtener cada instrucción.



Las **líneas de dirección** se utilizan para designar el origen o destino de los datos que se encuentran en el bus de datos. Por ejemplo, si se desea leer una determinada palabra desde la memoria, se pone la dirección de la localidad de memoria en las líneas de dirección. Debe ser obvio, que el ánchо del *bus de direcciones* determina la máxima capacidad de memoria del sistema. Además, las líneas de dirección son utilizadas también para accesar a los puertos I/O.

Las **líneas de control** se utilizan para controlar el acceso y el uso de las líneas de datos y direcciones. Como las líneas son recursos compartidos por todos los componentes, debe existir un método para controlar su uso. Las señales de control transmiten tanto información de comando como de sincronización. Líneas típicas de control incluyen:

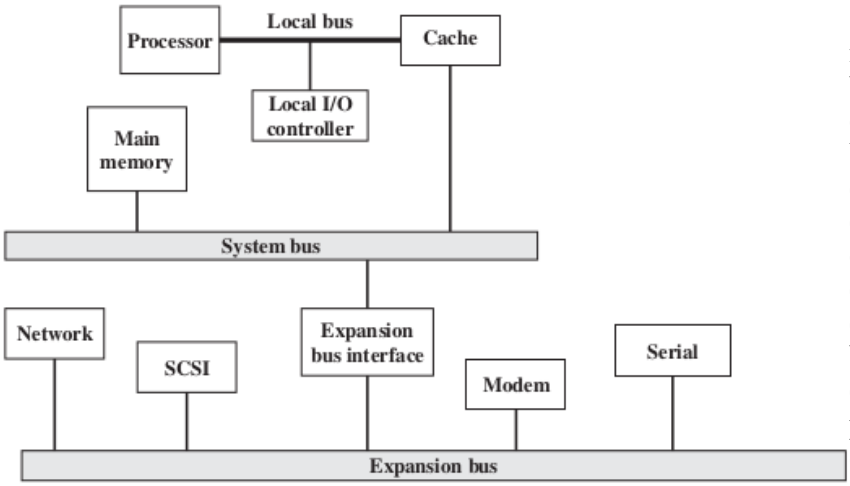
- Escribir a memoria: toma la información del bus de datos y la escribe en la localidad indicada por la dirección del bus de direcciones.
- Leer de memoria: toma la información de la localidad de memoria indicada en el bus de direcciones y la pone en el bus de datos.
- Escribir a I/O: toma la información del bus de datos y la entrega como salida al puerto I/O especificado.
- Leer de I/O: toma la información del puerto I/O especificado y la pone en el bus de datos.
- Acuse de recibo de transferencia: indica que la información ha sido aceptada del bus o puesta en el bus.
- Pedir bus: indica que un módulo requiere el control del bus.
- Otorgar bus: indica que se ha otorgado el control del bus a uno de los módulos.
- Petición de interrupción: indica que existe una interrupción pendiente.
- Acuse de recibido de interrupción: indica que una interrupción pendiente ha sido atendida.
- Reloj: se utiliza en operaciones de sincronización.
- Reset: inicializa todos los módulos.

Si un módulo desea enviar datos a otro, debe hacer dos cosas: (1) obtener el control de bus y (2) transferir los datos a través de él. Si un módulo desea obtener datos de otro, debe: (1) obtener control del bus, (2) transferir una petición de datos al otro módulo y (3) esperar que el otro módulo transifera los datos.

Jerarquías de bus múltiple

Si un gran número de dispositivos están conectados al bus, el desempeño bajará por 2 causas principales:

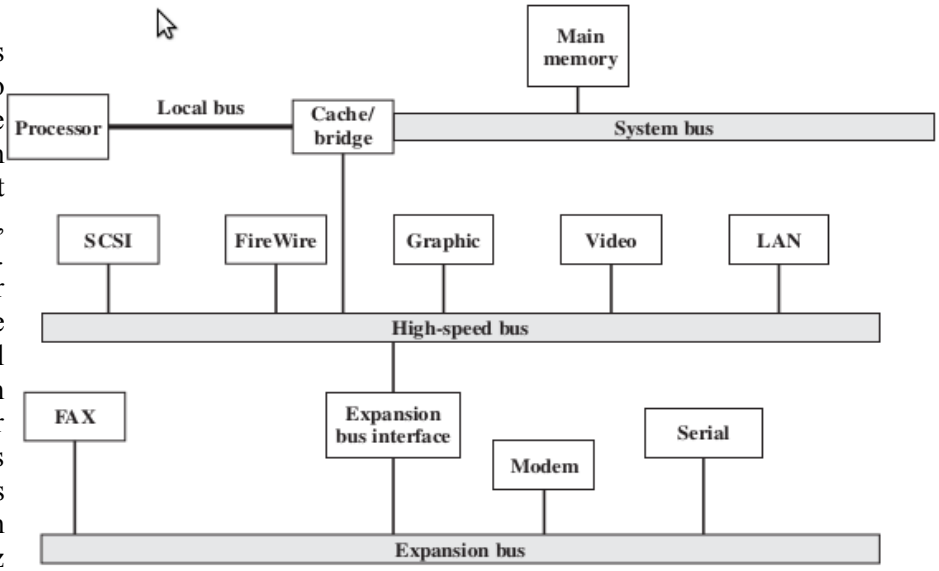
1. En general, mientras más dispositivos existan, mayor deberá ser la longitud del bus y, por tanto, mayor el retraso debido a la propagación. Este retraso determina el tiempo que le toma a los dispositivos coordinarse para usar el bus. Cuando el control del bus cambia frecuentemente, el retraso por propagación puede afectar notablemente el desempeño.
2. El bus se convierte en un cuello de botella conforme la demanda de transferencia de datos se acerca a la capacidad del bus. Este efecto se puede contrarrestar de cierta manera incrementando el ancho del bus. Sin embargo, puesto que la tasa de transferencia de datos generada por los dispositivos como los controladores gráficos o las interfaces de red están creciendo rápidamente, un solo bus será incapáz de solventar el problema.



Por las razones previas, la mayoría de las computadoras utilizan buses múltiples que tienen una determinada jerarquía entre sí. Usualmente, existe un bus local que conecta al procesador con su memoria caché y que puede soportar uno o dos dispositivos locales adicionales. El controlador de la memoria cache se conecta no sólo a este bus local, sino al bus de sistema al cual se encuentran conectados los módulos de la memoria principal. Al usar memoria caché, se evita que el procesador tenga que accesar a la memoria principal frecuentemente y así se puede conectar la memoria principal al bus del sistema en lugar del bus local. Así, las transferencias entre I/O y memoria no interfieren con la actividad del procesador.

Es posible conectar los controladores de los módulos I/O directamente al bus de sistema pero una solución más eficiente es utilizar uno o más buses de expanción para ello. Una interfaz de bus de expansión amortigua las transferencias de datos entre el bus de sistema y los controladores I/O. Este arreglo permite que el bus del sistema soporte una gran variedad de dispositivos I/O y al mismo tiempo aisla el trafico memoria-procesador del tráfico I/O.

La arquitectura anterior es razonablemente eficiente pero comienza a fallar con el uso de dispositivos I/O que requieren un desempeño mayor (ej. Fast Ethernet a 100Mbps, controladores de video y gráficos). La respuesta a ésto fue construir un bus de alta velocidad que se encuentra integrado al resto del sistema y requiere únicamente un puente entre el bus del procesador y el bus de alta velocidad. Los dispositivos con velocidades menores son soportados con un bus de expansión y una interfaz que amortigua el tráfico entre el bus de expansión y el bus de alta velocidad. La ventaja de este arreglo es que el bus de alta velocidad provee una mejor integración para los dispositivos I/O de alta demanda con el procesador



y, al mismo tiempo, es independiente del procesador: cambios en la arquitectura del procesador no afectan al bus de alta velocidad y viceversa.

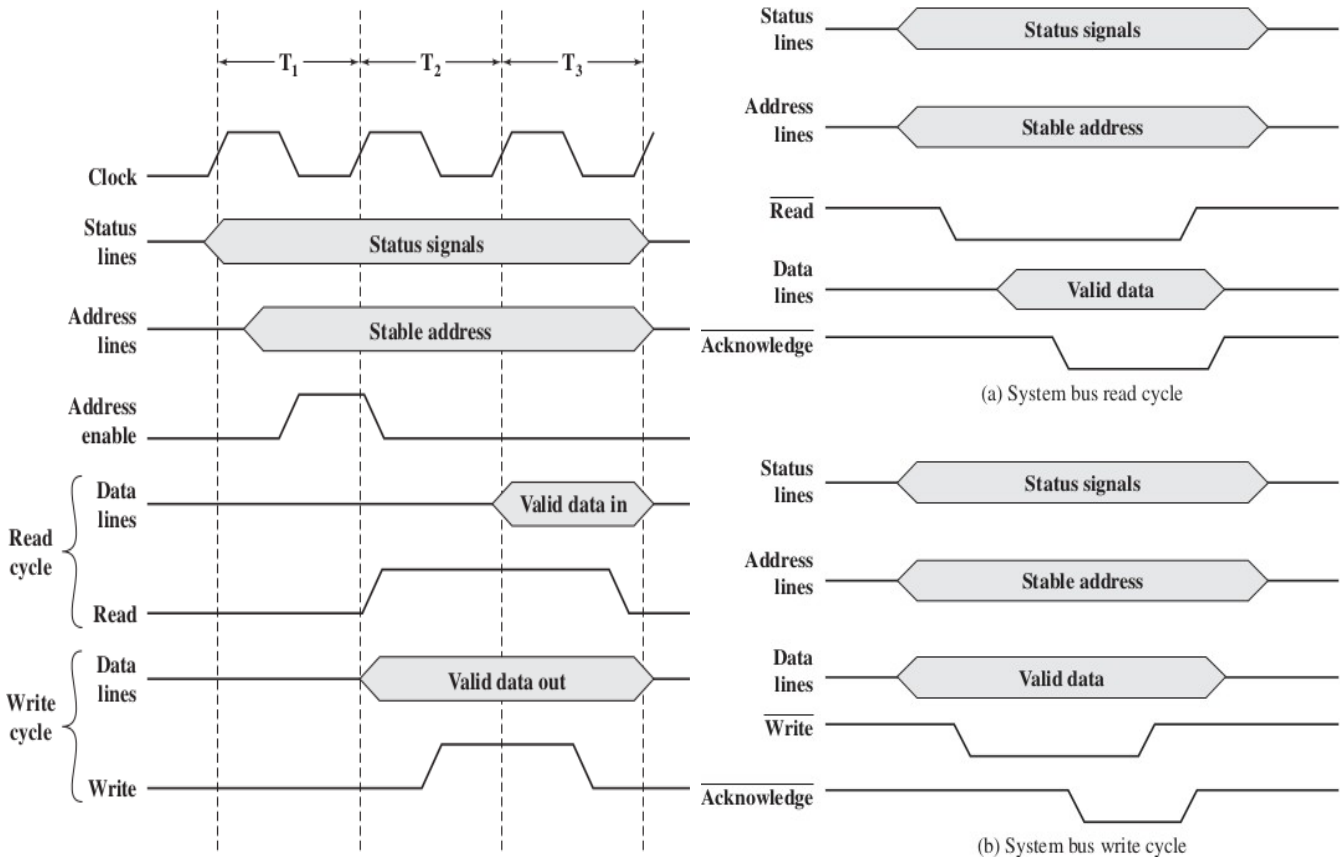
Elementos del diseño de bus

Existen algunos parámetros básicos o elementos de diseño que sirven para diferenciar los tipos de buses, entre ellos se encuentran los siguientes:

Tipo: dedicado o multiplexado. Un bus dedicado está asignado permanentemente a una función o a un subconjunto físico de componentes. Un ejemplo de bus dedicado son las líneas de direcciones y datos que son comunes en los buses. Por otro lado, si las mismas líneas se utilizan para distintas funciones, datos y direcciones por ejemplo, se dice que el bus es multiplexado. Las ventajas del multiplexado son que se utilizan menos líneas y, por lo tanto, se reduce el espacio y el costo. Las desventajas son que se necesitan circuitos más complejos y puede haber una reducción del desempeño. La dedicación física se refiere a utilizar buses que conectan sólo algunos módulos entre sí. Un ejemplo es el bus I/O que conecta todos los módulos I/O. El bus I/O es después conectado al bus principal mediante algún tipo de adaptador. Las ventajas son aumento del desempeño pues existe menos competencia por el bus, pero aumenta el costo y el tamaño del sistema.

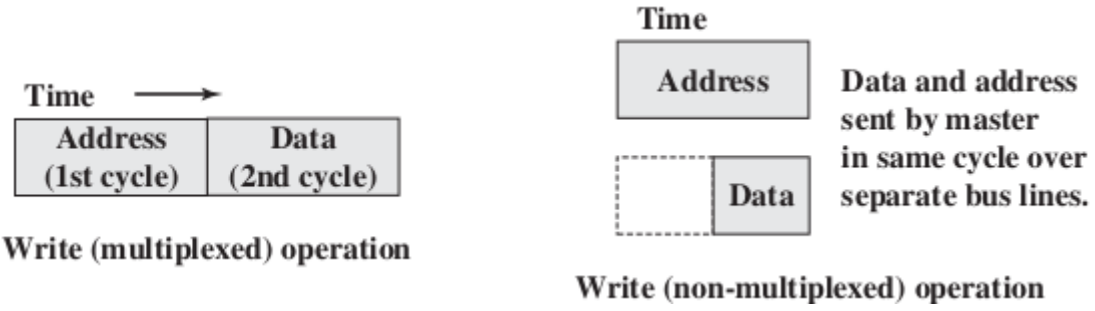
Método de arbitraje: centralizado o distribuído. Debido a que más de un módulo necesita el control del bus y solamente un dispositivo a la vez puede transmitir por él, es necesario un método para decidir a quién se le otorga el control del bus. A dicho método se le conoce como arbitraje y existen varias implementaciones, pero todas se pueden clasificar en 2 vertientes: centralizado y distribuído. En un sistema centralizado, un dispositivo único, llamado controlador o árbitro, es responsable de tomar la decisión. En un sistema distribuído, no existe un controlador central. Cada módulo tiene lógica de control de acceso y los módulos trabajan juntos para determinar quién tiene control del bus. Al módulo en control del bus se le llama maestro y al módulo con el que interactúa se le llama esclavo.

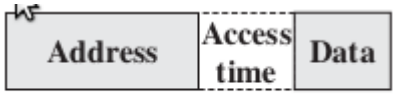
Temporización: síncrono o asíncrono. La temporización se refiere a la manera en que los eventos se coordinan en el bus. Con un **sistema síncrono**, la ocurrencia de eventos está determinada por un reloj. El bus incluye una línea de reloj, por la cual, un reloj transmite una secuencia alternante de 1s y 0s de igual duración. Una transmisión 1-0 se conoce como *ciclo de reloj* y define un espacio de tiempo. Todos los eventos comienzan al inicio de un ciclo de reloj. En un **sistema asíncrono**, la ocurrencia de un evento en el bus depende de la ocurrencia de un evento previo. Un sistema síncrono es más simple de implementar y probar, sin embargo, es menos flexible que uno asíncrono. Un sistema asíncrono puede aprovechar las ventajas que proveen los avances en el desempeño de los dispositivos: dispositivos viejos y nuevos pueden compartir el bus.



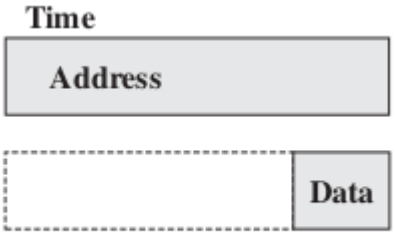
Ancho: de direcciones y de datos. Mientras más ancho sea el bus de datos, más bits se pueden trasferir a la vez. El ancho del bus de direcciones impacta la capacidad del sistema: mientras más ancho, más grande el rango de localidades que pueden ser referenciadas.

Tipo de transferenica de datos: escribir, leer, bloquear. Los buses soportan distintos tipos de transferencias de datos. Todos los buses soportan las operaciones de escritura (maestro a esclavo) y lectura (esclavo a maestro). Dichas operaciones se implementan de forma distinta dependiendo del tipo de bus con el que se trate, dedicado o multiplexado. Para un bus multiplexado, el bus se utiliza inicialmente para especificar la dirección y posteriormente para transferir los datos. En una operación de lectura, existe un tiempo de espera mientras los datos son obtenidos por el esclavo para ser puestos en el bus. En el caso de buses de dirección y datos dedicados, la dirección se pone en el bus de direcciones y se mantiene ahí. Para una operación de escritura, el maestro pone los datos en el bus de datos tan pronto como la dirección se ha estabilizado y el esclavo ha tenido oportunidad de reconocer su dirección. Para una operación lectura, el esclavo pone los datos en el bus tan pronto como ha reconocido su dirección y obtenido la información.





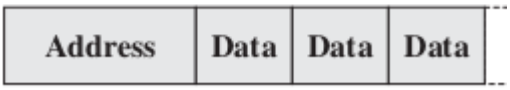
Read (multiplexed) operation



Read (non-multiplexed) operation

Read-modify-write operation

Read-after-write operation



Block data transfer

Existen además, varias combinaciones de operaciones. Una operación leer-modificar-escribir es simplemente una operación de lectura seguida inmediatamento por una escritura a la misma dirección. La operación es indivisible para evitar cualquier acceso a los datos por otros usuarios potenciales.

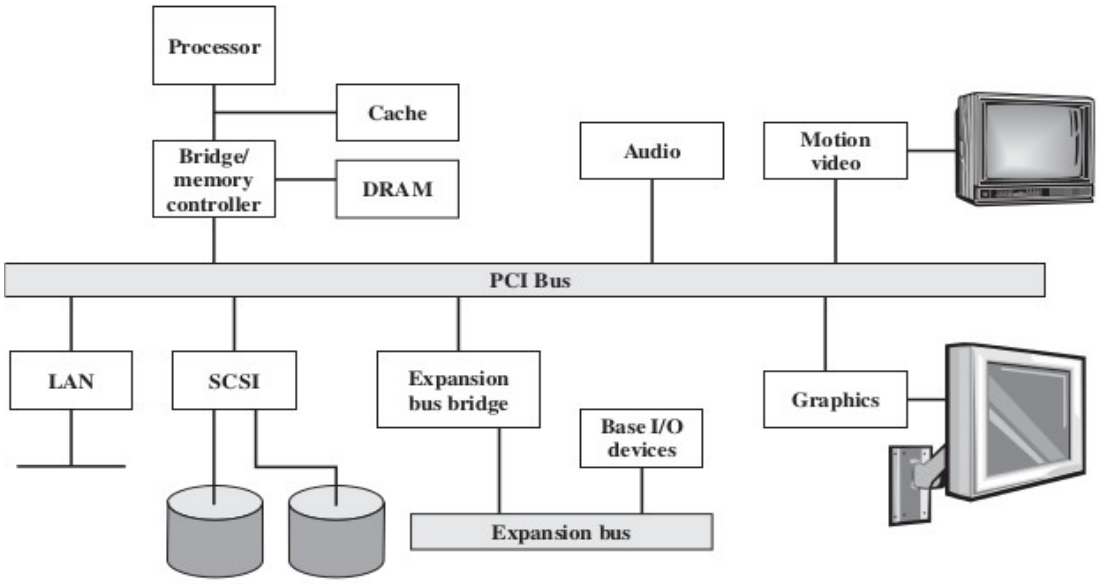
Leer-después de-escribir es una operación indivisible que consiste en una escritura seguida de una lectura en la misma dirección.

Algunos buses permiten transferencias de datos en bloque. En estos casos, un ciclo de dirección se sigue de *n* ciclos de datos. El primer item de datos se transfiere hacia o desde la dirección especificada, el resto de los items de datos se trasnfieren a direcciones subsecuentes.

3.5 El Bus PCI

El bus de interconexión de componentes periféricos (PCI, *Peripheral Component Interconnect*) es un bus de alto ancho de banda, independiente del procesador que provee de un mejor desempeño para subsistemas I/O de alta velocidad (adaptadores gráficos, controladores de red, controladores de disco, etc). El estandard actual permite el uso de hasta 64 lineas de datos a 66 MHz, para una tasa de transferencia de datos de 528 MBytes/s o 4.224Gbps. Adicionalmente, está específicamente diseñado para, de manera económica, alcanzar los requerimientos de I/O de los sistemas modernos: requiere muy pocos chips y suporta la conexión de otros buses. PCI ha sido ampliamente adoptado y se utiliza en computadoras personales, estaciones de trabajo y servidores.

PCI está diseñado para soportar una variedad de configuraciones, incluyendo sistemas con uno o varios procesadores, por lo que proveé un conjunto de funciones de propósito general. Es un sistema síncrono (basado en reloj) y utiliza un sistema de arbitraje centralizado. En la figura siguiente se muestra el uso típico del bus PCI en un sistema con un solo procesador. El controlador DRAM y el puente PCI se combinan en una sola unidad para proveer el acoplamiento con el procesador y la habilidad de entregar datos a gran velocidad. El puente actúa como un buffer de datos de manera que la velocidad del bus PCI puede ser diferente a la capacidad I/O del procesador.



(a) Typical desktop system

Estructura del bus PCI

El bus PCI puede ser configurado como un bus de 32 o 64 bits. Existen 49 líneas de señal obligatorias, las cuales se dividen en los siguientes grupos de acuerdo a su función:

- **Pins del sistema:** incluyen el reloj y el reset.
- **Pins de direcciones y de datos:** incluyen 32 líneas que son multiplexadas para direcciones y datos. Las otras líneas en este grupo se utilizan para interpretar y validar las líneas que llevan las direcciones/datos.
- **Pins de control de interfaz:** controlan la sincronización de las transacciones y proveen cordinación entre los iniciadores (maestros) y objetivos (esclavos).
- **Pins de arbitraje:** al contrario del resto de las líneas, estas líneas no son compartidas. Cada maestro PCI tiene su propio par de líneas de arbitraje que lo conectan directamente con el árbitro del bus PCI.
- **Pins de reporte de errores:** usadas para reportar errores de paridad, entre otros.

Adicionalmente, la especificación del PCI define 51 lineas de señal opcionales, divididas en los siguientes grupos de acuerdo a su función:

- **Pins de interrupción:** se proveen a los dispositivos PCI que deban generar peticiones de interrupción. Como los pines de arbitraje, estas lineas no son compartidas. Cada dispositivo PCI tiene su propia línea de interrupción que lo conecta con un controlador de interrupciones.
- **Pins de soporte caché:** se utilizan para soportar una memoria en el PCI que puede ser utilizada como caché para el procesador u otro dispositivo.
- **Pins de extensión a 64 bits:** incluyen 32 líneas adicionales multiplexadas para direcciones y datos que se combinan con las líneas obligatorias para formar un bus de direcciones/datos de 64 bits. Otras líneas en este grupo se utilizan para interpretar y validar las líneas de señal que llevan las direcciones/datos. Finalmente, hay dos líneas que permiten a 2 dispositivos PCI acordar el uso de 64 bits.

Designation	Type	Description
System Pins		
CLK	in	Provides timing for all transactions and is sampled by all inputs on the rising edge. Clock rates up to 33 MHz are supported.
RST#	in	Forces all PCI-specific registers, sequencers, and signals to an initialized state.
Address and Data Pins		
AD[31::0]	t/s	Multiplexed lines used for address and data
C/BE[3:0]#	t/s	Multiplexed bus command and byte enable signals. During the data phase, the lines indicate which of the four byte lanes carry meaningful data.
PAR	t/s	Provides even parity across AD and C/BE lines one clock cycle later. The master drives PAR for address and write data phases; the target drive PAR for read data phases.
Interface Control Pins		
FRAME#	s/t/s	Driven by current master to indicate the start and duration of a transaction. It is asserted at the start and deasserted when the initiator is ready to begin the final data phase.
IRDY#	s/t/s	Initiator Ready. Driven by current bus master (initiator of transaction). During a read, indicates that the master is prepared to accept data; during a write, indicates that valid data are present on AD.
TRDY#	s/t/s	Target Ready. Driven by the target (selected device). During a read, indicates that valid data are present on AD; during a write, indicates that target is ready to accept data.
STOP#	s/t/s	Indicates that current target wishes the initiator to stop the current transaction.
IDSEL	in	Initialization Device Select. Used as a chip select during configuration read and write transactions.
DEVSEL#	in	Device Select. Asserted by target when it has recognized its address. Indicates to current initiator whether any device has been selected.
Arbitration Pins		
REQ#	t/s	Indicates to the arbiter that this device requires use of the bus. This is a device-specific point-to-point line.
GNT#	t/s	Indicates to the device that the arbiter has granted bus access. This is a device-specific point-to-point line.
Error Reporting Pins		
PERR#	s/t/s	Parity Error. Indicates a data parity error is detected by a target during a write data phase or by an initiator during a read data phase.
SERR#	o/d	System Error. May be pulsed by any device to report address parity errors and critical errors other than parity.

Comandos PCI

La actividad en el bus ocurre en la forma de transacciones entre un iniciador (maestro) y un objetivo (esclavo). Cuando un maestro adquiere el control del bus, determina el tipo de transacción que ocurrirá. Durante la fase de dirección de la transacción, se utilizan las líneas C/BE (bus Command and Byte Enable) para indicar el tipo de transacción. Existen los siguientes comandos:

- Reconocimiento de Interrupción (*Interrupt Acknowledge*)
- Ciclo Especial
- Lectura I/O
- Escritura I/O
- Lectura de Memoria
- Lectura de Línea de Memoria
- Lectura Múltiple de Memoria
- Escritura de Memoria
- Escritura de Memoria e Invalidación

- Lectura de Configuración
- Escritura de Configuración
- Ciclo de Dirección Dual

El Interrupt Acknowledge es un comando utilizado por el controlador de interrupciones del bus PCI. Las líneas de dirección no se utilizan durante la fase de direcciones y las BE indican el tamaño del identificador de interrupción a regresar.

El comando Ciclo Especial Special Cycle es utilizado por el iniciador para transmitir un mensaje a uno o más objetivos.

Los comandos de lectura y escritura I/O se utilizan para transferir datos entre el iniciador y un controlador I/O. Cada dispositivo I/O tiene su propio espacio de direcciones, las lineas de dirección se utilizan para indicar un dispositivo en particular y, posteriormente, para especificar los datos a ser transferidos.

Los comandos de lectura y escritura de memoria se utilizan para especificar la transferencia de datos, que puede ocupar uno o más ciclos de reloj. El comando escritura de memoria e invalidación transfiere datos en uno o más ciclos a memoria y garantiza que al menos una linea de caché se escriba.

Read Command Type	For Cachable Memory	For Noncachable Memory
Memory Read	Bursting one-half or less of a cache line	Bursting 2 data transfer cycles or less
Memory Read Line	Bursting more than one-half a cache line to three cache lines	Bursting 3 to 12 data transfers
Memory Read Multiple	Bursting more than three cache lines	Bursting more than 12 data transfers

Los dos comandos de configuración permiten que un maestro lea y actualize los parámetros de configuración de un dispositivo conectado al PCI. Cada dispositivo PCI puede incluír hasta 256 registros internos que se utilizan para configurarlo. El comando de ciclo de dirección dual se usa por un iniciador par indicar que utiliza direccionamiento de 64 bits.

Transferencias de datos

Cada transferenica de datos en el bus PCI es una transacción única, que consiste en una fase de dirección y una o más fases de datos. A continuación se describe una transacción de lectura. Todos los eventos se sincronizan con las transiciones de caída del reloj, que ocurren en el medio de cada ciclo de reloj. Los dispositivos muestrean al bus al inicio de cada ciclo de reloj (subida).

a. Una vez que un maestro gana el control del bus, indica el inicio de una transacción activando la línea FRAME. Dicha línea continúa activa hasta que el iniciador va a compleatar la última fase de datos. El iniciador también pone la dirección inicial en el bus de dirección (AD) y el comando de lectura en las líneas C/BE.

b. Al inicio del ciclo 2, el dispositivo objetivo reconocerá su dirección en las líneas AD.

c. El iniciador deja el mando del bus AD. Se requiere un ciclo de intercambio cuando hay lineas que se deban manejar por más de un dispositivo. Así, la caída de la señal de dirección prepara al bus para ser utilizado por el dispositivo objetivo. El iniciador utiliza ahora las líneas C/BE para identificar cuáles líneas AD se utilizarán para la transmisión (de 1 a 4 bytes). El iniciador activa IRDY para indicar que está listo para recibir el primer item de datos.

d. El dispositivo objetivo seleccionado activa DEVSEL para indicar que ha reconocido su propia dirección y que dará una respuesta. Pone los datos solicitados en AD y activa TRDY para indicar que datos válidos están presentes en el bus.

e. El iniciador lee los datos al inicio del ciclo 4 y cambia las líneas C/BE en preparación para la lectura siguiente.

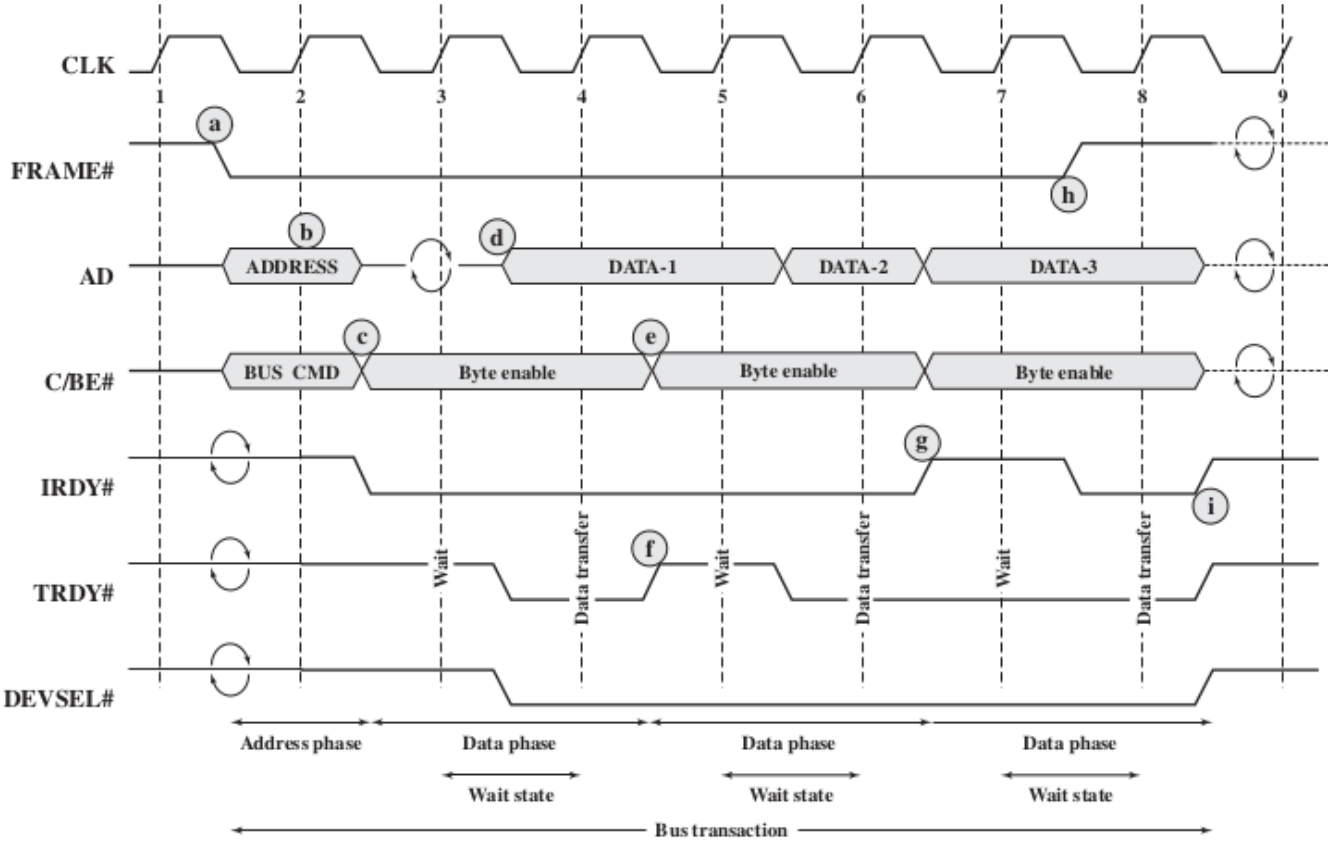
f. En este ejemplo, el objetivo necesita de un tiempo para preparar el segundo bloque de datos. Por lo tanto, desactiva la señal TRDY para indicar al iniciador que no habrá nuevos datos durante el siguiente ciclo. En respuesta, el iniciador no lee las lineas de datos al comienzo del quinto ciclo de reloj y no modifica las líneas BE. El bloque de datos se lee al inicio del ciclo 6.

g. Durante el ciclo 6, el objetivo pone el tercer bloque de datos en el bus. Sin embargo, en este ejemplo, el iniciador no se encuentra listo para leerlo, entonces desactiva la línea IRDY. Lo anterior ocasiona que el objetivo mantenga los mismos datos en el bus por un ciclo extra.

h. El iniciador sabe que el tercer bloque de datos es el último y por ello desactiva FRAME para indicar al

objetivo que ésta será la última transferencia. También activa IRDY para indicar que está listo para completar la transferencia.

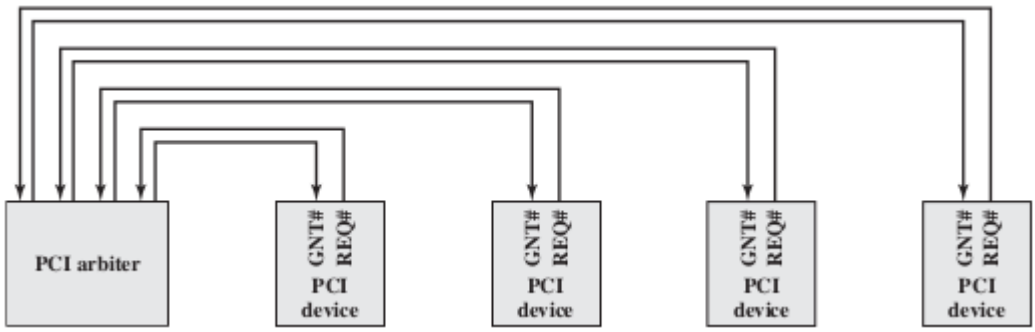
i. El iniciador desactiva IRDY, con lo cual, el bus regresa a su estado de inactivadad. A su vez, el objetivo desactiva TRDY y DEVSEL.



Arbitraje

PCI utiliza un esquema de arbitraje centralizado en el cual cada maestro tiene dos líneas directas de conexión con el árbitro, una para pedir el bus (REQ) y otra para otorgarlo (GNT). Un simple intercambio de señales de petición y otorgamiento es suficiente para tener acceso al bus.

La especificación del bus PCI no menciona ningún algoritmo de arbitraje en particular. El árbitro puede utilizar un enfoque FCFS (*First-Come-First-Seved*), *Round-Robin* o cualquier otro esquema de prioridades. Cada dispositivo maestro deberá arbitrar por el control del bus para cada transacción que desee efectuar, donde una transacción consiste de una fase de dirección seguida de una o más fases de datos continuas.



En la figura siguiente se ejemplifica el proceso de arbitraje entre dos dispositivos maestros, A y B, la siguiente secuencia ocurre;

a. En cierto momento, previo al inicio del ciclo de reloj 1, A activó su señal REQ. El árbitro muestrea dicha señal al inicio del ciclo 1.

b. Durante el ciclo 1, B solicita el uso del bus activando su señal REQ.

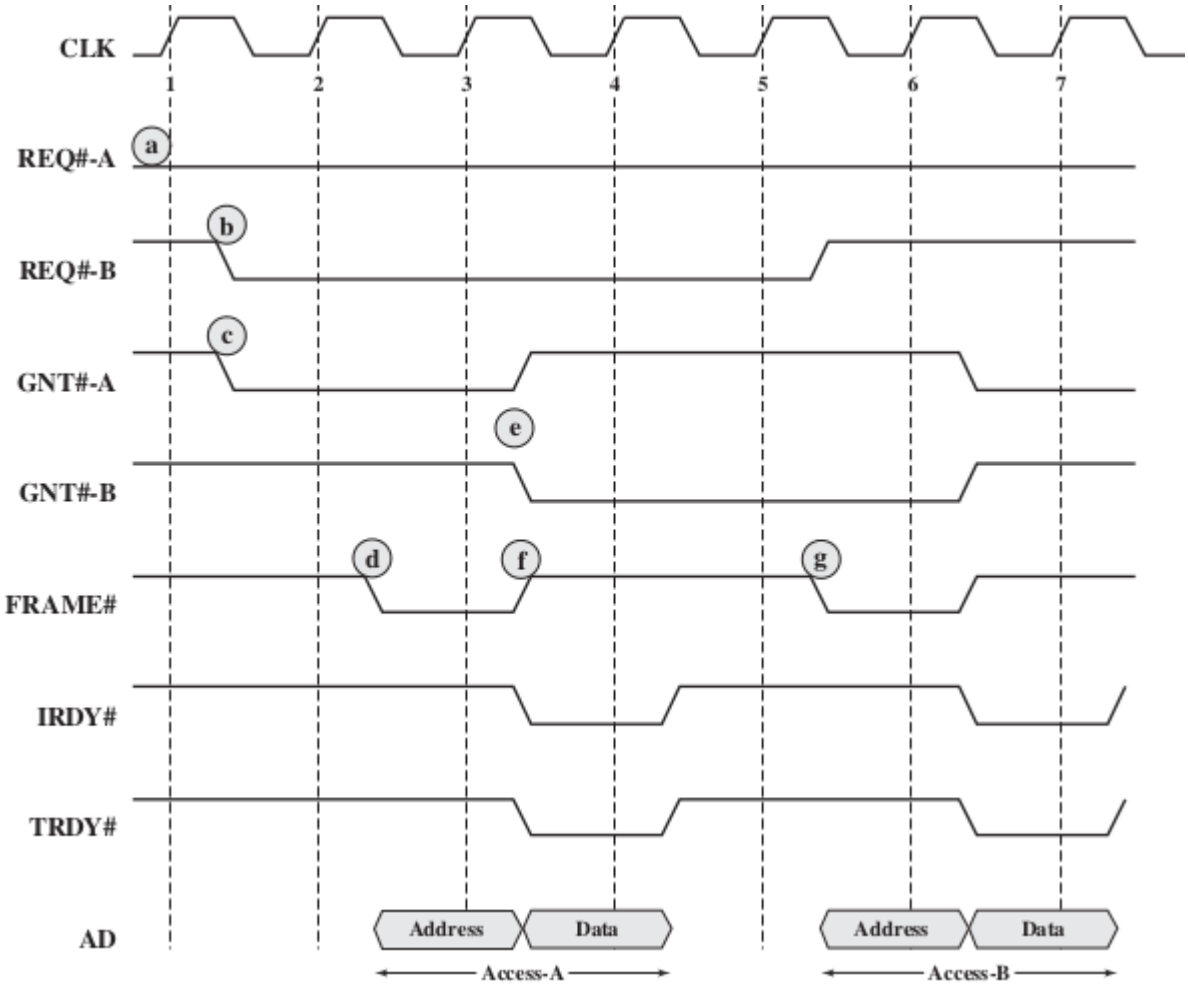
c. Al mismo tiempo, el árbitro activa GNT-A y da acceso al bus a A.

d. El dispositivo A muestrea GNT-A al inicio del ciclo 2 y se da cuenta que le han asignado acceso al bus. Sensa IRDY y TRDY y las encuentra desactivadas, indicando que el bus se encuentra inactivo. Entonces, activa FRAME, pone la dirección en AD y el comando en C/BE (no se muestra). Además, deja activa REQ-A pues necesita realizar una segunda transacción.

e. El árbitro sensa todas las líneas REQ al inicio del ciclo 3 y toma la decisión de dar control del bus al dispositivo B para la siguiente transacción. Entonces, activa GNT-B y desactiva GNT-A. Sin embargo, B no podrá utilizar el bus hasta que éste se encuentre inactivo.

f. A desactiva FRAME para indicar que la última (y única) transferencia de datos está en progreso. Pone los datos en AD e indica que se encuentra listo activando IRDY. El objetivo lee los datos al inicio del siguiente ciclo.

g. Al inicio del ciclo 5, B encuentra IRDY y FRAME desactivados y toma el control del bus activando FRAME. También desactiva REQ-B puesto que sólo requiere efectuar una transacción.



Posteriormente, se le otorga el control del bus al dispositivo A y así sucesivamente. Nótese que el proceso de arbitraje sucede de manera concurrente a la transferencia de datos entre maestro y esclavo. Por tanto, no se desperdician ciclos de reloj al realizar el arbitraje. A lo anterior se le denomina *arbitraje escondido.*

3.6 Interconexión Punto a Punto

La arquitectura de bus compartido fue el enfoque estandar para la interconexión entre el procesador y otros componentes por décadas. Pero los sistemas contemporáneos utilizan cada vez más la interconexión punto a punto en lugar de buses compartidos.

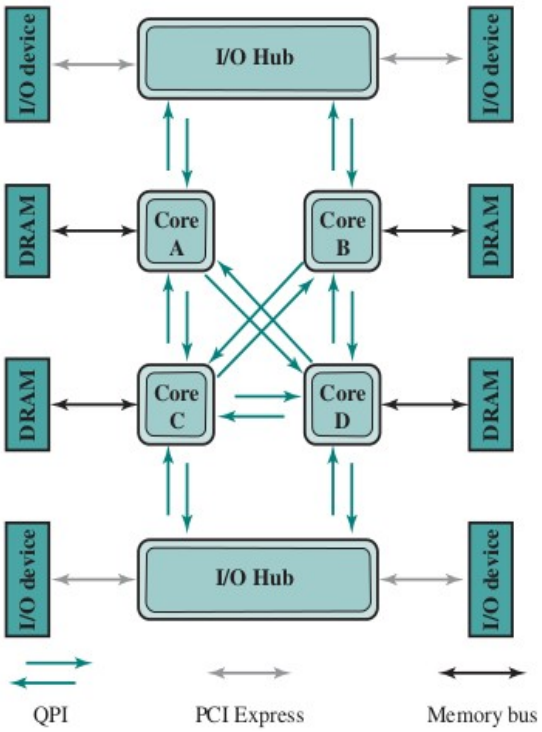
La razón principal detrás de este cambio son las restricciones eléctricas encontradas con el aumento de la frecuencia en los buses síncronos. A velocidades cada vez más altas, se vuelve incrementalmente más difícil realizar las funciones de arbitraje y sincronización necesarias. Además, con la llegada de los chips multi-núcleo, con múltiples procesadores y bastante memoria en un solo chip, se encontró que el uso de un bus compartido convencional en el mismo chip magnificaba las dificultades de incrementar la tasa de transferencia de datos y reducir la latencia del bus para mantenerse a la par de los procesadores.

Comparada con el bus compartido, la interconexión punto a punto tiene menor latencia, mayor velocidad de transferencia y mejor escalabilidad.

Un representante importante del enfoque punto a punto es el **QuickPath Interconnect (QPI)** de Intel, el cual fue presentado en 2008. Las siguientse son algunas características importantes de QPI y otros esquemas punto a punto:

- **Multiples conexiones directas:** múltiples componentes dentro del sistema gozan de un par directo de conexiones a otros componentes. Ésto elimina la necesidad del arbitraje encontrada en los medios compartidos.
- **Arquitectura de protocolos por capas:** como se halla también en los ambientes de red, como las redes de datos basadas en TCP/IP, estas interconexiones a nivel procesador utilizan una arquitectura de protocolos por capas, en lugar del uso simple de señales de control como en los medios compartidos.
- **Transferencias de datos por paquetes:** los datos no se envían como un flujo de datos puro. En lugar de eso, los datos se envían como una secuencia de paquetes, cada uno de los cuales incluye cabeceras de control y códigos de control de errores.

La figura siguiente muestra el uso típico de QPI en una computadora multi-núcleo. Los enlaces QPI forman una "tela" que permite a los datos moverse a través de la red. Conexiones QPI directas se pueden establecer entre cada par de núcleos. Si el núcleo A necesita accesar al controlador de memoria del núcleo D, puede enviar su petición a través de los núcleos B o C, los cuales a su vez deberán reenviar la petición al núcleo D, o puede utilizar su conexión directa con D.

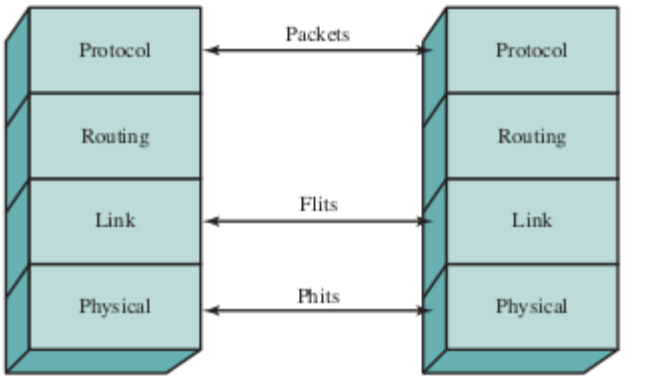


Adicionalmente, QPI se utiliza para conectarse a un módulo I/O, llamado Hub I/O (IOH) que actua como un switch para dirigir el tráfico desde y hacia los dispositivos I/O. Típicamente, en sistemas nuevos, el enlace entre el IOH y el dispositivo I/O utiliza un bus PCI Express (PCIe) que se describe en la sección siguiente. El IOH traduce entre los formatos de los protocolos QPI y los protocolos PCIe,

Cada núcleo se conecta con su módulo de memoria principal utilizando un bus de memoria dedicado.

QPI se define como una arquitectura de protocolos de cuatro capas;

1. **Física:** consiste de los cables que llevan las señales, así como la circuitería y lógica que dan soporte a las características necesarias para la transmición y recepción de los 0's y los 1's. La unidad de transferencia de la capa física es de 20 bits, los cuales son llamados **Phit** (physical unit).
2. **Enlace:** responsable de la transmisión confiable



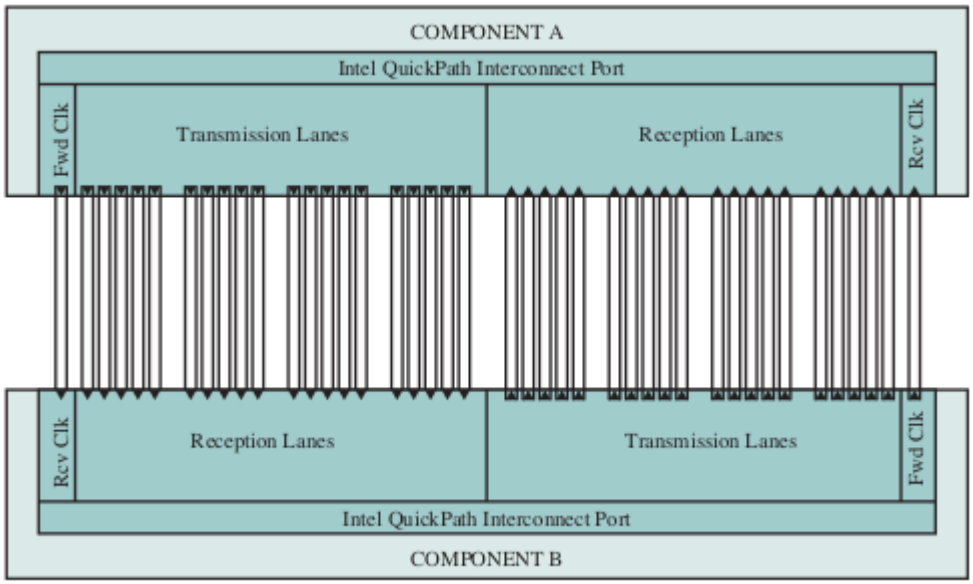
y el control de flujo. La unidad de transferencia de la capa de enlace es el **Flit** (flow control unit) de 80 bits.

3. **Enrutamiento:** provee la plataforma para dirigir los paquetes a través de la red.
4. **Protocolo:** el conjunto de reglas de alto nivel para intercambiar los **paquetes** de datos entre dispositivos. Un paquete se compone de un número integral de Flits.

Capa Física QPI

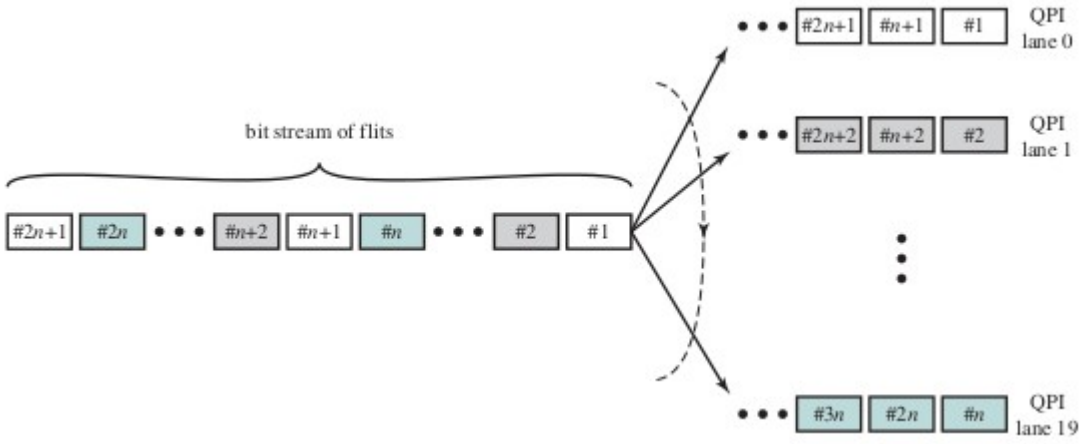
La figura siguiente muestra la arquitectura física de un puerto QPI. El puerto QPI consiste de 84 enlaces individuales que se agrupan como sigue: cada camino de datos consiste de un par de cables que transmiten un bit a la vez, a cada par se le llama canal. Existen 20 canales de datos en cada dirección (transmitir y recibir), además de un canal de reloj en cada dirección. Así, QPI es capaz de transmitir 20 (1 Phit) bits de datos en paralelo en cada dirección. Velocidades típicas de transferencia son de 16 GB/s, y como existen pares bidireccionales, la capacidad total es de 32 GB/s.

Los canales en cada dirección se agrupan en 4 cuadrantes de 5 canales cada uno. Para algunas aplicaciones, el enlace puede operar a un mitad o un cuarto de ancho para reducir el consumo de energía o trabajar a pesar de posibles fallas.



La forma de transmisión en cada canal se conoce como señalamiento diferencial o transimsión balanceada. Con transmisión balanceada, las señales se transmiten como una corriente que viaja por un conductor y regresa por el otro. El valor binario depende de la diferencia de voltaje. Típicamente, una línea tiene un voltaje positivo y la otra tiene un valor de voltaje igual a cero. Una línea está asociada con el valor binario 1 y la otra con el valor binario 0. Específicamente, la técnica utilizada por QPI se conoce como Señalamiento Diferencial a Voltaje Bajo (LVDS, Low-Voltage Differential Signaling). En una implementación típica, el transmisor inyecta una pequeña corriente a uno de los cables, dependiendo del nivel lógico a enviar. La corriente pasa a través del resistor en el receptor y entonces regresa al transmisor por el otro cable. El receptor sensa la polaridad del voltaje a través del resistor y determina el valor binario.

Otra función realizada por la capa física es que maneja la traducción entre los flits de 80 bits y los phits de 20 bits utilizando una técnica conocida como distribución multicanal. Los flits pueden ser considerados como un flujo de bits que está distribuído a través de los canales de datos (el primer bit en el primer canal, el segundo bit en el segundo canal, etc.



Capa de Enlace QPI

La capa de enlace QPI realiza dos funciones clave: el control de flujo y el control de errores. Estas funciones se efectúan como parte del protocolo de capa de enlace QPI y opera a nivel de flit. Cada flit consiste de un mensaje de 72 bits y un código de control de errores de 8 bits llamado CRC.

El contenido de un flit puede ser un mensaje de información o datos. Los flits de datos transmiten los bits entre los diferentes núcleos o entre un núcleo y el IOH. Los flits de mensaje son utilizados para las funciones de control de flujo, control de errores y coherencia de caché.

El control de flujo es necesario para asegurar que un transmisor QPI no abrume al receptor enviando datos más rápido de lo que el receptor puede procesar los datos y limpiar sus buffers para más datos entrantes. Para controlar el flujo de los datos, QPI hace uso de un esquema de créditos. Durante la inicialización, se le dan al transmisor un número de créditos para enviar flits a algún receptor. Cuando un flit es enviado, el transmisor decrementa en uno su cantidad de créditos. Cuando se libera un buffer en el receptor, se regresa un crédito al transmisor. Así, el receptor controla el ritmo al cual se transmiten los datos.

Ocasionalmente, un bit transmitido en la capa física cambia durante la transmisión, debido al ruido o algún otro fenómeno. El control de errores detecta y se recupera de tales errores de bit y así, aisla a las capas superiores de dichos errores. El procedimiento funciona de la manera siguiente:

1. Como se mencionó, cada flit contiene un campo CRC de 8 bits. El CRC es una función del valor de los 72 bits restantes. Al transmitir, se calcula el valor CRC para cada flit.
2. Cuando un flit es recibido, se vuelve a calcular su valor CRC para los 72 bits recibidos y se compara con el valor CRC recibido. Si los dos valores no coinciden, se ha detectado un error.
3. Cuando se detecta un error, se envía una petición al transmisor para retransmitir el flit que contiene el error. Sin embargo, como el transmisor probablemente ya ha transmitido otros flits además del que contiene un error, la petición es que se retransmitan todos los flits a partir del erróneo.

Capa de Enrutamiento QPI

La capa de enrutamiento se utiliza para determinar el curso que un paquete tomará a través el sistema de interconexiones disponibles. Tablas de enrutamiento se definen mediante firmware y describen los posibles caminos que un paquete puede seguir. En configuraciones pequeñas, como una plataforma de dos puertos, las opciones de enrutamiento son limitadas y las tablas bastante simples. Para sistemas más grandes, las opciones son más complejas, dando flexibilidad para enrutar y reenrutar tráfico dependiendo de cómo se encuentran ubicados los dispositivos, cómo se particionan los recursos el sistema y se mapea alrededor de un recurso que ha fallado.

Capa de Protocolo QPI

En esta capa, se utiliza como unidad de transferencia el paquete. La definición de los contenidos de un

paquete se encuentra estandarizada con algo de flexibilidad para permitir cumplir con los requerimientos de distintos segmentos de mercado. Una función clave en este nivel es el protocolo de coherencia de caché, el cual se encarga de que los valores de la memoria principal almacenados en cachés múltiples sean consistentes. El contenido típico de un paquete es un bloque de datos que es enviado hacia o desde el caché.

3.7 PCI Express

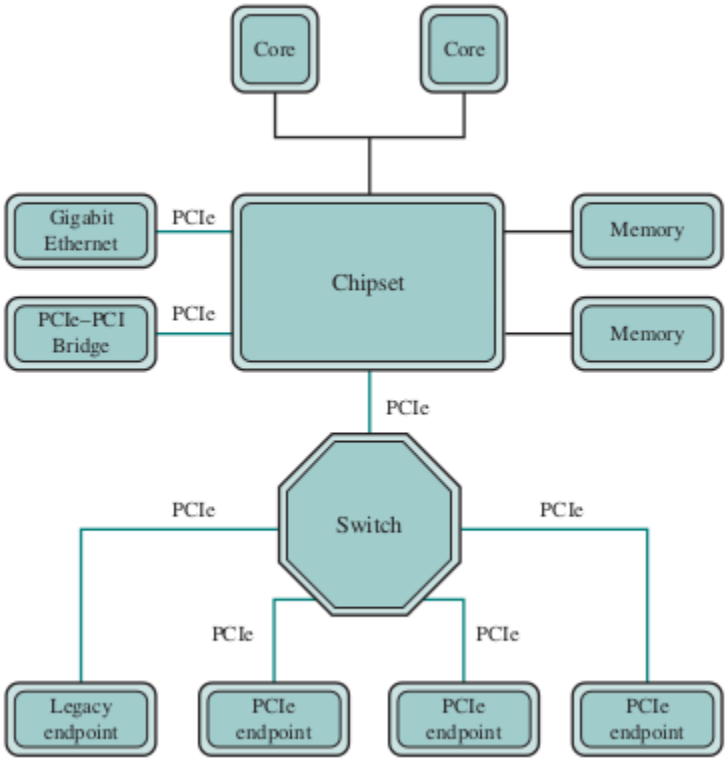
El esquema del bus PCI no ha podido mantener el paso con las demandas de transferencia de datos de los dispositivos que se conectan a él. Así pues, una nueva versión, el PCI Express (PCIe) ha sido desarrollado, como con QPI, para reemplazar los esquemas que utilizan PCI.

Un requerimiento clave de para PCIe es la alta capacidad de soportar las necesidades de los dispositivos I/O con altas velocidades de transferenica, como el Ethernet a Gigabit. Otro requerimiento consiste en soportar flujos de datos dependientes del tiempo. Las aplicaciones tales como el video bajo demanda y la redistribución de audio han creado restricciones de tiempo real en los servidores también. Muchas aplicaciones de comunicación y sistemas de control embebidos también procesan datos en tiempo real.

Las plataformas actuales también deben lidiar con transferencias múltiples concurrentes y ya no es aceptable tratar todos los datos como iguales, así que los datos deben de ser etiquetados para que el sistema de I/O pueda prioritizar el flujo a través de la plataforma.

Arquitectura Lógica y Física de PCIe

La figura siguiente muestra una configuración típica que soporta el uso de PCIe. Un dispositivo raíz, también conocido como chipset o host bridge, conecta el procesador y el subsistema de memoria a la tela de switches PCI Express que se compone de uno o más switches PCIe.



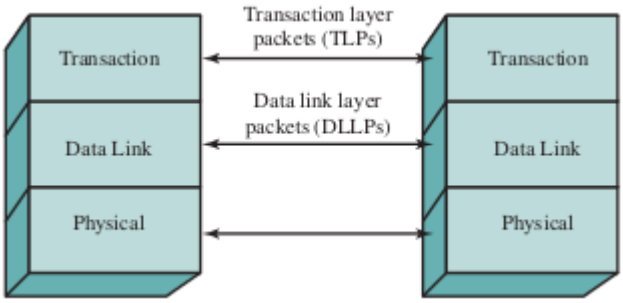
El chipset actúa como un dispositivo buffer, para lidiar con la diferencia entre tasas de transferencia de distintos controladores I/O, los componentes de memoria y el procesador. El chipset también traduce entre los formatos de transacción PCIe y los requerimientos de señales y control de la memoria y el procesador. El chipset típicamente soporta múltiples puertos PCIe, algunos de los cuales se conectan directamente a un dispositivo PCIe y uno o más se conectan a un switch que maneja múltiples flujos PCIe.

Los enlaces PCIe del chipset pueden conectarse a los siguientes tipos de dispositivos:

- **Switch:** un switch que maneja múltiples flujos PCIe.
- **Dispositivo final PCIe:** un dispositivo I/O o un controlador que implementa PCIe como puede ser un switch de Ethernet a Gigabit, un controlador de video o gráficos, una interfaz de disco, etc.
- **Dispositivo final de legacía:** esta categoría se refiere para diseños existentes previamente que pueden haber sido migrados a PCIe.
- **PCIe/PCI brigde:** permite que dispositivos PCI se conecten a sistemas basados en PCIe.

Como con QPI, las interacciones PCIe se definen utilizando una arquitectura de protocolos. Esta arquitectura incluye las siguientes capas:

- **Física:** consiste de los cables que transportan las señales, así como la circuitería y lógica necesarias para soportar las características requeridas en la transmisión y recepción de los 1's y 0's.
- **Enlace de datos:** es responsable de la transmisión confiable y el control de flujo. Los paquetes de datos generados y consumidos por la capa de enlace de datos (DLL) se llaman paquetes de la capa de enlace de datos (DLLP's, Data Link Layer Packets).
- **Transacción:** genera y consume paquetes de datos utilizados para implementar los mecanismos de transferencia para cargar/guardar y también maneja el control de flujo de dichos paquetes entre dos componentes en un enlace. Los paquetes de datos generados y consumidos en la capa de transacción se conocen como paquetes de la capa de transacción (TLP's, Transaction Layer Packets).

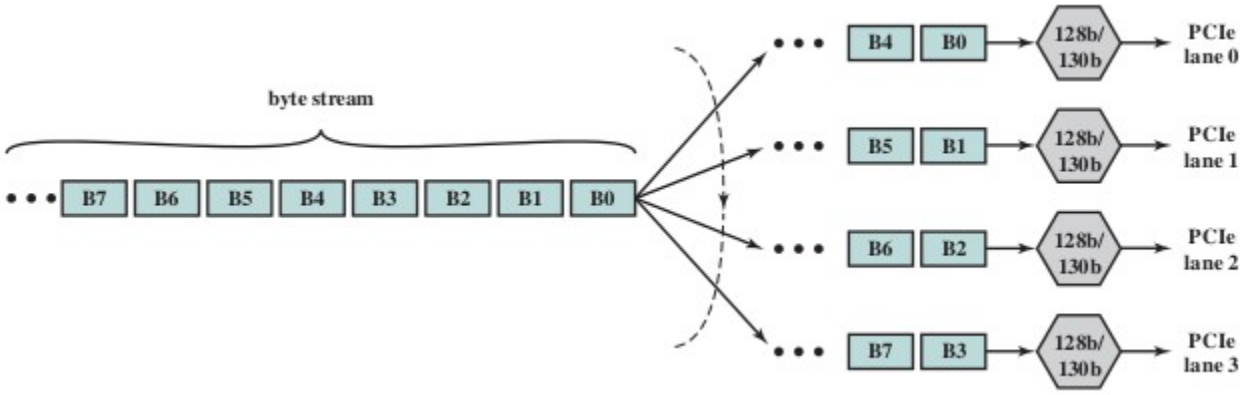


Por encima de la capa de transacción se encuentran las capas de software que generan las peticiones de lectura y escritura que son transportadas por la TL hacia los dispositivos I/O utilizando un protocolo de transacción basado en paquetes.

Capa Física PCIe

Similar a QPI, PCIe es una arquitectura punto a punto. Cada puerto PCIe consiste de un número de canales bidireccionales (note que en QPI, un canal se refiere a transferencia unidireccional). Un puerto PCI puede proveer 1, 5, 6, 16 o 32 canales.

Como QPI, PCIe utiliza una técnica de distribución multicanal. La siguiente figura muestra un ejemplo



de un puerto PCI que consiste de 4 canales. Los datos se distribuyen a los 4 canales 1 bit a la vez. En cada canal físico, los bits son almacenados en un buffer y procesados 16 bytes a la vez. Cada bloque de 128 bits se codifica como código único de 130 bits para transmitirlo, a esto se le llama codificación 128b/130b.

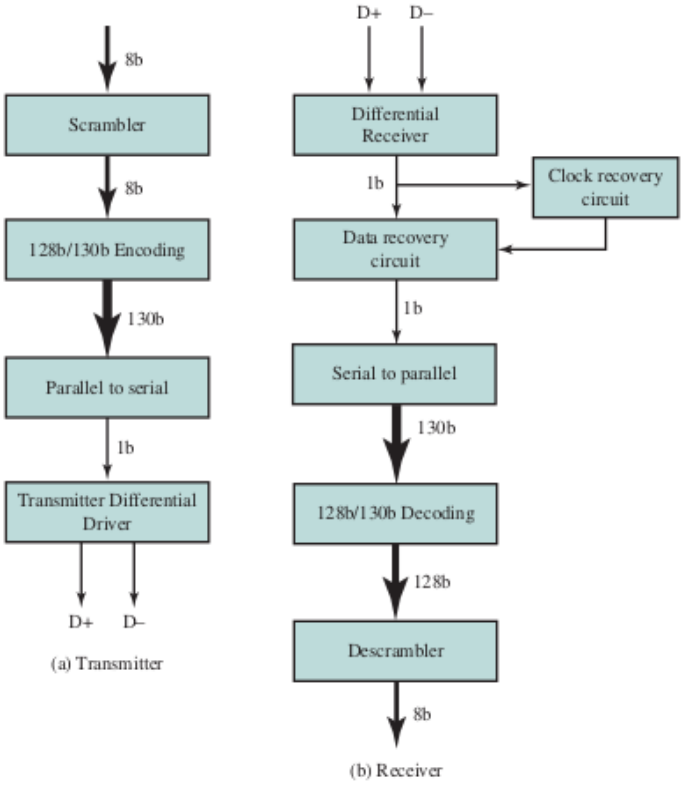
Para entender la razón de esta codificación note que, a diferencia de QPI, PCIe no utiliza la linea de reloj para sincronizar el flujo de bits. Es decir, la línea de reloj no se utiliza para determinar el inicio y fin de cada bit; se utliza para otros propósitos de señalización únicamente. Sin embargo, es necesario que el receptor se encuentre sincronizado con el transmisor, de manera que el receptor sepa dónde comienza y termina cada bit.

PCIe se basa en que el receptor se sincronize con el transmisor basándose en la señal transmitida. Como con QPI, PCIe utiliza la señalización diferencial para transmitir los bits. La sincronización se puede lograr cuando el receptor encuentra una transición en los datos y sincronizándose a la transición. Sin embargo, considere que con una cadena larga de 1's o 0's, al utilizar señalización diferencial, la salida será un voltaje constante a lo largo de un periodo de tiempo. Bajo estas circunstancias, es difícil determinar el punto de inicio y terminación de los bits puesto que no hay transiciones.

Un enfoque para sobreponerse al problema mencionado es el *scrambling*. El scrambling no aumenta el número de bits a transmitir, puesto que es una técnica de mapeo que hace parecer a los datos como más aletarorios. El scrambling tiende a separar las transiciones de manera que aparezcan en el receptor espaciadas de manera más uniforme, lo cual es bueno para la sincronización.

Otra técnica que ayuda en la sincronización es la codificación, en la cual, bits adicionales se insertan al flujo de bits para forzar transiciones. Para PCIe 3.0, cada grupo de 128 bits se mapea a un bloque de 130 bits añadiendo un encabezado de 2 bits. El valor del encabezado es 10 para un bloque de datos y 01 para paquetes de información de enlace de datos.

La figura a la derecha ilustra el uso de scrambling y codificado para la transmición. Usando estas técnicas se puede lograr una tasa de transferencia de 16 GB/s.



Capa de Transacción PCIe

La capa de transacción o TL recibe peticiones de lectura y escritura del software que se encuentra por encima de ella y crea paquetes para su transmición a un destino a traŭes de la capa de enlace.

La mayoría de las transacciones utilizan una técnica llamada **transacción divida** que funciona de la siguiente manera: un paquete de petición se envía desde el dispositivo PCIe origen, el cuál después aguarda por una respuesta, llamada paquete de terminación. La terminación que sigue a una petición es iniciada por el dispositivo PCIe destino sólo cuando tiene los datos o el estado listos para entrega. Cada paquete tiene un identificador único que permite que los paquetes de terminación sean dirigidos al origen correcto. Con la técnica de transacción dividida, la terminación está separada en el tiempo de la petición, en contraste con la operación de un bus típico en el cual ambos lados de la transacción deben estar disponibles al mismo tiempo. Entre la petición y la terminación, otros dispositivos PCIe pueden utilizar el enlace.

Otras transacciones, llamadas **transacciones publicadas** no requieren de respuesta por parte del destino como pueden ser transacciones de escritura o mensajes de control.

Address Space	TLP Type	Purpose
Memory	Memory Read Request	Transfer data to or from a location in the system memory map.
	Memory Read Lock Request	
	Memory Write Request	
I/O	I/O Read Request	Transfer data to or from a location in the system memory map for legacy devices.
	I/O Write Request	
Configuration	Config Type 0 Read Request	Transfer data to or from a location in the configuration space of a PCIe device.
	Config Type 0 Write Request	
	Config Type 1 Read Request	
	Config Type 1 Write Request	
Message	Message Request	Provides in-band messaging and event reporting.
	Message Request with Data	
Memory, I/O, Configuration	Completion	Returned for certain requests.
	Completion with Data	
	Completion Locked	
	Completion Locked with Data	

El paquete TL soporta formatos de direccionamiento de 32 o 64 bits e incluyen atributos como "no-snoop", "relaxedordering" y "priority" que pueden ser utilizados para enrutar óptimamente dichos paquetes a través del subsistema de I/O.

Espacios de Direcciones y Tipos de Transacciones

La capa de transacción soporta 4 espacios de direcciones:

- **Memoria:** que incluye la memoria principal del sistema y también los dispositivos I/O PCIe. Ciertos rangos de memoria mapean a dispositivos I/O.
- **I/O:** utilizado para dispositivos PCI de legacía, con rangos de direcciones de memoria reservados para dispositivos I/O antiguos.
- **Configuración:** este espacio de direcciones permite que la TL lea/escriba en los registros de configuración asociados con dispositivos I/O.
- **Mensaje:** relacionado con las señales de control para el manejo de interrupciones, manejo de errores y manejo de energía.

La tabla a la izquierda muestra las transacciones soportadas por la TL. Para los espacios de direcciones de memoria, I/O y configuración existen transacciones de lectura y escritura. En el caso de las transacciones de memoria, existen peticiones de lectura bloqueantes en las cuales se solicita hacer una serie de operaciones de manera atómica como puede ser una operación leer-modificar-escribir. Durante una transacción bloqueante el chipset bloquea el enlace PCIe para que no pueda ser utilizado por ningún otro dispositivo.

Para mantener la compatibilidad con los dispositivos PCI, PCIe soporta ciclos de configuración Tipo 0 (PCIe) y Tipo 1 (PCI).

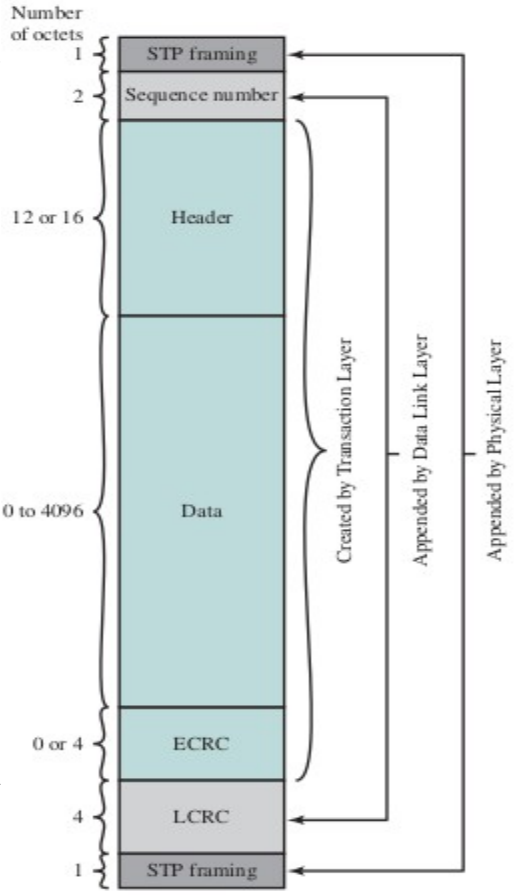
Finalmente, mensajes de terminación son utilizados para las transacciones de memoria, I/O y configuración que así lo requieran.

Ensamblaje del Paquete TLP

Como se mencionó antes, las transacciones PCIe se logran utilizando paquetes de la capa de transacción o TLP's. Un TLP se origina en la TL del dispositivo transmisor y termina en la TL del dispositivo receptor. La figura a la derecha muestra la estructura del TLP.

El software manda a la TL la información necesaria para que se cree el núcleo del TLP el cuál consiste de los siguientes campos:

- **Encabezado:** describe el tipo de paquete e incluye la información necesaria para que el receptor procese el paquete, incluyendo cualquier información de enrutamiento necesaria.

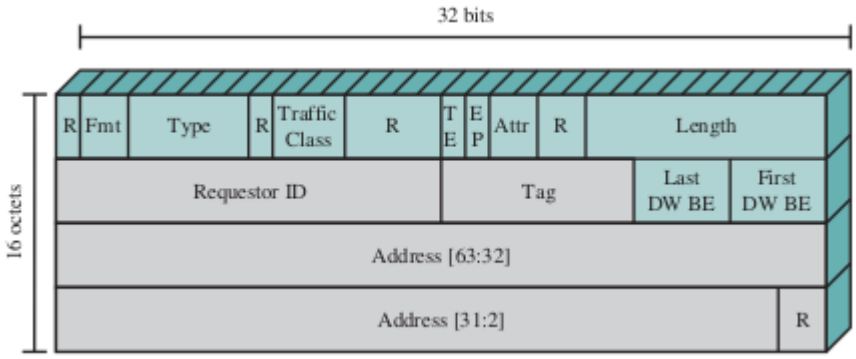


(a) Transaction Layer Packet

- **Datos:** un campo de datos de hasta 4096 bytes puede ser incluído en el paquete aunque no todos los TLP's lo continen
- **ECRC:** campo opcional de chequeo de errores, permite a la TL revisar si existen errores en el encabezado o los datos.

En seguida, en la figura de abajo, se muestra un ejemplo del formato de encabezado, para una petición de lectura de memoria. Los campos en verde se encuentran presentes en todos los encabezados. Además de los campos reservados para uso futuro (R), estos campos incluyen los siguientes:

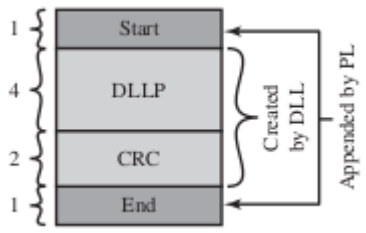
- Longitud: la longitud del campo de datos en Palabras Dobles o Double Words, 1 DW = 4 bytes.
- Atributos: consiste de 2 bits. El bit de ordenamiento relajado (*relaxed ordering*) que indica que una transacción puede ser completada antes que otras transacciones que se iniciaron previamente; y el *no-snoop* bit que indica que no existen problemas de coherencia de caché respecto al TLP.
- EP: bit de datos envenenados, que indica que los datos del TLP deben ser considerados como inválidos aunque se permita completar la transacción normalmente.
- TE: que indica si está presente el campo ECRC.
- Clase de Tráfico: asigna un tipo o prioridad al flujo de tráfico que permite prioritizar el servicio.
- Tipo y Formato: estos dos campos, 7 bits en total, especifican el tipo de transacción, tamaño del encabezado y si está presente el campo de datos.
- Habilitación de bytes de la primera DW: estos 4 bits indican, respectivamente si el byte correspondiente en la primera palabra doble es válido.
- Habilitación de bytes de la última palabra doble: estos 4 bits indican, respectivamente, si el byte correspondiente en la última palabra doble es válido. Éste y el campo precedente permiten realizar tarnsferencias más pequeñas que una DW compleata así como defasar las direcciones de inicio y terminación.



En la figura se muestra el encabezado de una transacción de petición de memoria. El ID del solicitante identifica al dispositivo origen, e indica al terminador a dónde enviar su respuesta. La etiqueta (*tag)* es un número asignado a esta transacción por el solicitante, terminador debe incluír esta etiqueta en su respuesta de manera que el solicitante pueda hacer coincidir la petición con la respuesta. El campo de dirección indica la dirección de memoria inicial desde la cual leer.

Capa de Enlace de Datos PCIe

El propósito de la capa de enlace de datos es asegurar la entrega confiable de los paquetes a través del enlace PCIe. La DLL (*Data Link Layer*) participa en la formación de los TLP's y transmite DLLP's.



Paquetes de Capa de Enlace de Datos

La figura siguiente muestra el formato de los paquetes de la capa de enlace de datos. Existen tres tipos importantes de DLLP's utilizados en el manejo de los enlaces: paquetes de control de flujo, paquetes de manejo de energía y paquetes de reconocimiento ACK y NAK. Los paquetes de control de flujo regulan el ritmo al cual se transmiten los TLP's y DLLP's por un enlace. Los paquetes ACK y NAK se utilizan en el procesamiento de TLP's.

Procesamiento de TLP's

La DLL añade dos campos al núcleo del TLP creado por la TL: un número e secuencia de 16 bits y un CRC de 32 de capa de enlace o LCRC. Mientras que los campos del núcleo creados por la TL únicamente se utilizan en la TL destino, los dos campos agregados por la DLL son procesados en cada nodo intermedio que se cruza en camino al destino.

Cuando un TLP llega a un dispositivo, la DLL revisa el número de secuencia y el campo LCRC. Al revisarlo existen dos posibilidades:

1. Si no se detectan errores, el núclo del TLP se pasa a la TL local. Si éste dispositivo es el destino del TLP, entonces la TL procesa el TLP. Si no es el destino, la TL determina la ruta para el paquete y lo pasa de regreso a la DLL para su transmición a través del siguiente enlace.
2. Si se detecta un error, la DLL crea un paquete NAK y lo envía de regreso a transmisor origen del TLP. Se elimina el TLP recibido.

Cuando la DLL transmite un TLP, almacena una copia del paquete. Si se recibe un NAK para el paquete con ese número de secuencia, se retransmite la copia del TLP. Cuando se recibe un ACK, se descarta el TLP almacenado en el búffer.