Arquitecturas de Almacenamiento

Bloque Temático 1

Conceptos básicos de la E/S y buses

Contenidos

- Tema I: Organización de la E/S
- Tema II : Análisis y evolución del bus PCI

Arquitecturas de Almacenamiento

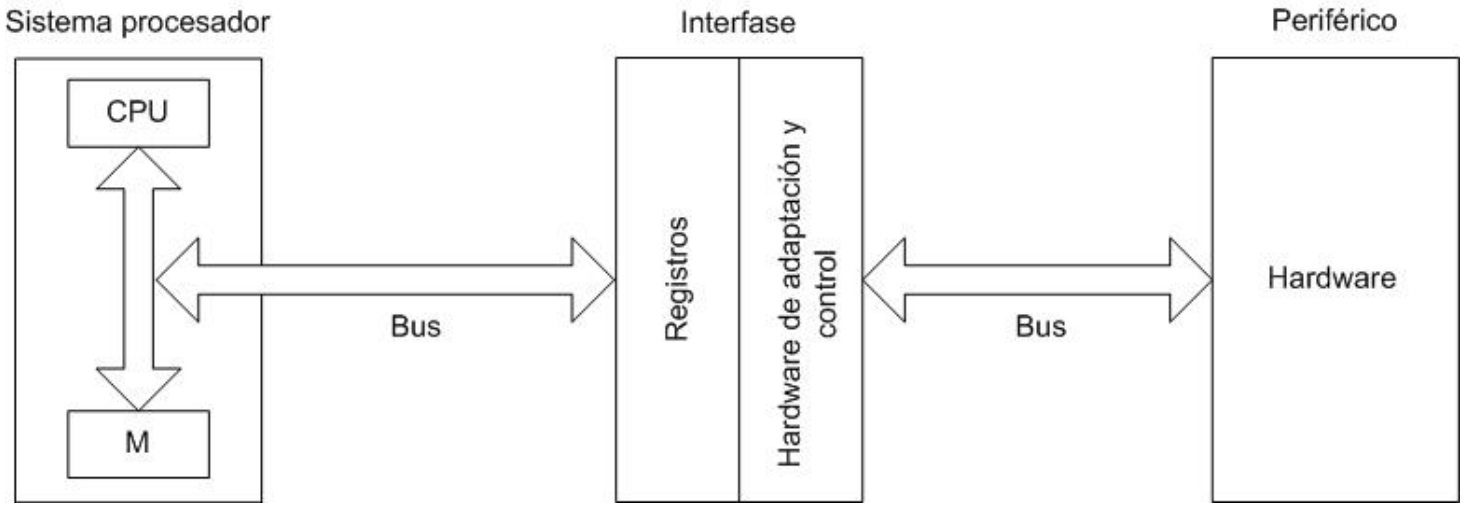
Tema I

Organización de la E/S

Contenidos

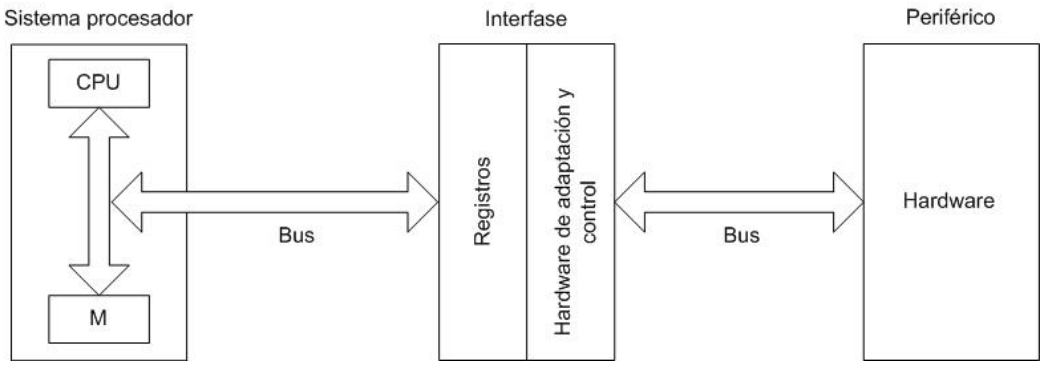
- Estructura de la E/S
- Interfase y modelo de programación del periférico
- E/S por PIO o DMA
- Papel de las interrupciones en la E/S
- Mecanismos de gestión de interrupciones y DMA
- Diferenciación entre buses e interfases
- Transacciones de bus
  - Transacciones divididas, e impacto en la eficiencia
- Evaluación del rendimiento de la E/S

I – Estructura de la E/S



- Organización de la E/S se ajusta siempre a este esquema
- Interfase suele estar integrada en periférico, pero puede ser un dispositivo independiente
- "Sistema procesador" puede ser CPU/Memoria de ordenador, o CPU/Memoria de un dispositivo intermedio (ej: HBA SCSI)

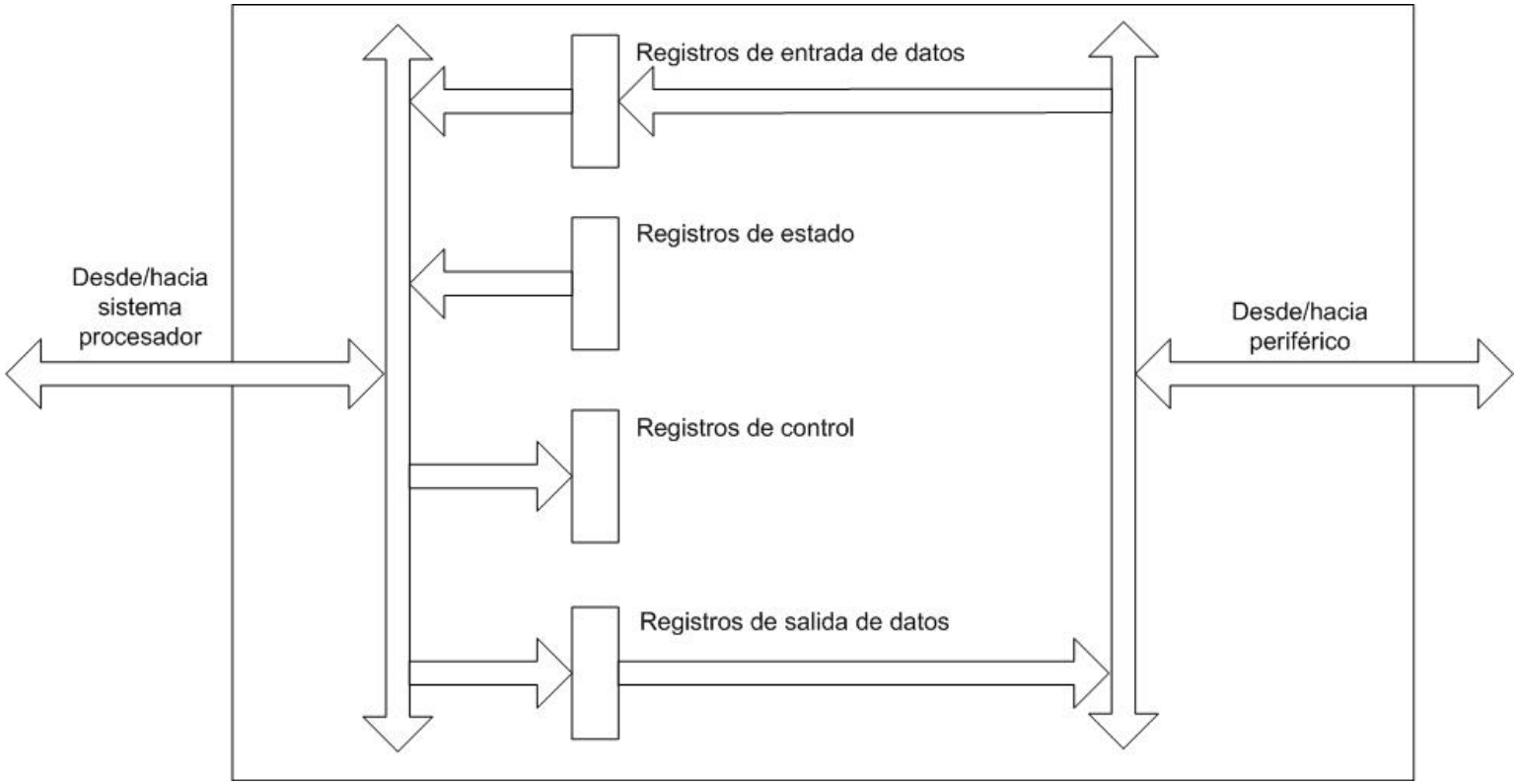
Estructura de la E/S



- "Driver" del dispositivo: software que gestiona y controla la transferencia de datos
  - Corre en el sistema procesador

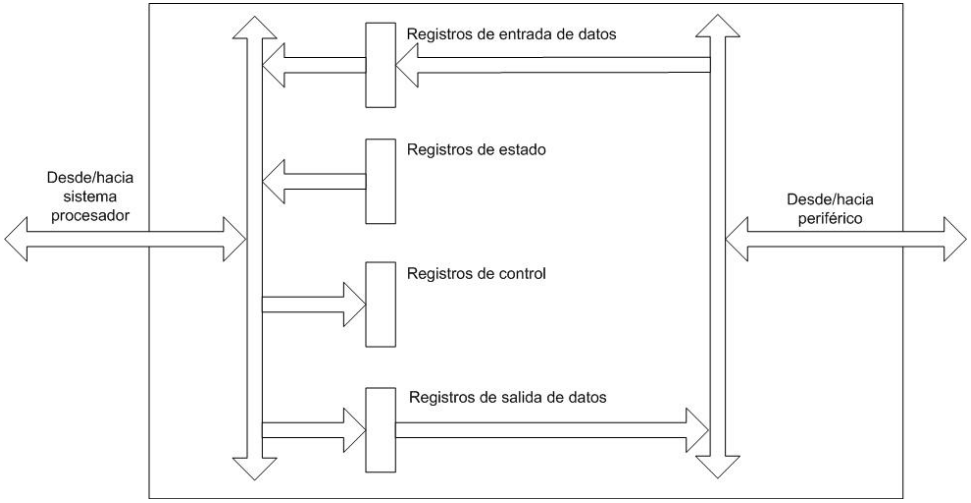
- Dirección de las transferencias referida a sistema procesador
  - Lectura (R = Read) : Entrada
    - Datos provienen del periférico y entran al sistema procesador
  - Escritura (W = Write) : Salida
    - Datos provienen del sistema procesador y van al periférico

Estructura de la E/S



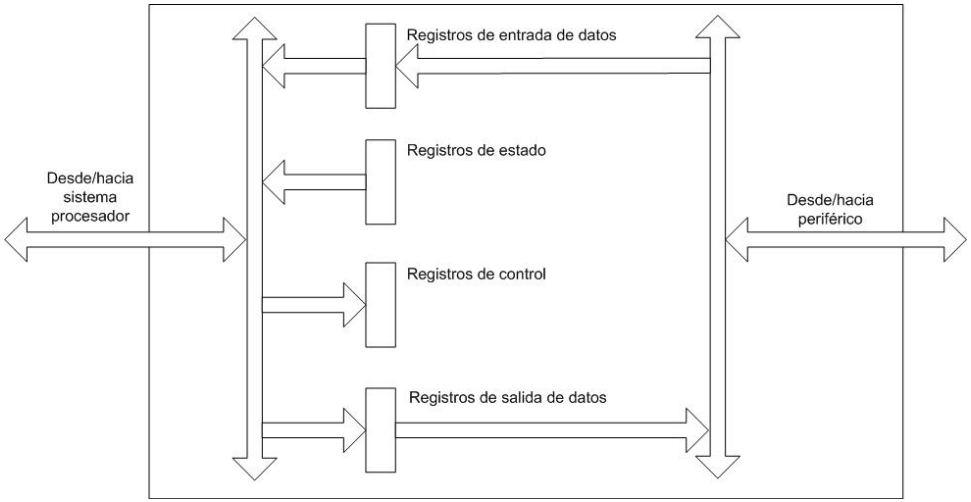
Organización típica de registros de la interfase

Estructura de la E/S



- Registros control: Escritura de valores sobre ellos permite el control indirecto del hardware
- Registros estado: Leyendo sus valores obtenemos información actual del estado del periférico

Estructura de la E/S



- Registros entrada datos: Hardware que alberga temporalmente los datos entregados desde el periférico
  - Puede ser un único registro, o un buffer RAM con múltiples posiciones
- Registros salida datos: Hardware que alberga temporalmente los datos enviados hacia el periférico
  - Puede ser un único registro, o un buffer RAM con múltiples posiciones

Modelo de programación del periférico

- Es la descripción completa de:
  - Los registros que componen la interfase
  - Su organización y estructura
  - El conjunto de comandos y parámetros que acepta la interfase
  - Su temporización y las condiciones de error

10.4 8259 Interrupt Controller (PIC) Registers (LPC I/F—D31:F0)

10.4.1 Interrupt Controller I/O MAP (LPC I/F—D31:F0)

The interrupt controller registers are located at 20h and 21h for the master controller (IRQ 0–7), and at A0h and A1h for the slave controller (IRQ 8–13). These registers have multiple functions, depending upon the data written to them. Table 10-3 shows the different register possibilities for each address.

Table 10-3. PIC Registers (LPC I/F—D31:F0)

Port	Aliases	Register Name	Default Value	Type
20h	24h, 28h, 2Ch, 30h, 34h, 38h, 3Ch	Master PIC ICW1 Init. Cmd Word 1	Undefined	WO
		Master PIC OCW2 Op Ctrl Word 2	001XXXXXb	WO
		Master PIC OCW3 Op Ctrl Word 3	X01XXX10b	WO
21h	25h, 29h, 2Dh, 31h, 35h, 39h, 3Dh	Master PIC ICW2 Init. Cmd Word 2	Undefined	WO
		Master PIC ICW3 Init. Cmd Word 3	Undefined	WO
		Master PIC ICW4 Init. Cmd Word 4	01h	WO
		Master PIC OCW1 Op Ctrl Word 1	00h	R/W
A0h	A4h, A8h, ACh, B0h, B4h, B8h, BCh	Slave PIC ICW1 Init. Cmd Word 1	Undefined	WO
		Slave PIC OCW2 Op Ctrl Word 2	001XXXXXb	WO
		Slave PIC OCW3 Op Ctrl Word 3	X01XXX10b	WO
A1h	A5h, A9h, ADh, B1h, B5h, B9h, BDh	Slave PIC ICW2 Init. Cmd Word 2	Undefined	WO
		Slave PIC ICW3 Init. Cmd Word 3	Undefined	WO
		Slave PIC ICW4 Init. Cmd Word 4	01h	WO
		Slave PIC OCW1 Op Ctrl Word 1	00h	R/W
4D0h	–	Master PIC Edge/Level Triggered	00h	R/W
4D1h	–	Slave PIC Edge/Level Triggered	00h	R/W

10.4.2 ICW1—Initialization Command Word 1 Register (LPC I/F—D31:F0)

Offset Address:	Master Controller – 20h Slave Controller – A0h	Attribute:	WO
		Size:	8 bit /controller
Default Value:	All bits undefined		

A write to Initialization Command Word 1 starts the interrupt controller initialization sequence, during which the following occurs:

1. The Interrupt Mask register is cleared.
2. IRQ7 input is assigned priority 7.
3. The slave mode address is set to 7.
4. Special mask mode is cleared and Status Read is set to IRR.

Once this write occurs, the controller expects writes to ICW2, ICW3, and ICW4 to complete the initialization sequence.

Bit	Description
7:5	**ICW/OCW Select —** WO. These bits are MCS-85 specific, and not needed. 000 = Should be programmed to "000"
4	**ICW/OCW Select —** WO. 1 = This bit must be a 1 to select ICW1 and enable the ICW2, ICW3, and ICW4 sequence.
3	**Edge/Level Bank Select (LTIM) —** WO. Disabled. Replaced by the edge/level triggered control registers (ELCR, D31:F0:4D0h, D31:F0:4D1h).
2	ADI — WO. 0 = Ignored for the Intel® ICH7. Should be programmed to 0.
1	**Single or Cascade (SNGL) —** WO. 0 = Must be programmed to a 0 to indicate two controllers operating in cascade mode.
0	**ICW4 Write Required (IC4) —** WO. 1 = This bit must be programmed to a 1 to indicate that ICW4 needs to be programmed.

Modelo de programación del periférico

- El modelo de programación permite desacoplar el hardware de su comportamiento lógico

- El fabricante puede así cambiar el hardware sin alterar su función
  - Sólo necesita conservar la compatibilidad con el modelo de programación previo

Direccionamiento de la interfase

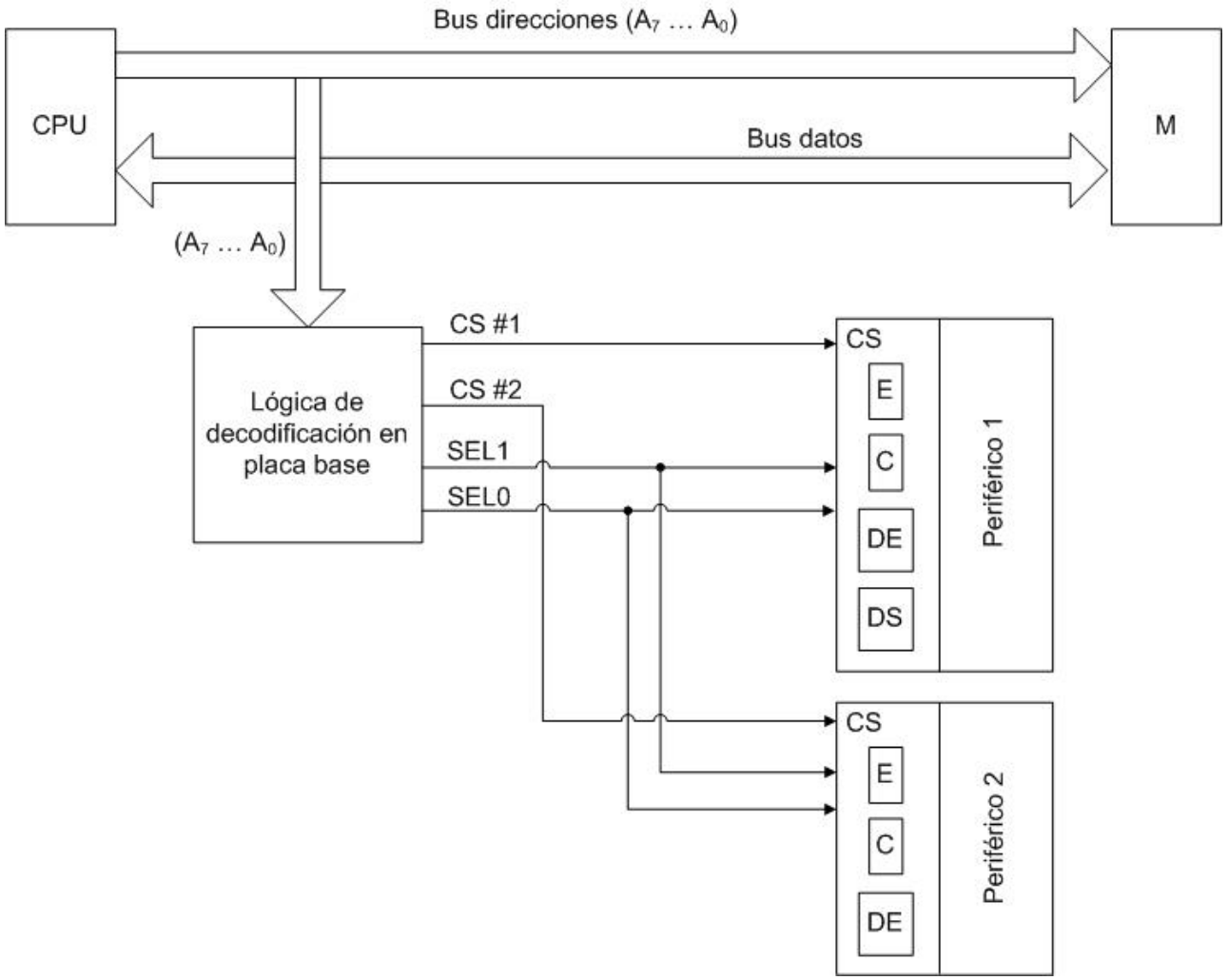
- En un mismo ordenador existirán múltiples periféricos
  - Cada periférico tendrá una o más interfases
  - Cada interfase tendrá uno o más registros
- Para realizar una transferencia desde el sistema computador hay que identificar con precisión:
  - qué periférico estará involucrado
  - el registro origen o destino de la transferencia
- Se llama *direccionamiento* al mecanismo que permite esta identificación

Direccionamiento de la interfase

- El direccionamiento requiere combinación de mecanismos hardware:
  - Asignación de un valor a las líneas de un bus de direcciones
  - Uso de una línea de activación selectiva (Chip Select, CS) de la interfase.
  - Decodificación del valor presente en el bus de direcciones
    - La decodificación dará como resultado la activación de una determinada interfase y un cierto registro en esa interfase

Direccionamiento de la interfase

Ejemplo:



Direccionamiento de la interfase

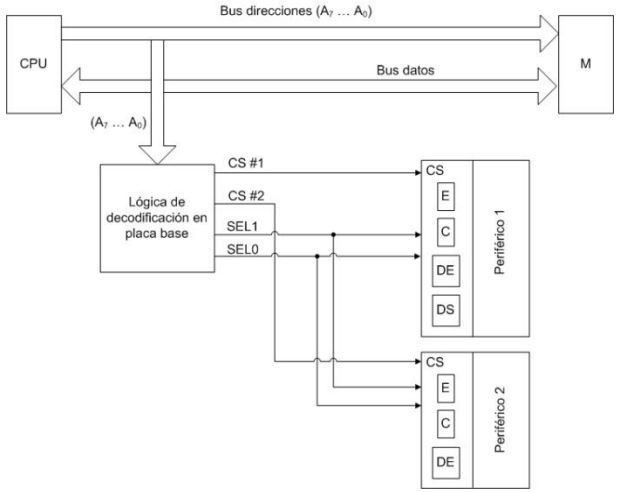
Direccionamiento local en periférico 1

A_1	A_0	Registro
0	0	Estado
0	1	Control
1	0	Datos Entrada
1	1	Datos Salida

Direccionamiento local en periférico 2

A_1	A_0	Registro
0	0	Estado
0	1	Control
1	0	Datos Salida
—	—	Reservado

- Debemos asignar, arbitrariamente, al menos 7 de las 256 direcciones posibles para el acceso a la E/S
  - Elegimos arbitrariamente ubicar periférico1 en 0x80
  - Análogamente, elegimos ubicar periférico2 en 0x90



A_7 … A_0	Dispositivo direccionado	CS1	CS2	SEL1	SEL2
(128) 0x00 a 0x7F	M	0	0	x	x
0x80	periférico 1, registro E	1	0	0	0
0x81	periférico 1, registro C	1	0	0	1
0x82	periférico 1, registro DE	1	0	1	0
0x83	periférico 1, registro DS	1	0	1	1
(12) 0x84 a 0x8F	M	0	0	x	x
0x90	periférico 2, registro E	0	1	0	0
0x91	periférico 2, registro C	0	1	0	1
0x92	periférico 2, registro DE	0	1	1	0
0x93	Reservada	No se debe usar			
(108) 0x94 a 0xFF	M	0	0	x	x

Direccionamiento de la interfase

- Las direcciones de E/S están entremezcladas con las de memoria
  - CPU usará la misma instrucción para acceder a una posición de memoria o a una posición de E/S

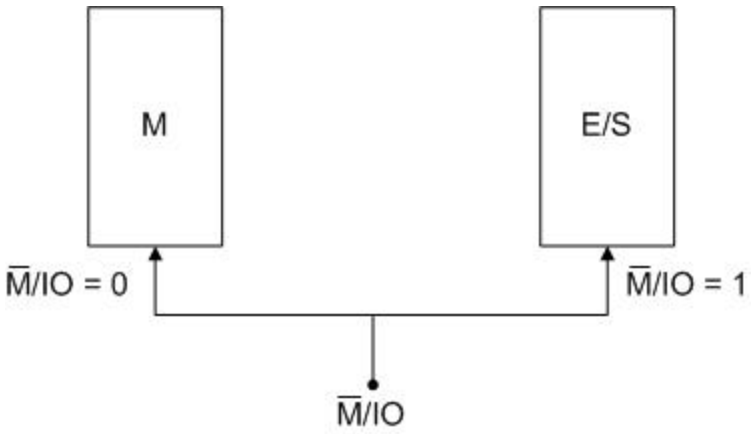
- Este tipo de direccionamiento se llama *E/S mapeada por memoria*

Direccionamiento de la interfase

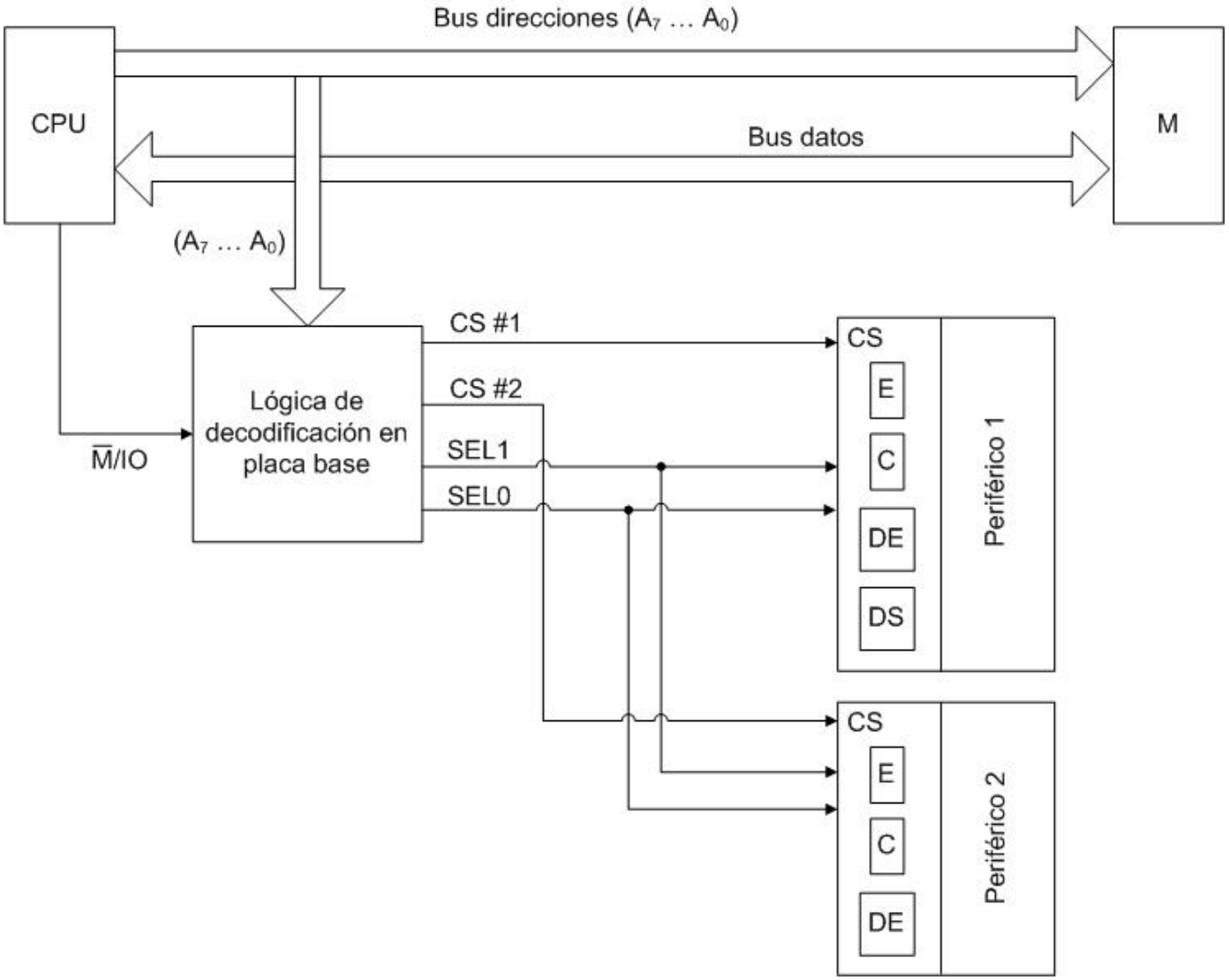
- Ventajas de E/S mapeada por memoria:
  - Registro E/S se ve simplemente como una posición "especial" de memoria
  - CPU no necesita usar instrucciones especiales para acceder a E/S
- Inconvenientes:
  - Fragmenta el espacio de direcciones
    - En el ejemplo, ya no es posible conseguir un bloque de más de 128 posiciones de memoria contiguas
  - Un error de programación puede hacernos direccionar accidentalmente posiciones de E/S
    - Debe tenerse esto en cuenta, y diseñar el sistema de forma que no se puedan provocar accidentalmente daños físicos

Direccionamiento de la interfase

- Alternativa de direccionamiento: **E/S diferenciada**
  - Usa un espacio de direcciones separado para la E/S
  - Utiliza una línea física específica para identificar accesos al espacio de E/S
  - Requiere el uso de una instrucción especial (IN o OUT)
    - Instrucción activa línea física de direccionamiento E/S
    - *Puertos* de E/S: direcciones accedidas usando IN o OUT



Direccionamiento de la interfase



Direccionamiento de la interfase

- Ventajas de E/S diferenciada:
  - Separa totalmente accesos de E/S de accesos a memoria
  - Evita fragmentación del espacio de direcciones de memoria
- Inconvenientes:
  - Requiere el uso de instrucciones especiales (IN o OUT)
    - Sólo existen en algunos procesadores (Intel)
    - Direccionamiento limitado a 16 bits (65536 direcciones)
  - Accesos son más lentos que E/S mapeada por memoria

Direccionamiento de la interfase

- Computadores actuales usan un modelo mixto de direccionamiento
  - Se usan puertos de E/S para acceder a registros críticos o dispositivos “legacy”
  - Se usa E/S mapeada por memoria para accesos veloces a E/S
  - También se usa E/S mapeada por memoria para direccionar funciones que requieren muchos registros

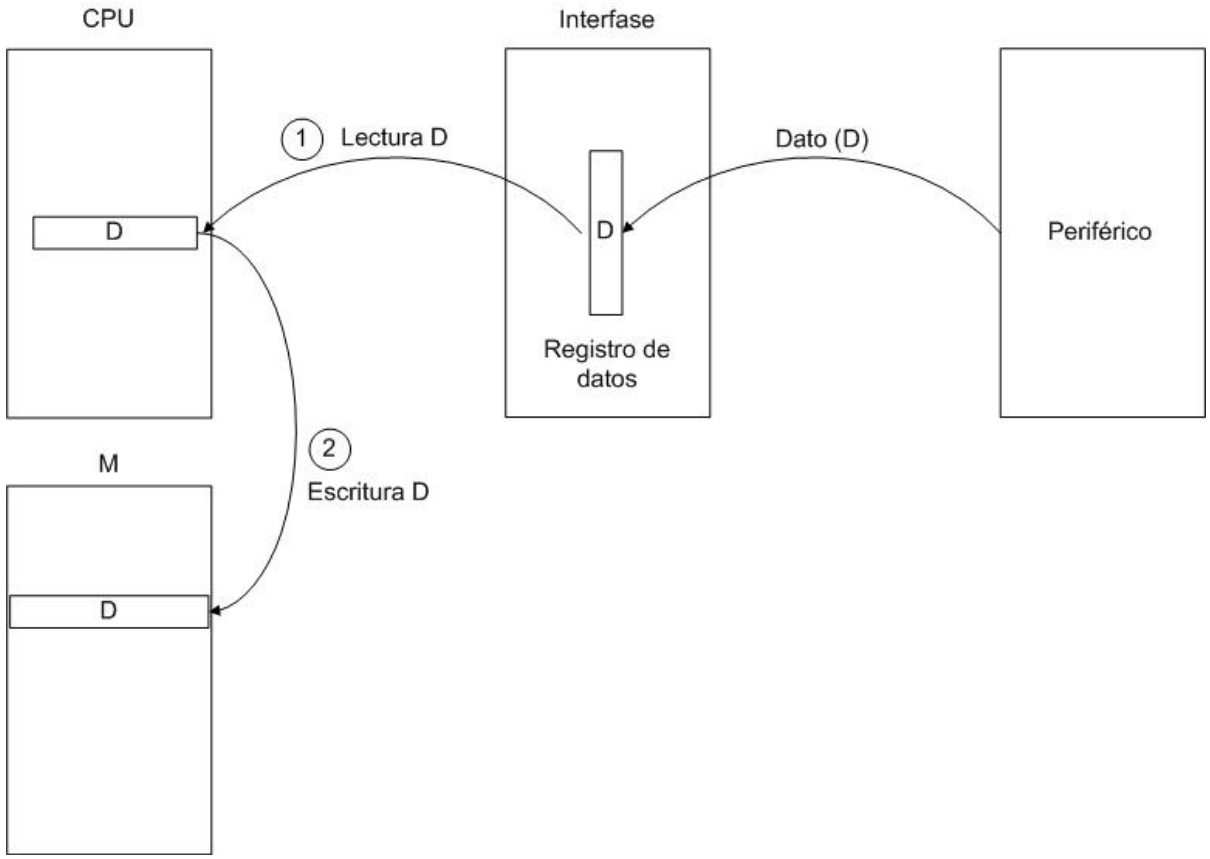
II – Métodos de E/S

- Existen dos formas de realizar las transferencias de E/S: PIO y DMA

- E/S por programa (PIO = Program I/O)
  - Involucra uso explícito de la CPU en cada movimiento de datos
    - Registro de CPU actúa como intermediario entre M y periférico

- E/S por DMA (Direct Memory Access)
  - No requiere emplear explícitamente la CPU en cada transferencia
  - Precisa empleo de hardware especial para controlar la transferencia y provocar el movimiento de los datos

E/S por programa (PIO)



- Diagrama muestra transacción de lectura
  - Transacción de escritura es análoga, invirtiendo el sentido y el orden de las operaciones
  - Necesario ejecutar dos instrucciones en la CPU por cada transferencia

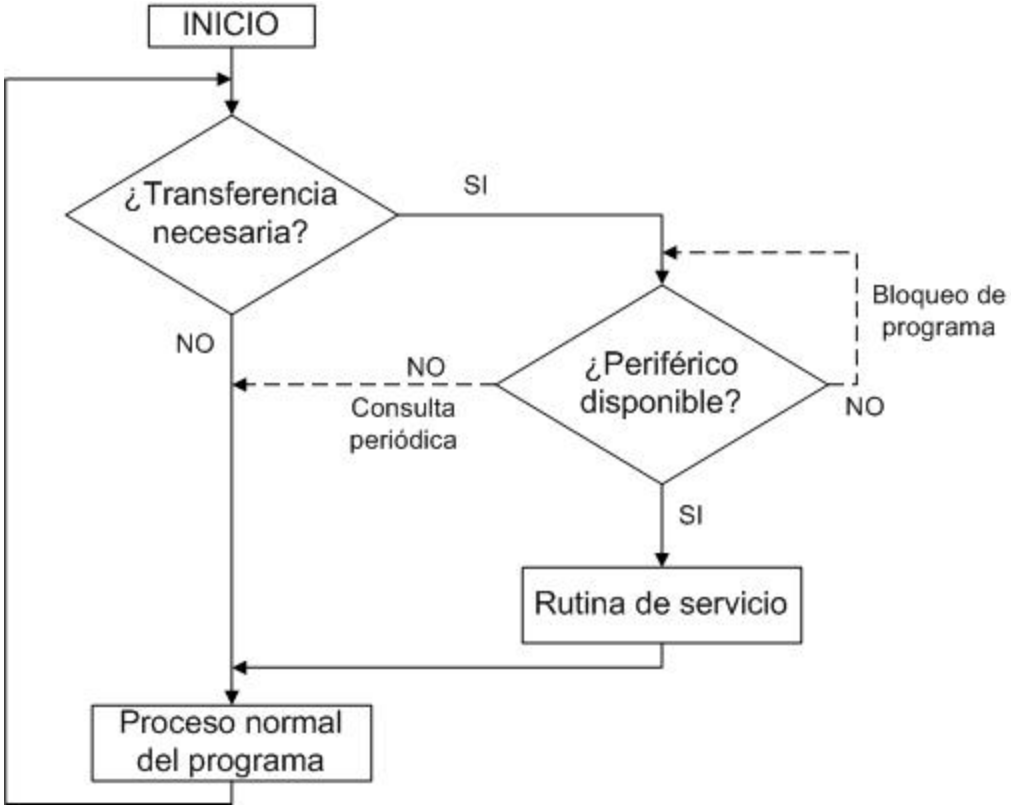
E/S por programa (PIO)

- Ventajas:
  - No requiere hardware especial
  - Sencilla de implementar
- Inconvenientes:
  - Requiere el uso continuo de CPU durante la transferencia
    - Movimiento de bloques de datos ralentiza notablemente al sistema
- Usada para transferencias puntuales, que involucran pocos datos

Consulta de estado

- Antes de realizar la transferencia PIO debe comprobarse si el periférico está preparado para llevarla a cabo
  - Requiere protocolo de *handshaking* (intercambio de información de control
- *Consulta de estado*: acceso a registros de la interfase (estado o control) para realizar este handshaking

Consulta de estado



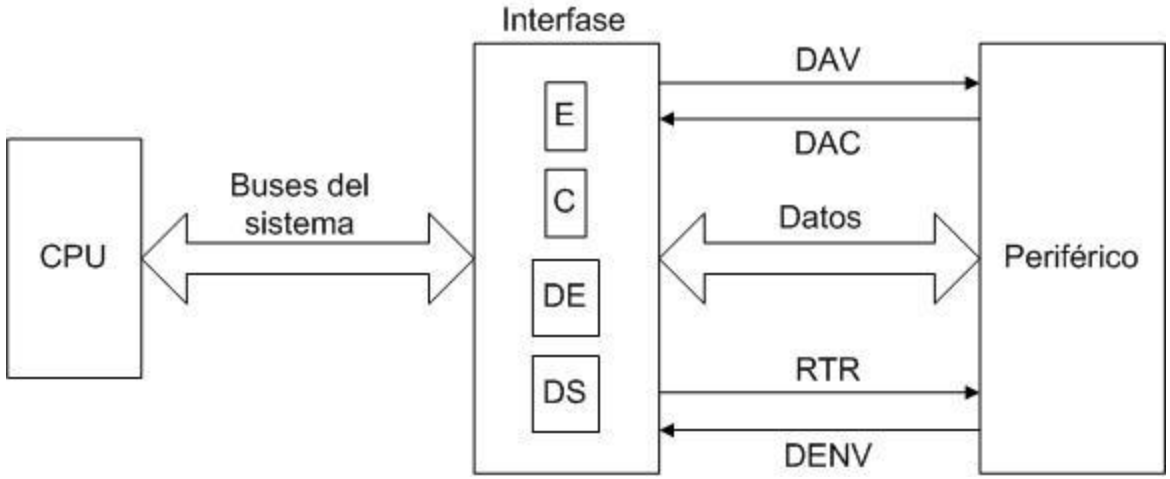
- Consulta de estado puede realizarse de dos formas distintas:
  - Bloqueo de programa
  - Consulta periódica

Consulta de estado

- Bloqueo programa: driver entra en bucle de espera, leyendo continuamente registro hasta que periférico esté preparado
  - Garantiza tiempo de respuesta mínimo...
  - pero paraliza la operación del sistema

- Consulta periódica: Si periférico no está listo, driver permite al programa hacer otras cosas antes de volver a intentar el acceso
  - No garantiza un tiempo de respuesta mínimo
  - Puede ser difícil acotar el tiempo máximo de respuesta
  - A cambio, permite al sistema operar concurrentemente con la E/S

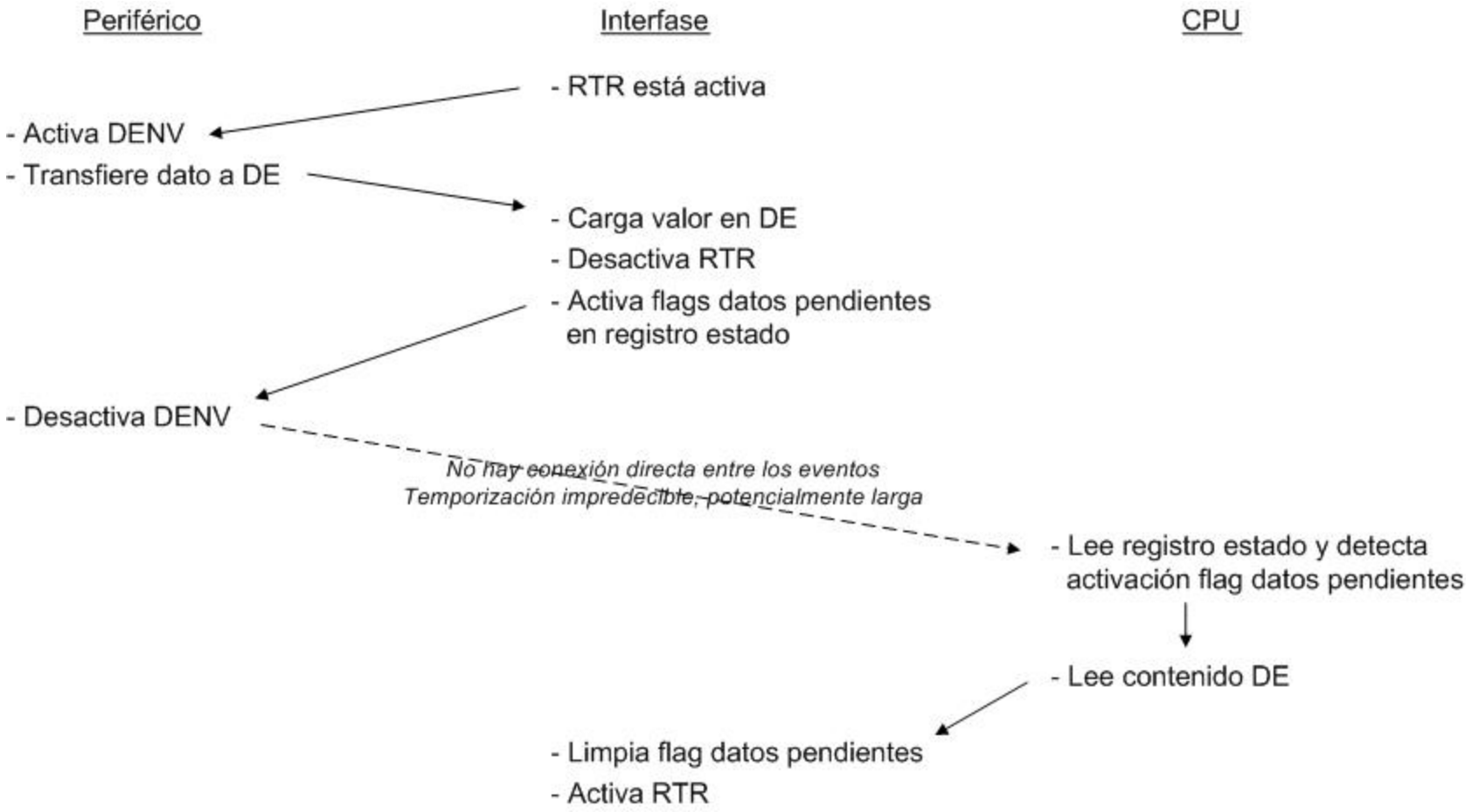
Ejemplo de E/S por PIO



- Señalización entre interfase y periférico usa 4 líneas:
  - DAV (Dato Válido): Interfase indica a periférico envío de dato. Se mantiene activa hasta recibir DAC
  - DAC (Dato Aceptado): Periférico indica que ha aceptado el dato
  - RTR (Ready To Receive): Interfase activa esta señal para indicar al periférico disponibilidad para recibir un dato
  - DENV (Dato Enviado): Periférico indica a la interfase que le está enviando un dato

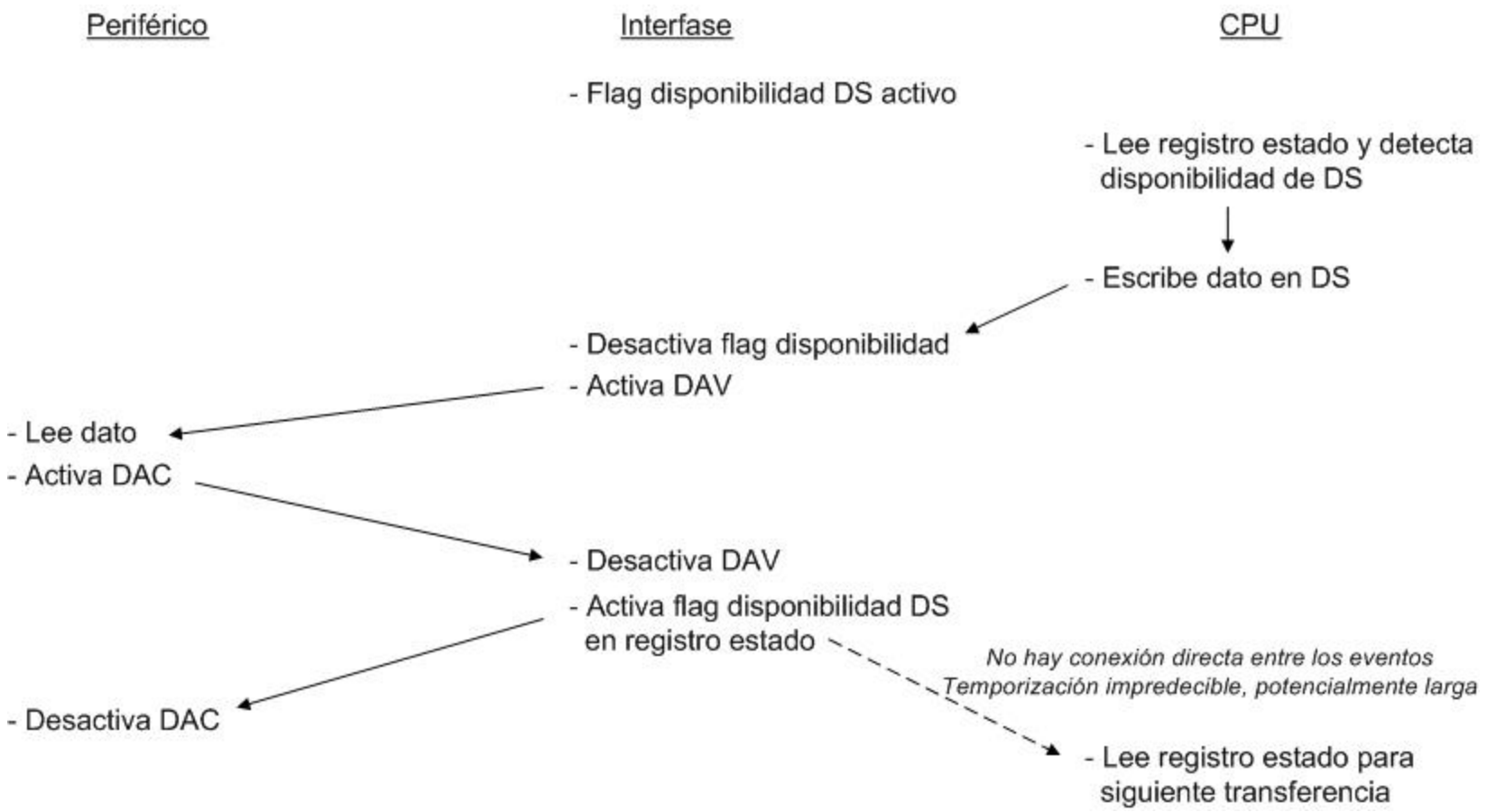
Ejemplo de E/S por PIO

Descripción de una lectura de datos, previamente solicitada desde la CPU



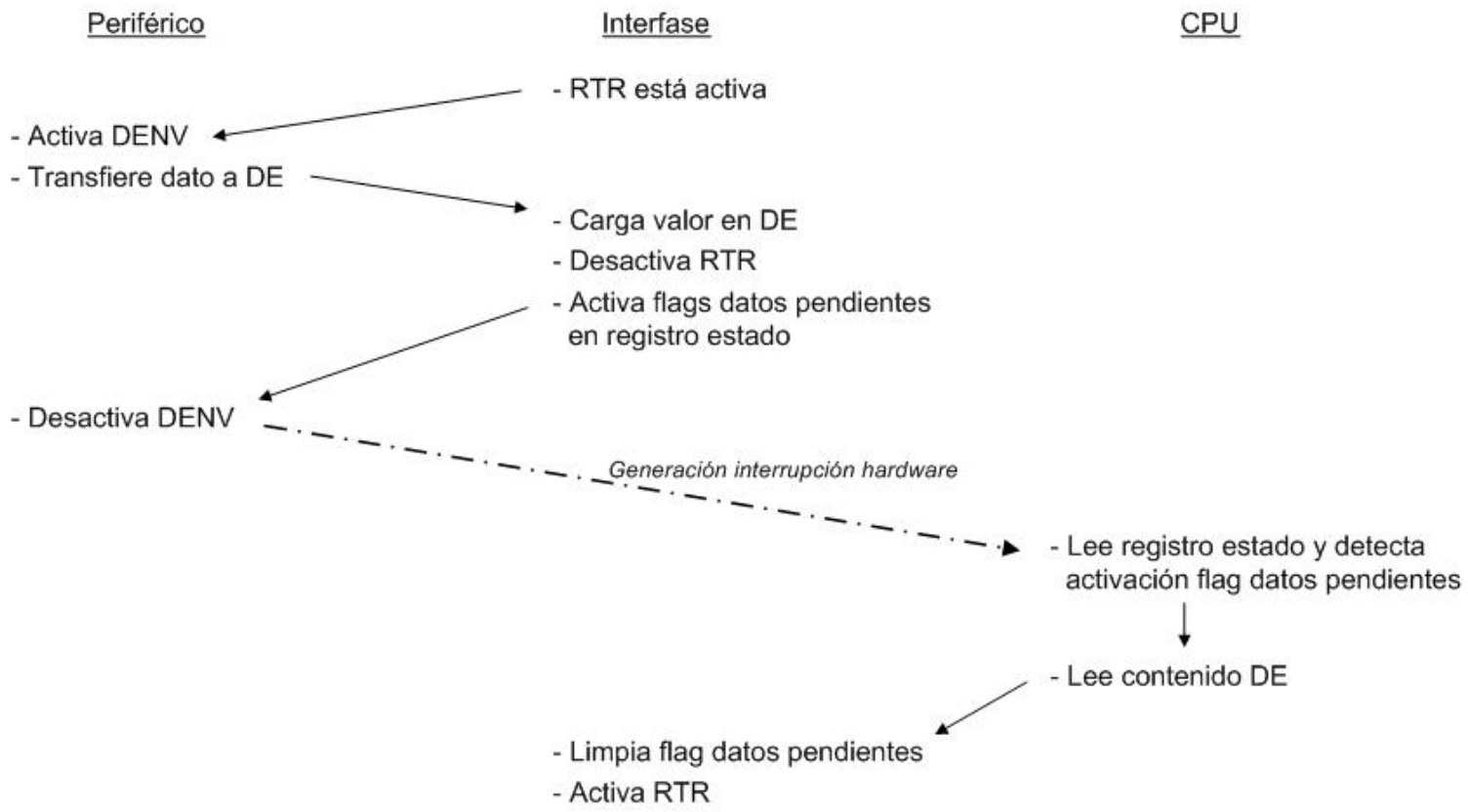
Ejemplo de E/S por PIO

Descripción de una escritura de datos, previamente solicitada desde la CPU



Interrupciones

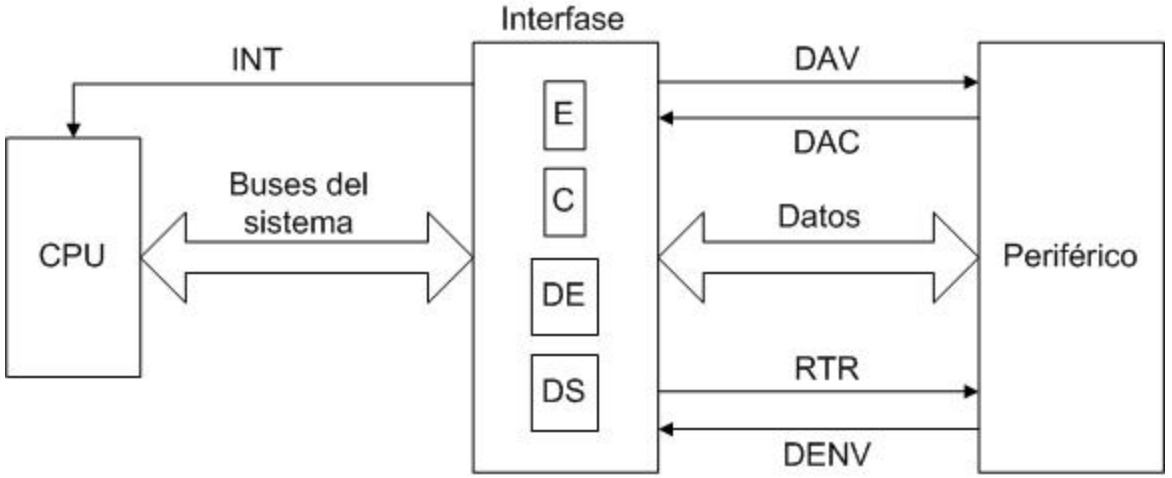
- Uso de interrupciones hardware permite a la interfase generar explícitamente un evento que provoque la respuesta de la CPU
  - Elimina la indeterminación en la temporización de la E/S



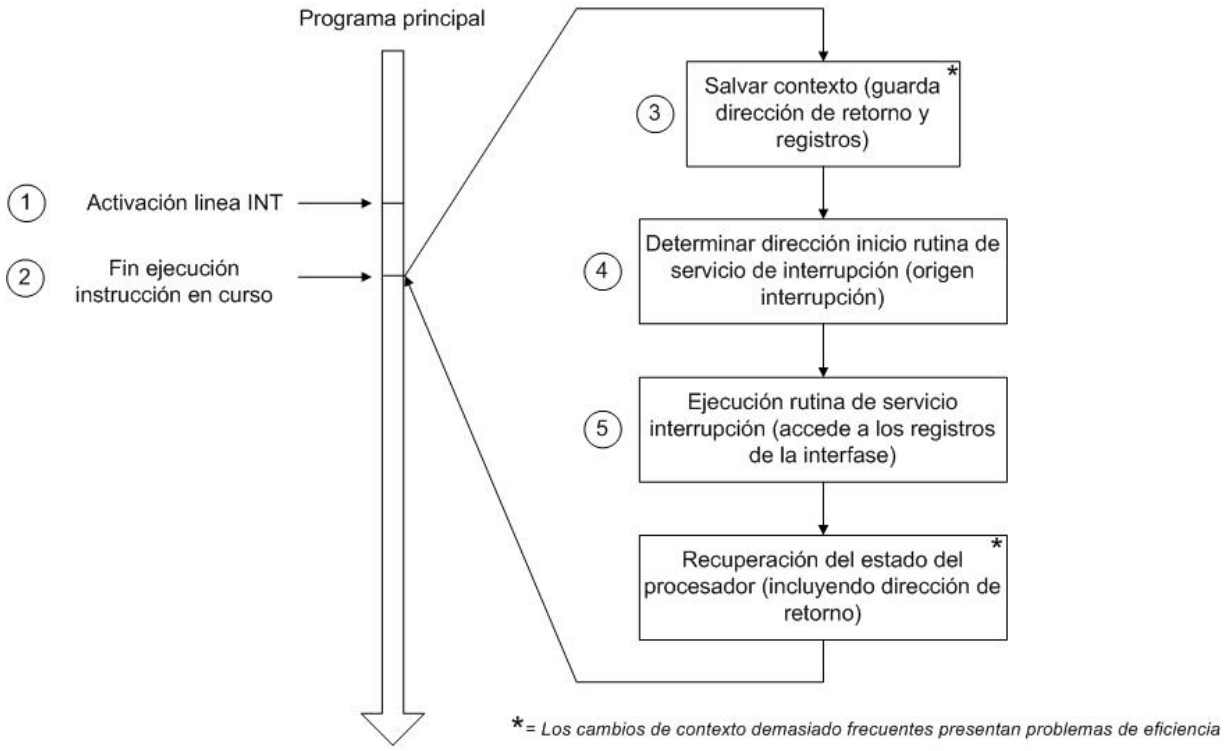
Interrupciones

- Interfase y CPU son dispositivos físicos distintos
  - Interrupción necesita de una línea hardware que los interconecte

- Rutina de servicio de interrupción gestionará la operación de E/S

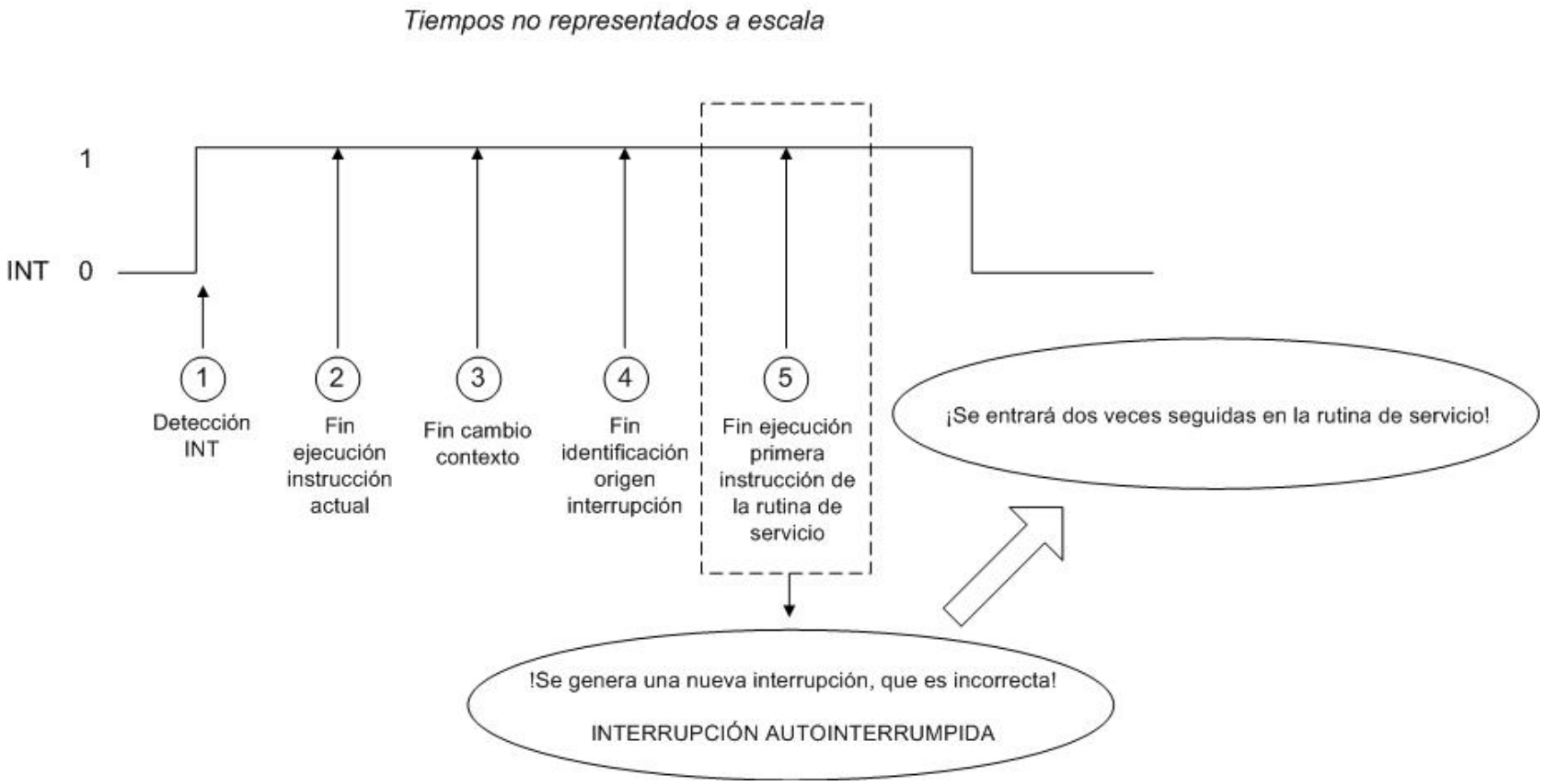


Interrupciones



- Esquema básico de servicio de una interrupción hardware
  - Interrupción se atiende al finalizar la instrucción que está en ejecución cuando se detecta la activación de INT
- Este esquema básico, sin embargo, no funciona

Interrupciones

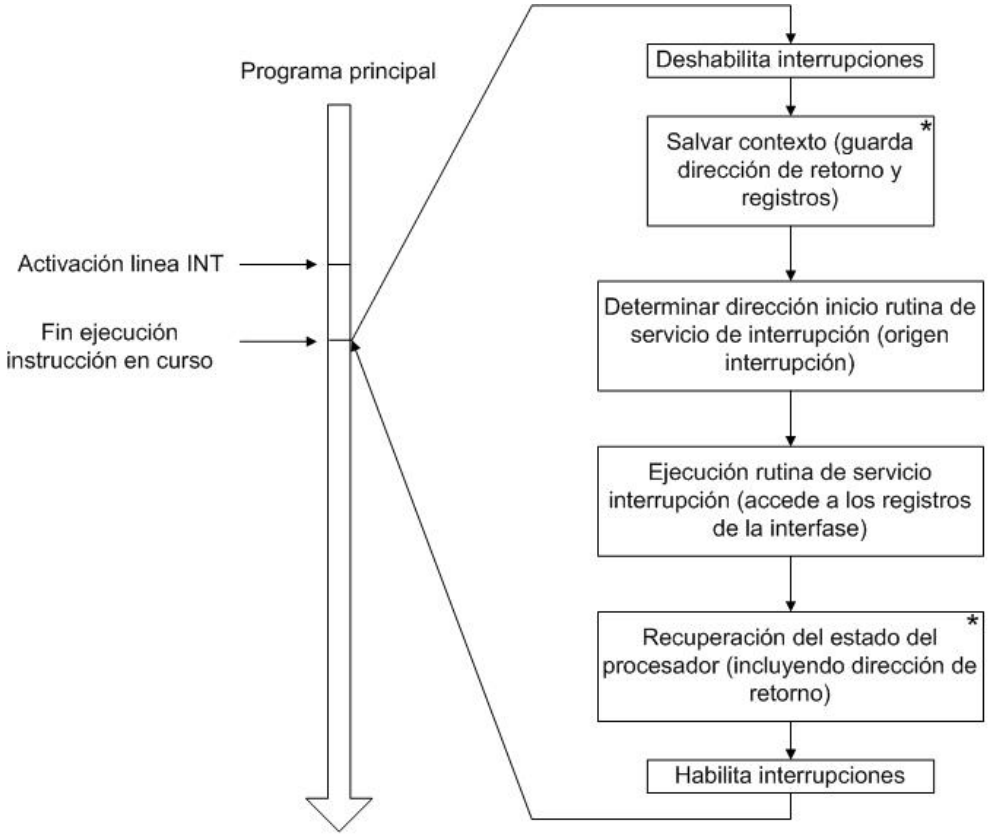


Interrupciones

- Interrupción autointerrumpida provoca serio problema hardware
  - Interfase del periférico es accedida dos veces consecutivas, cuando sólo espera una
  - Potencialmente pude bloquear físicamente el sistema
    - "Pantallazo azul de la muerte" (BOSD) en Windows
    - "Kernel panic" en Linux
- Necesario evitar interrupción autointerrumpida
  - Hay dos formas posibles de hacerlo:
    - Usar como señalización de INT sólo flanco de subida o de bajada
    - Usar enmascaramiento de interrupciones

Interrupciones

- Enmascaramiento bloquea nueva interrupción hasta que termine la actual



*= *Los cambios de contexto demasiado frecuentes presentan problemas de eficiencia*

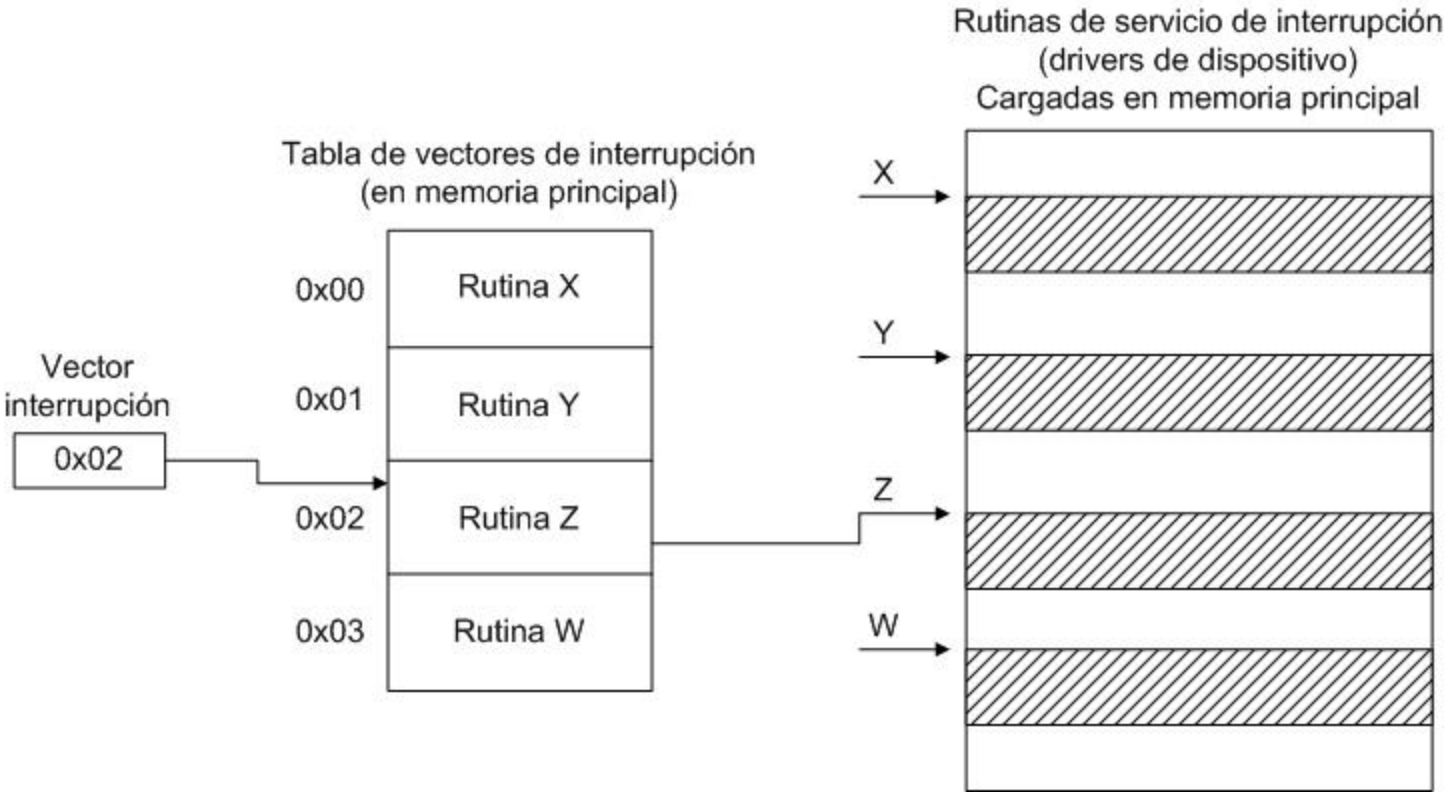
Vector de interrupción

- Interrupción provoca la ejecución de una rutina de servicio
  - Esta rutina debe ser la correcta para acceder a la interfase que ha provocado la interrupción

- *Vector de interrupción* es el mecanismo que enlaza origen de interrupción y rutina de servicio
  - Es un valor entero de 8 bits, que actúa como identificador del origen de la interrupción
  - Proporciona indirectamente un puntero al inicio de la rutina de servicio, a través de una (o más) tablas de vectores de interrupción

Vector de interrupción

Indirección vía tabla de vectores de interrupción



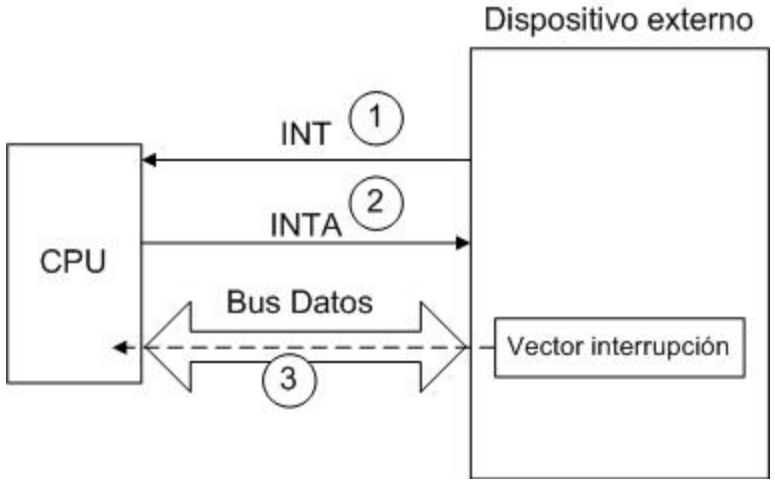
Vector de interrupción

- En interrupciones generadas desde software (excepciones, funciones estilo INT(21h)), el vector de interrupción se genera automáticamente

- Para el resto de las interrupciones, vector de interrupción debe ser entregado por el hardware que produce la interrupción

  - En interrupciones señalizadas vía una línea INT, se entrega durante ciclo especial de reconocimiento de interrupción

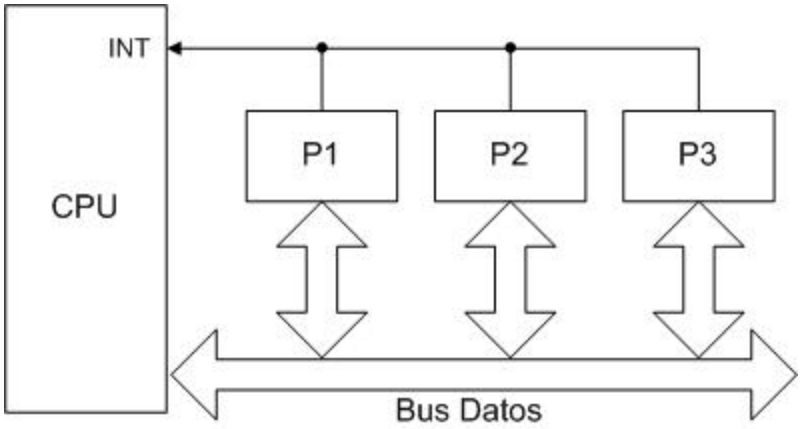
Vector de interrupción



- Secuencia de eventos del proceso es:
  1) Activación linea INT desde dispositivo externo
  2) Al iniciarse el procesamiento de la interrupción, CPU activa la línea de reconocimiento de interrupción, INTA
  3) Dispositivo externo deposita en el bus de datos el valor del vector de interrupción, que es leído por la CPU
  4) Usando este vector de interrupción para obtener el puntero a la rutina de servicio, la CPU continúa el procesamiento de la interrupción

Gestión de orígenes múltiples: polling

- Si múltiples periféricos pueden generar interrupciones, deben compartir uso línea INT
  - Necesario mecanismo para identificar origen de la interrupción
- Mecanismo más sencillo de compartición es por polling
  - Activación INT hace entrar a CPU en rutina de servicio genérica
  - Rutina genérica comprueba uno a uno los registros de estado en las interfaces de los periféricos, hasta encontrar uno con interrupción pendiente
  - Rutina usa entonces vector preestablecido por software para ese dispositivo



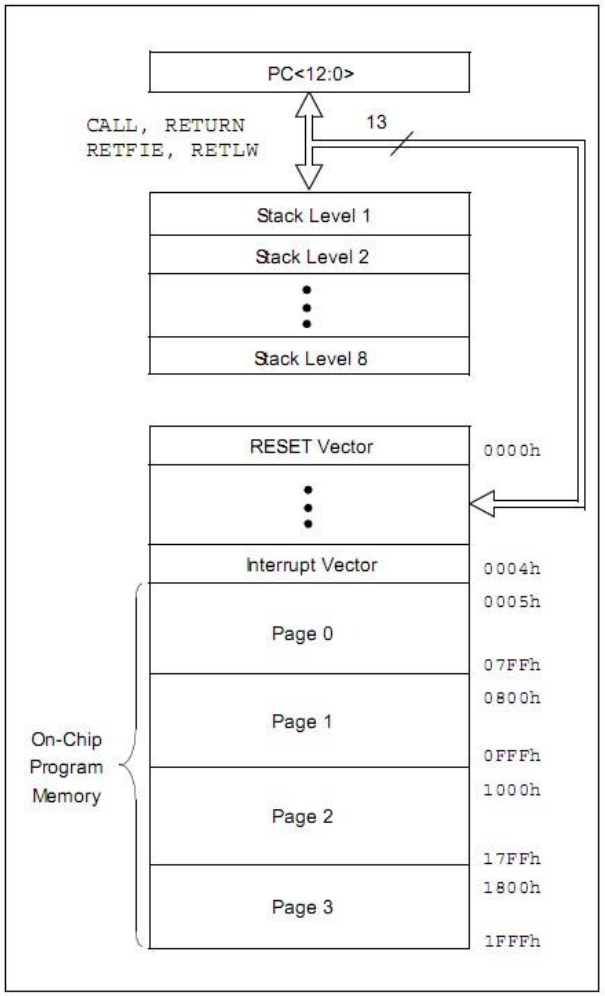
Gestión de orígenes múltiples: polling

- Ventajas:
  - No requiere hardware especializado
  - Prioridad de dispositivos se cambia fácilmente, alterando el orden de muestreo
- Inconvenientes
  - Es relativamente lenta para el último dispositivo de la cola, por necesidad de muestrear todos los anteriores.

Gestión de orígenes múltiples: polling

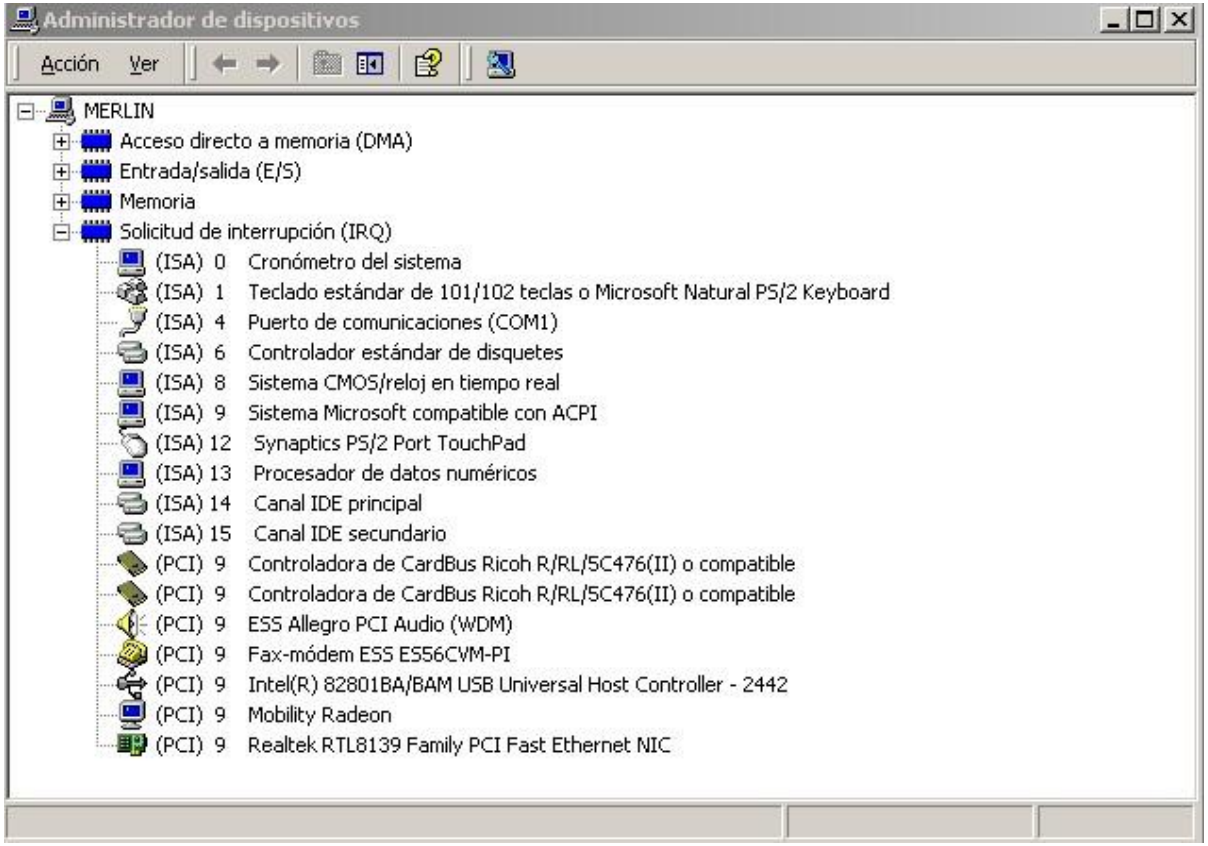
- Usado en sistemas sencillos, que sólo tienen un único vector de interrupciones
  - Ejemplo: microcontroladores
- Figura muestra mapa del espacio de direcciones del microcontrolador Microchip 16F877
  - Hay un único vector de interrupciones, en la dirección 0004h
  - Esa dirección apunta a una rutina común de servicio
  - Necesario polling para identificar orígen de la interrupción

FIGURE 2-1: PIC16F877/876 PROGRAM MEMORY MAP AND STACK



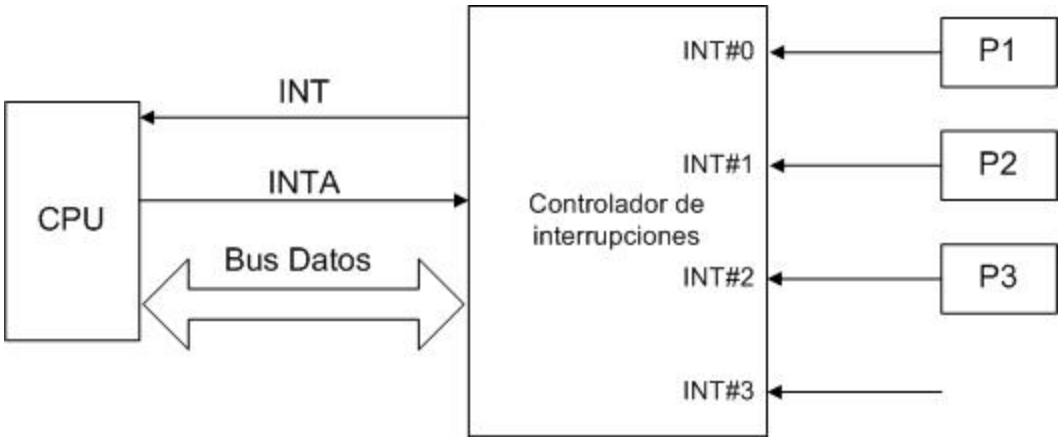
Gestión de orígenes múltiples: polling

- Usado también en sistemas operativos complejos para compartir un mismo vector de interrupción cuando hay un número limitado de vectores hardware de interrupción
  - Ejemplo: Compartición IRQ 9 (*IRQ sharing*) en un Windows XP



Controlador de interrupciones

- Gestión de orígenes múltiples de interrupciones puede hacerse mediante un circuito especializado, llamado *controlador de interrupciones*
  - Intermediario entre periféricos y CPU
  - Cada periférico tiene su propia entrada diferenciada de petición de interrupción.
  - El controlador:
    - Prioriza las peticiones, y determina si se deben o no atender
    - Activa la línea o señal de interrupción
    - Entrega a la CPU el vector de interrupción asociado al periférico

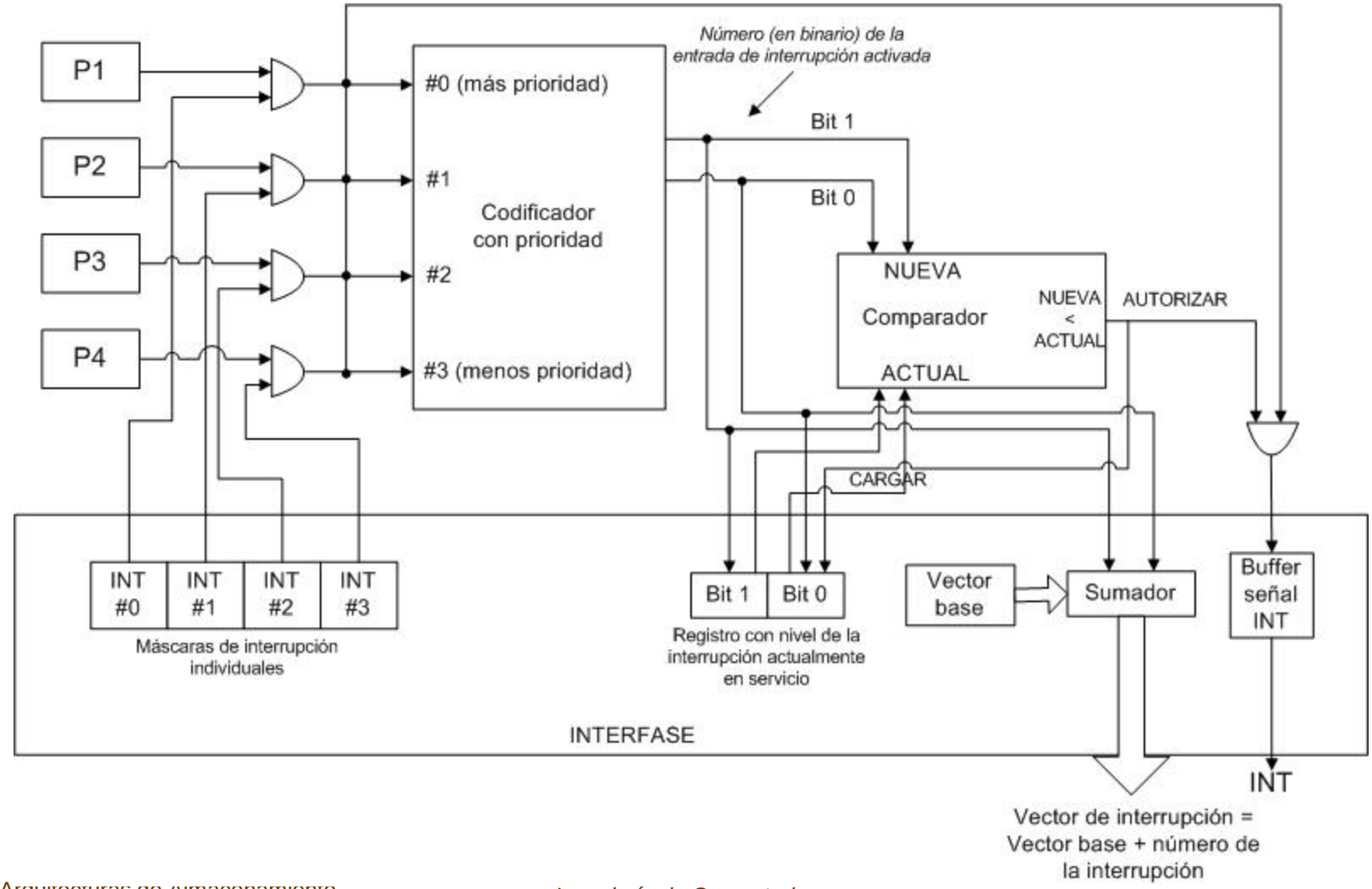


Controlador de interrupciones

- Controlador de interrupciones no es más que un periférico especializado
  - Tiene, por tanto, una interfase con registros y un modelo de programación
- Registros del controlador de interrupciones permiten:
  - Enmascarar interrupciones individuales
  - Evitar interrupciones autointerrumpidas
    - Usa para ello un registro con el nivel de interrupción en servicio
    - Sólo se atiende a una interrupción más prioritaria que la actualmente en servicio
  - Determinar el rango de posibles vectores de interrupción

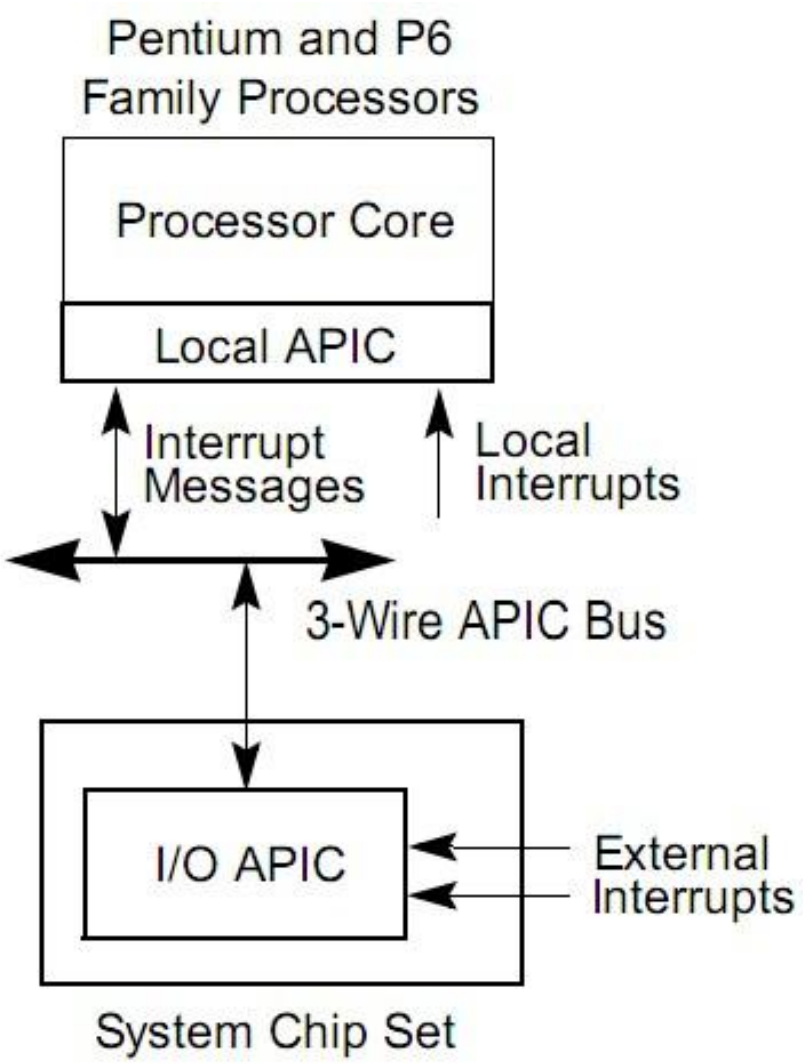
Controlador de interrupciones

- Ejemplo de posible arquitectura interna



MSI (Message Signaled Interrupts)

- El esquema tradicional de interrupciones requiere la activación, desde el controlador de interrupciones externo, de las líneas de interrupción del procesador
  - Complica diseño interfases físicas y lógicas del procesador y del chipset de placa madre
  - El diseño se complica aún más en sistemas multiprocesador

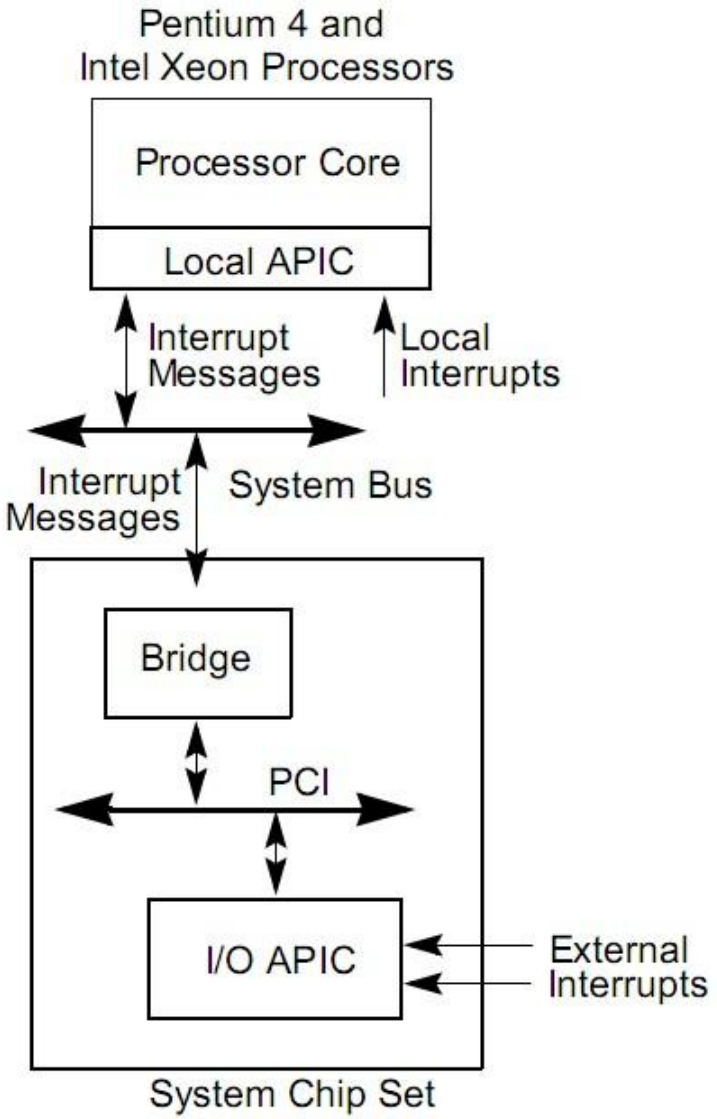


MSI (Message Signaled Interrupts)

- Especificación 2.2 del estándar del bus PCI introdujo una nueva forma de señalizar la necesidad de una interrupción
  - MSI (Message Signaled Interrupts) = Interrupciones señalizadas por mensaje
- Interrupción MSI no emplea la línea de petición INT
  - Interrupción es señalizada desde el controlador de interrupciones realizando una operación de escritura a una cierta dirección de memoria, especificada por el sistema
  - MSI es, por tanto, un caso especial de E/S mapeada por memoria

MSI (Message Signaled Interrupts)

- Al ser una escritura a memoria, mensaje es transmitido a través de los buses de datos del sistema
- Mensaje es interpretado por controlador de interrupciones (APIC) local del procesador
  - APIC local es el que activa el procesamiento hardware de la interrupción



MSI (Message Signaled Interrupts)

- Dirección a la que hay que escribir se obtiene de un registro en la interfase del controlador de interrupciones
  - Dirección destino siempre empieza por 0FEEh

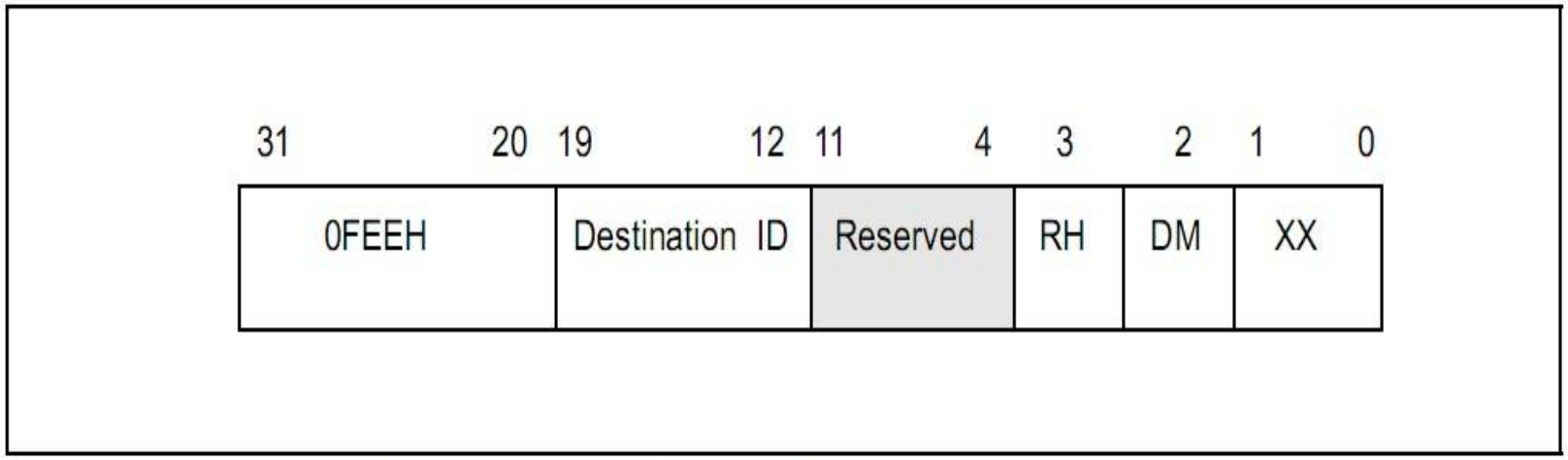


Figure 8-24. Layout of the MSI Message Address Register

MSI (Message Signaled Interrupts)

- Interfase de la CPU ofrece en esa dirección de destino un registro, llamado Message Data Register
  - Con una sóla operación la escritura carga en el registro todos los parámetros de la petición de interrupción

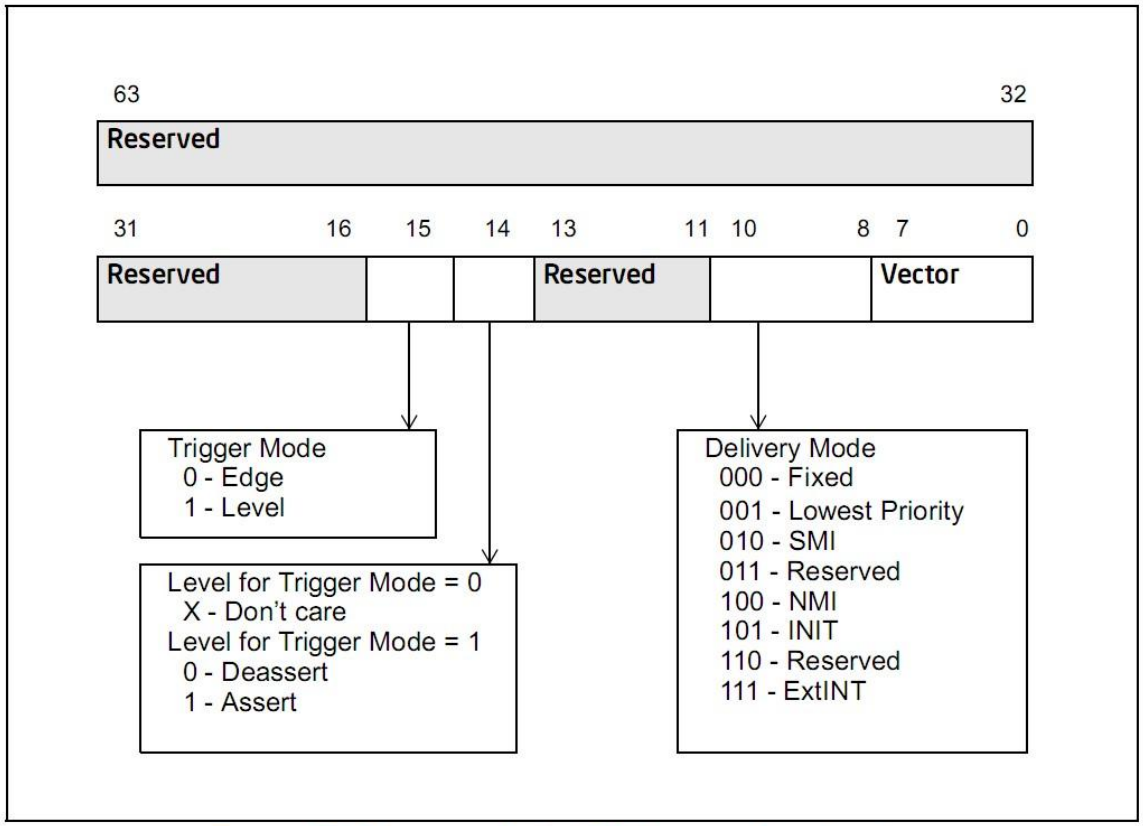
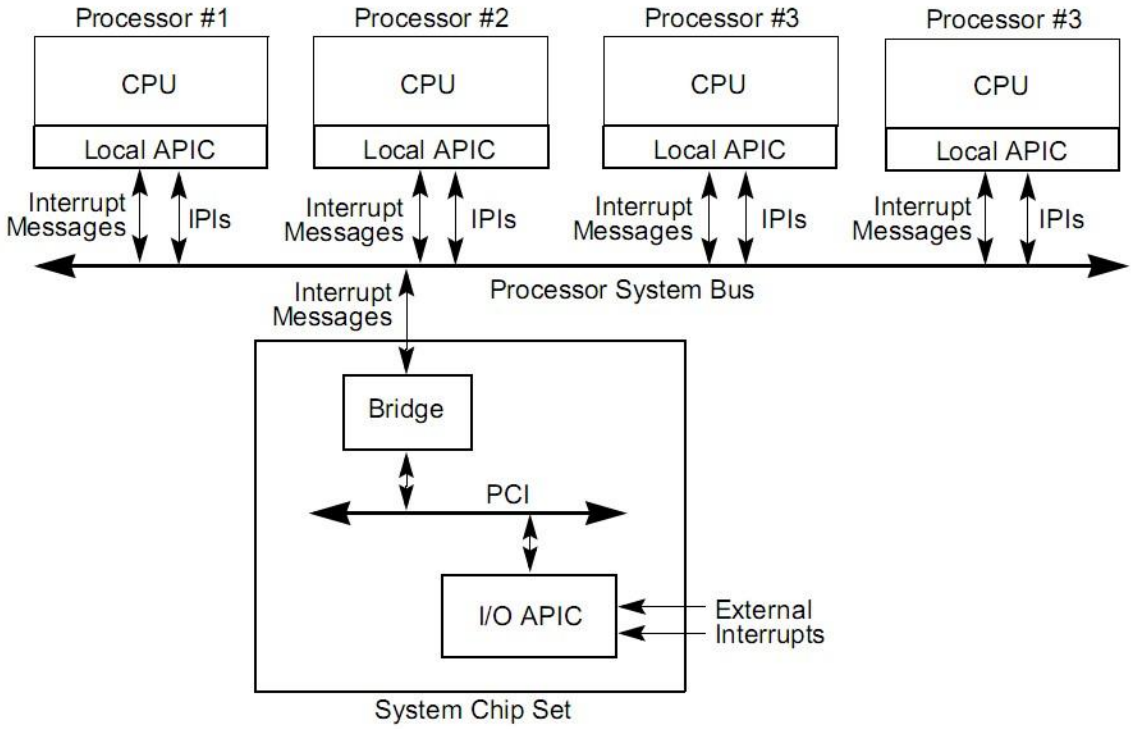


Figure 8-25. Layout of the MSI Message Data Register

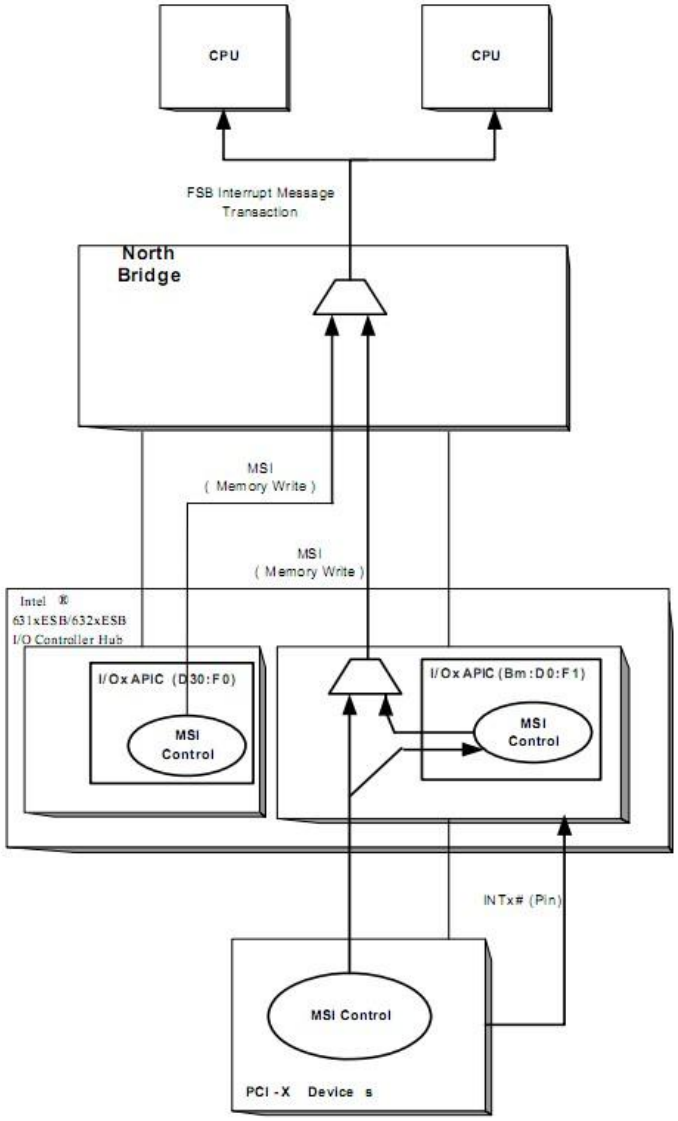
MSI (Message Signaled Interrupts)

- Uso de MSI simplifica diseño de multiprocesadores
  - El mismo bus lleva las transacciones CPU – M y las peticiones de interrupción
  - Sincronización con la operación de interrupción es mucho más fácil



MSI (Message Signaled Interrupts)

- Uso de MSI simplifica también diseño e interfases de chipsets
  - Interrupción es enrutada a través de los integrados del chipset (Puente Norte/Puente Sur) usando el mismo bus por el que pasan los datos
  - Ya no se necesita llevar por la placa madre lineas de activación de interrupciones
- Todos los ordenadores modernos señalizan sus interrupciones usando solamente MSI



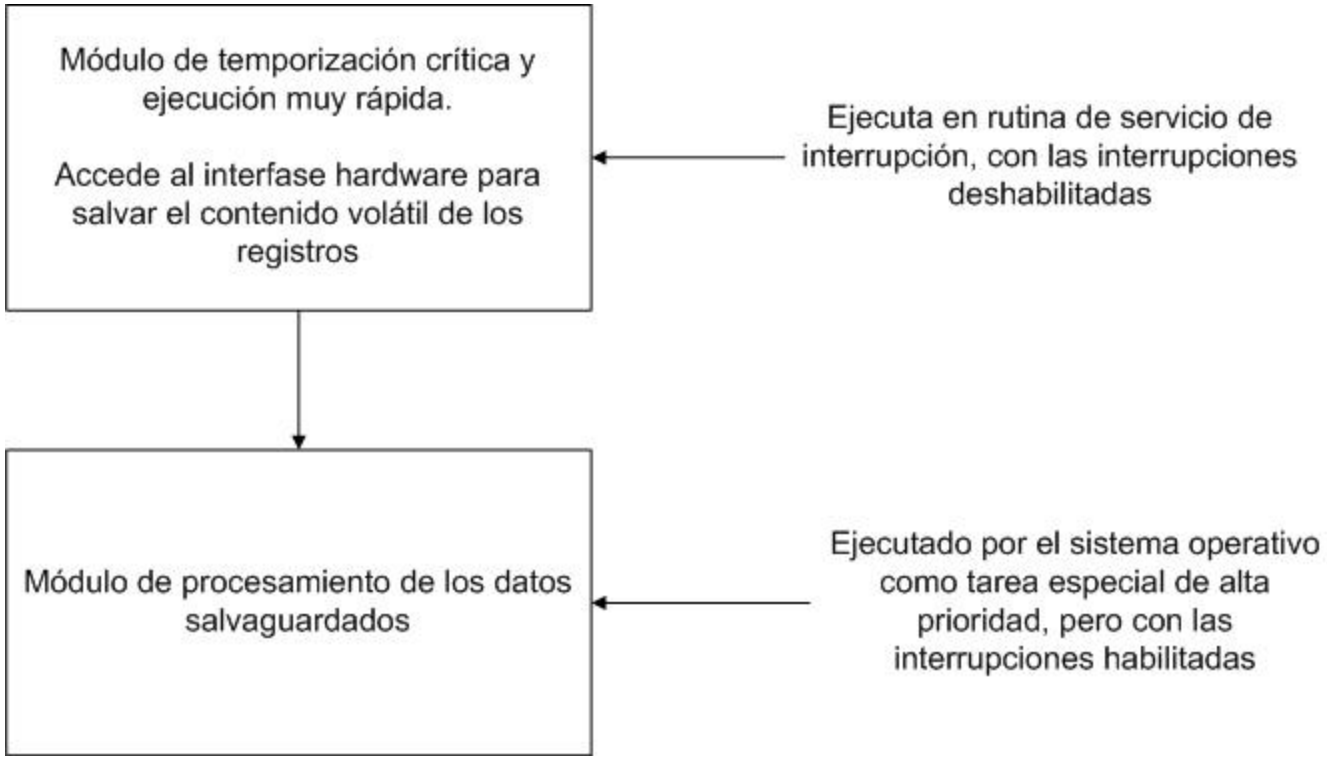
Servicio diferido de interrupciones

- Complejidad de procesadores modernos hace casi mandatorio deshabilitar totalmente las interrupciones durante la rutina de servicio
  - Esto crea problemas si la rutina necesita un procesamiento largo

- Solución empleada hoy día por todos los sistemas operativos es dividir el procesamiento de la interrupción en dos partes
  - En Windows, la tarea en que se procesan los datos es llamada DPC (Deferred Procedure Call)
  - En Linux, los módulos son llamados *upper half* (mitad superior) y *bottom half* (mitad inferior)

Servicio diferido de interrupciones

- La rutina de servicio encola en el sistema operativo la tarea de procesamiento pendiente, que se ejecutará de forma automática y a la mayor brevedad posible



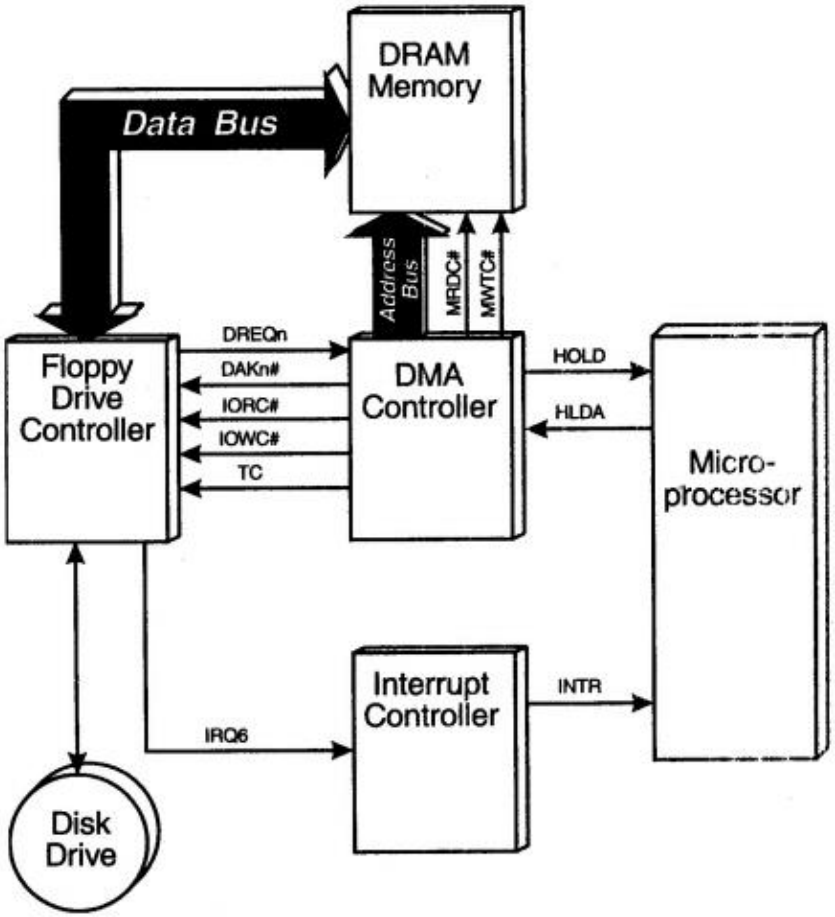
E/S por DMA

- E/S por programa (PIO) enlentece el ordenador, porque involucra a CPU en cada movimiento de datos

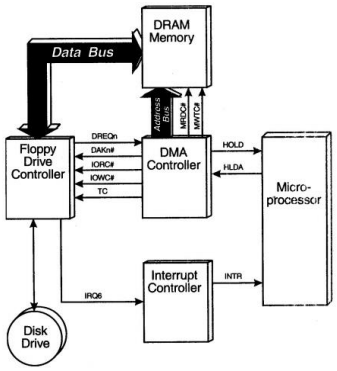
- E/S por DMA transfiere los datos sin necesidad de usar la CPU para moverlos
  - Datos son intercambiados directamente entre memoria principal e interfase del periférico
  - Requiere el uso de hardware especial (controlador DMA) para que realice en cada transferencia los direccionamientos necesarios (asignaciones de valores a líneas de bus de direcciones y bus de control)
    - El controlador DMA **NO** mueve los datos; sólo controla su movimiento

E/S por DMA

- Figura muestra arquitectura DMA tradicional (en controlador diskettes del PC)
  - En sistemas simples se siguen usando esquemas similares



E/S por DMA



- Observar que bus datos permite mover datos directamente entre floppy y memoria
  - Controlador DMA sólo dará valores a líneas del bus de direcciones y a algunas líneas de control

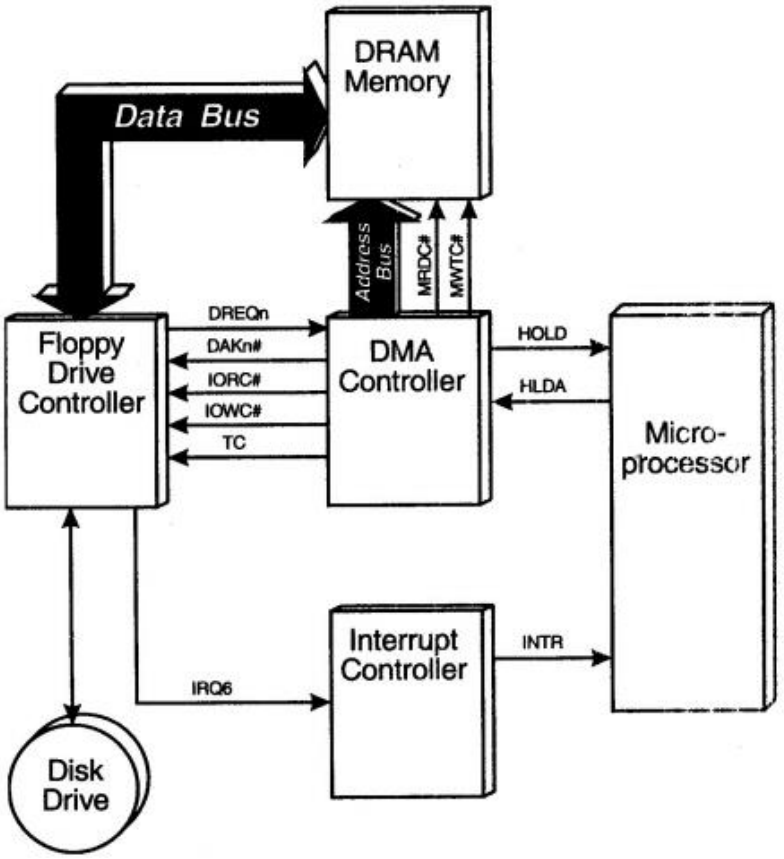
- Si transferencia DMA usa bus de direcciones y bus de datos, CPU no puede emplearlos durante la transferencia
  - CPU pone en alta impedancia sus conexiones a estos buses (se desconecta físicamente de ellos)
  - No puede, por tanto, leer instrucciones desde memoria
  - Necesario parar CPU durante la transferencia

E/S por DMA

- Secuencia de transferencia DMA:

1) CPU programa transferencia en registros de las interfases de controlador de floppy y de controlador DMA
2) CPU ordena a controlador floppy iniciar una operación (lectura/escritura) a través de registro de comandos de su interfaz
3) Controlador floppy activa DREQ para solicitar transferencia DMA

(sigue...)



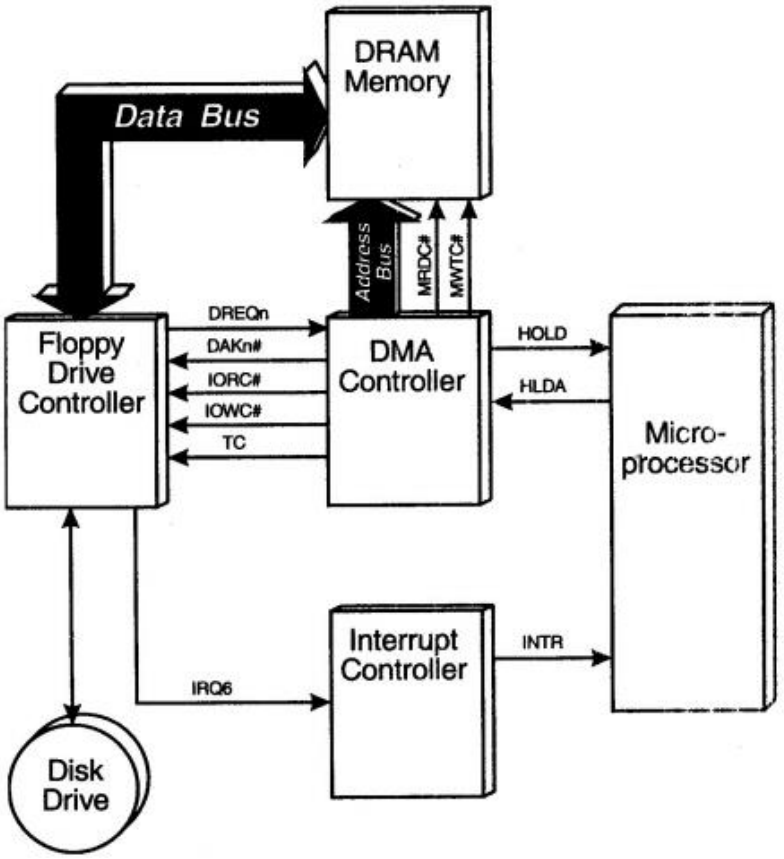
E/S por DMA

4) Controlador DMA activa HOLD para solicitar a CPU el control de los buses

5) CPU detiene su operación, cede los buses desconectándose de ellos, y activa HLDA (HOLD Acknowledge) para indicarlo

6) Controlador DMA activa DAK para indicar a floppy inicio transferencia

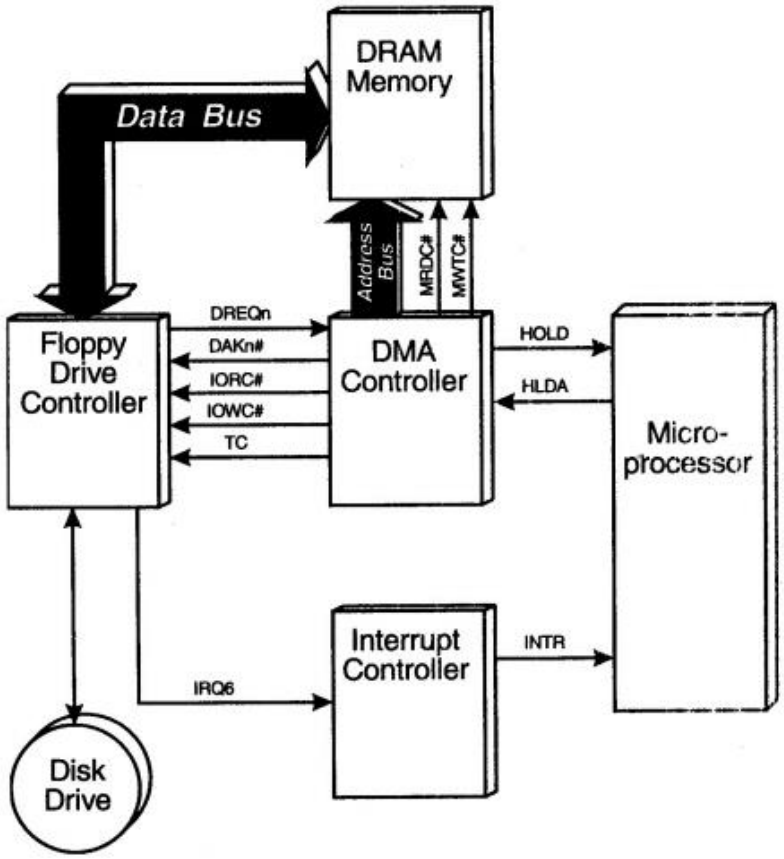
(sigue...)



E/S por DMA

7) Para cada dato del bloque a mover, controlador DMA direcciona posición de memoria, y activa líneas de control (strobes) que mueven el dato
   - Direccionamiento en floppy es automático, porque datos salen de un único registro de su interfaz

8) Al finalizar la transferencia, controlador DMA libera buses y desactiva HOLD
   - CPU retoma operación normal

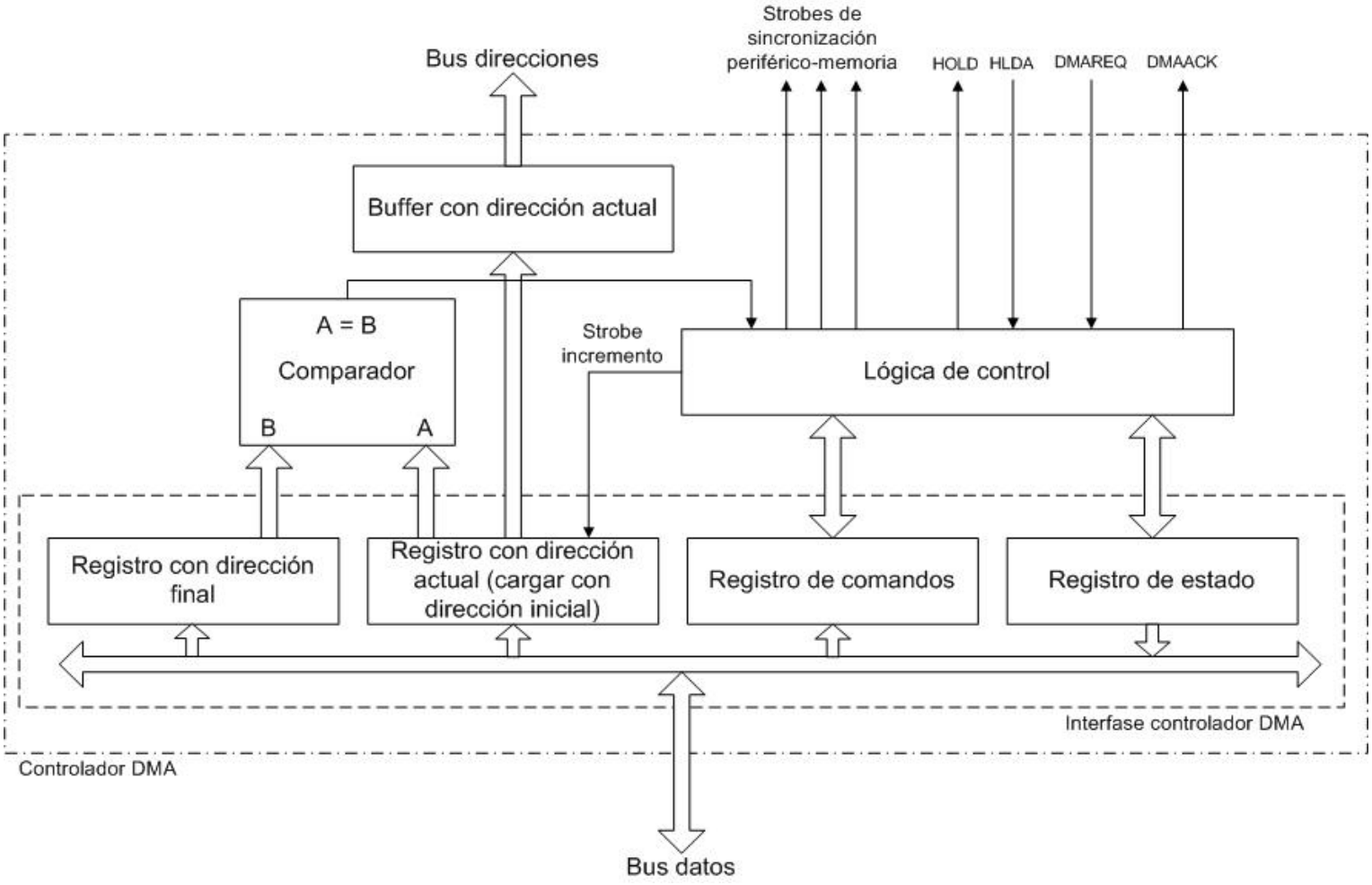


E/S por DMA

- Necesidad parar al procesador implica que bloque no puede ser muy largo
  - Movimiento del bloque, sin embargo, es más rápido que en PIO, porque cada transferencia individual se hace más rápidamente

E/S por DMA

- Figura muestra una posible implementación interna de un controlador simple de DMA

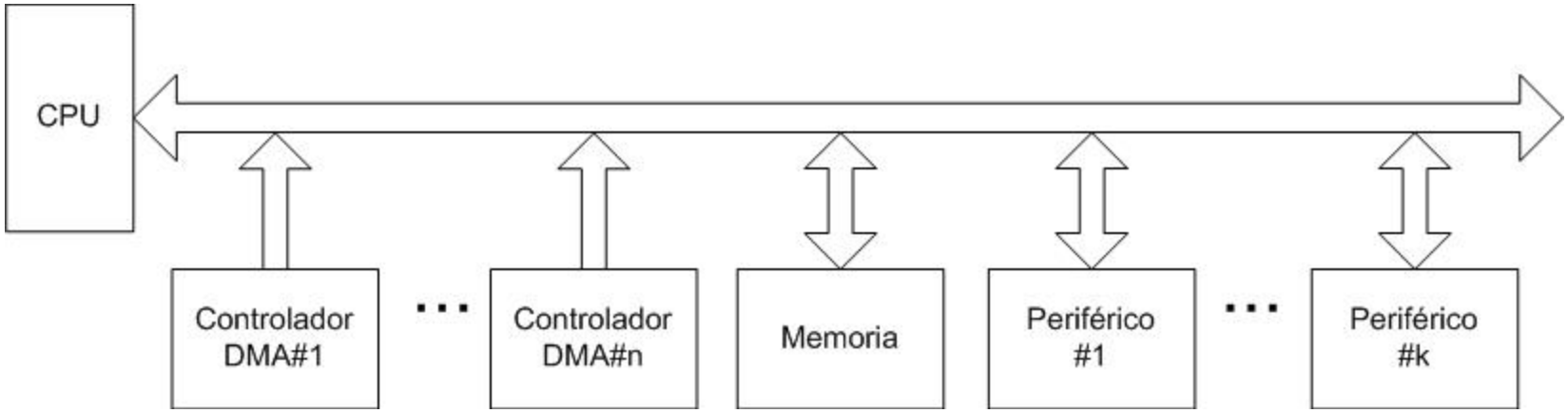


E/S por DMA

- CPU programa transferencia cargando (en modo PIO) registros del controlador DMA con:
  - Dirección inicio de la transferencia
  - Número de datos a mover (o dirección final de la transferencia)

Control de acceso al bus: Arbitración

- En el caso más general, en un mismo ordenador tendremos varios dispositivos capaces de dar valores al bus de direcciones y líneas de control:
  - Al menos una CPU
  - Varios controladores DMA



Control de acceso al bus: Arbitración

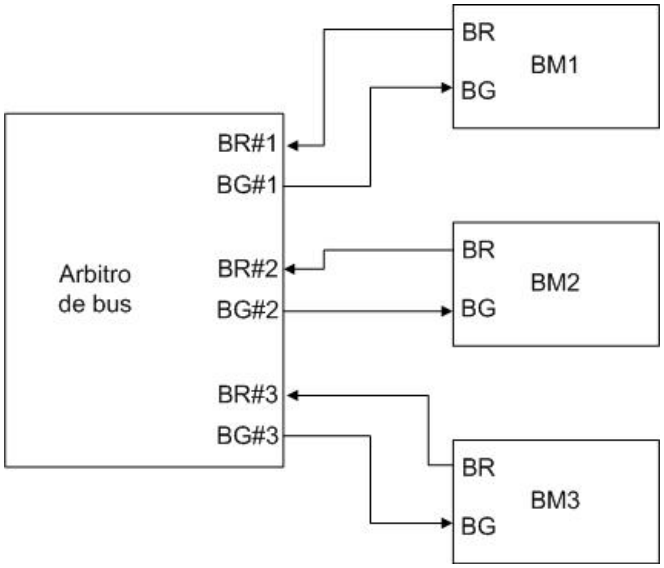
- En cada momento sólo uno de estos dispositivos puede estar dándole valores a estos buses

- Es necesario, por tanto, un mecanismo de coordinación que determine en cada transferencia cual es el dispositivo que tomará el control de los buses
  - Se llama *arbitración de bus* a este mecanismo de coordinación

- Arbitración requerirá siempre un intercambio de señales
  - Dispositivo que quiere controlar el bus activará una línea de petición
  - El ganador del proceso de arbitración se indicará activando una línea de cesión para ese dispositivo

Arbitración paralela centralizada

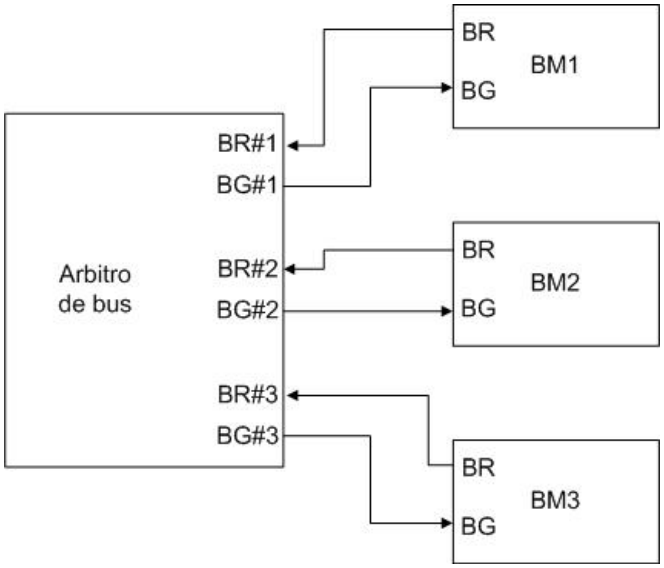
- Usa un circuito, llamado árbitro de bus, que es el encargado, antes de iniciar cada transacción, de seleccionar qué dispositivo tomará el control del bus
- Arbitro usa para cada posible controlador del bus una pareja independiente de líneas:
  - BR = Bus Request = petición de bus
  - BG = Bus Grant = cesión de bus



Arbitración paralela centralizada

- Todos los dispositivos que quieran competir por el bus activan sus líneas BR

- Arbitro decide el ganador, y se lo indica activando, sólo a él, su línea BG



Arbitración paralela centralizada

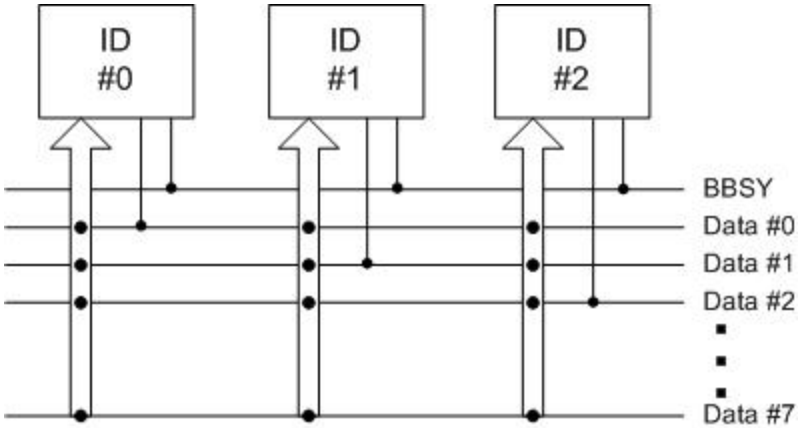
- Ventajas:
  - Sencillo de implementar para los controladores de bus
  - Permite uso de algoritmos complejos, en el árbitro, para controlar prioridades y evitar acaparamiento del bus por un solo dispositivo
  - No usa las líneas del bus de datos para arbitrar
    - Arbitración puede, por tanto, realizarse en paralelo a transferencias de datos

- Inconvenientes:
  - Requiere la presencia, en la placa madre, del circuito árbitro de bus
  - Requiere enrutar las líneas de petición y cesión entre dispositivos y árbitro de bus
    - No viable si se quiere minimizar el número de líneas del bus, o si tienen longitudes largas

Arbitración distribuida con autoselección

- En la arbitración distribuida no existe un circuito único central que resuelve la arbitración
- Función de arbitración debe ser implementada, de forma cooperativa, por la operación simultánea de todos los dispositivos conectados al bus
- Cada dispositivo tiene su propia circuitería de control de arbitración
  - Más sencilla que la del árbitro centralizado

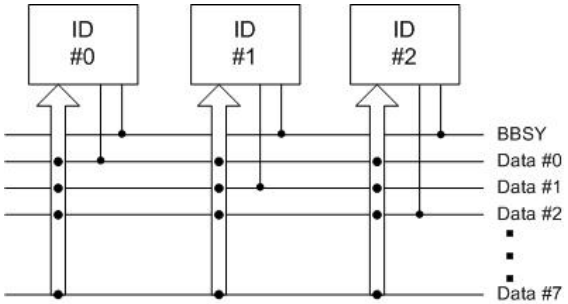
Arbitración distribuida con autoselección



- Este ejemplo muestra un modelo simplificado de arbitración del bus paralelo SCSI

- Cada dispositivo SCSI debe tener asignado un identificador numérico único ID, del 0 al 7
  - Asignación de ID hace que el dispositivo, durante la arbitración, tome control de la línea del bus de datos que tenga ese número de orden

Arbitración distribuida con autoselección



- Al inicio de la arbitración cada dispositivo que quiera controlar bus pone a 1 su línea
  - Existe priorización, con ID=7 máxima prioridad, ID=0 mínima
- Tras poner a 1 su línea de petición, dispositivo monitoriza líneas de bus de datos
  - Si un dispositivo más prioritario solicita el control del bus, los dispositivos de menor prioridad que él deben retirar sus solicitudes y poner a 0 sus líneas
- Gana arbitración el último dispositivo que mantenga su línea de petición a 1
  - Lo indica al resto del sistema activando BBSY (= Bus Busy)

Arbitración distribuida con autoselección

- Ventajas:
  - En buses largos no se requiere enrutar un par de líneas de arbitración por cada dispositivo
    - Reduce el número total de líneas del bus
  - Es más sencillo sincronizar la arbitración entre dispositivos de diferentes velocidades

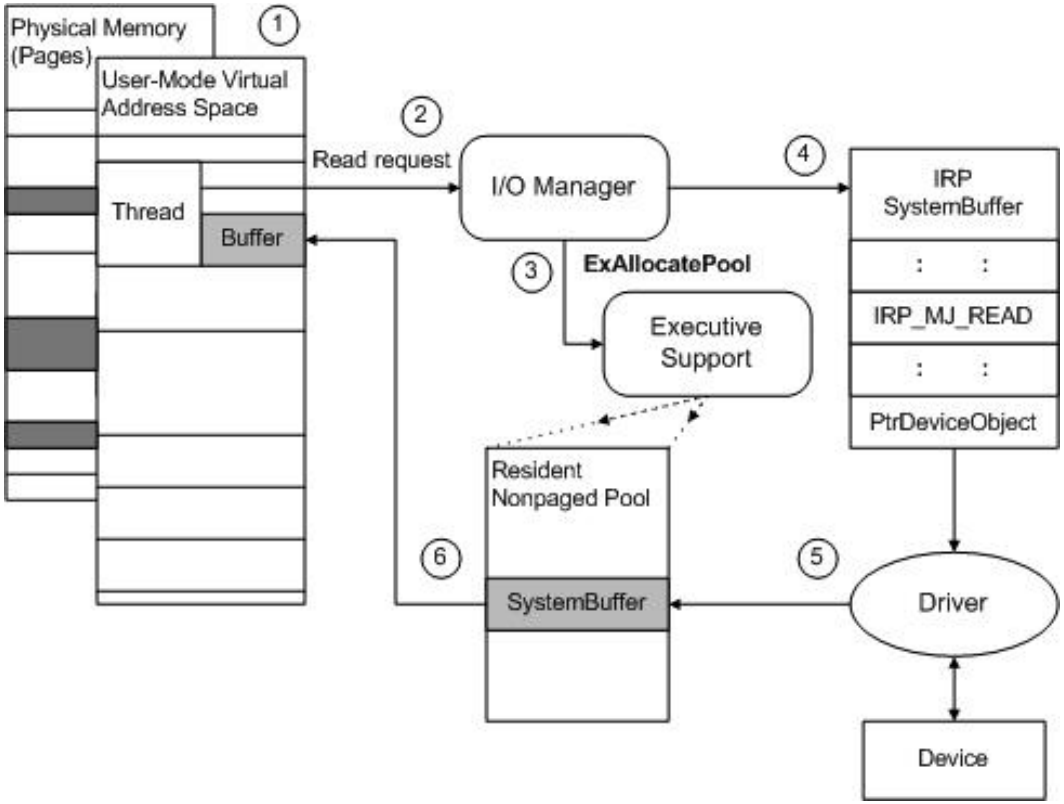
- Inconvenientes:
  - La función de control a implementar en cada dispositivo es más compleja que en la arbitración centralizada
  - Necesita usar las líneas del bus de datos durante la arbitración
  - Por tanto, las arbitraciones no pueden ser muy frecuentes, o sufrirá la eficiencia del bus

DMA con E/S por buffer (Buffered I/O)

- En buffered I/O, transferencias DMA requieren paso de datos a través de una zona intermedia de memoria (buffer), gestionada por sistema operativo
- Las operaciones de E/S con buffer requieren dos pasos
  - En escritura (Write):
    1. Sistema operativo copia datos de memoria de programa a buffer
    2. Transferencia DMA mueve bloque de datos de buffer a periférico
  - En lectura (Read):
    1. Transferencia DMA mueve bloque de datos de periférico a buffer
    2. Sistema operativo copia datos de buffer a memoria de programa
- Transferencia datos entre buffer y memoria suele ser realizada mediante DMA memoria - memoria

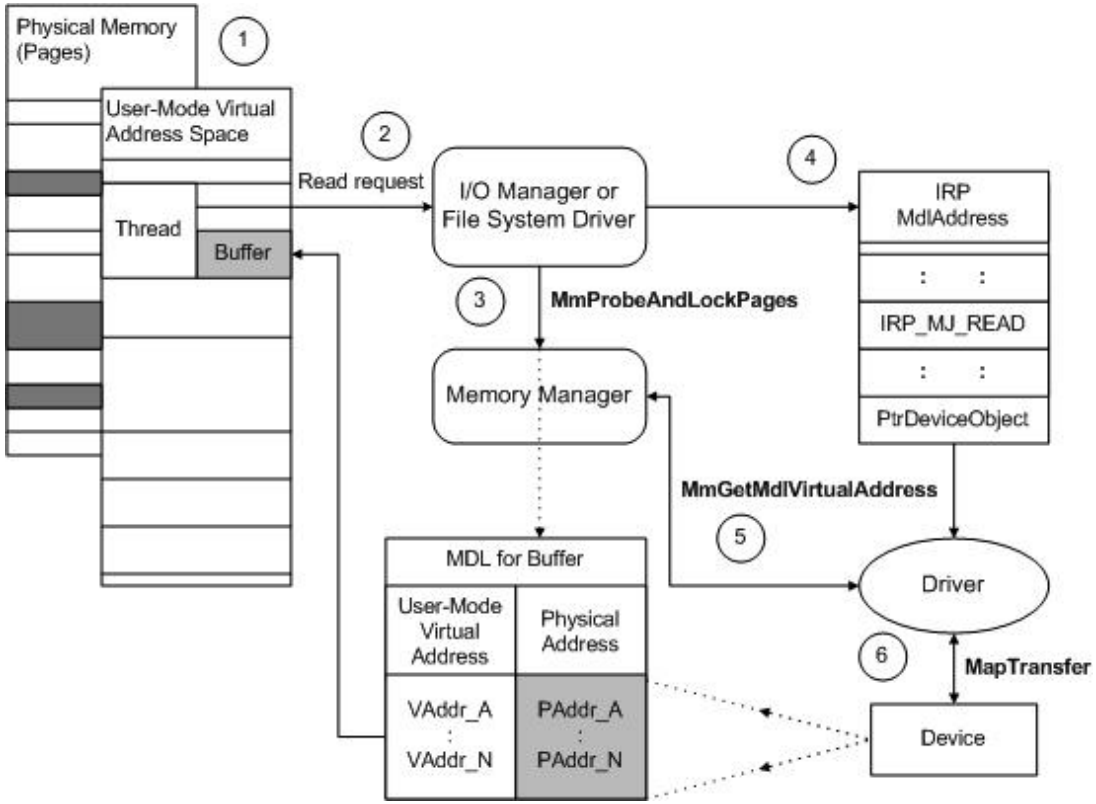
DMA con E/S por buffer (Buffered I/O)

- Propósito de buffer es dar a DMA bloque de direcciones contiguas para la transferencia
  - Permite que driver DMA sea más sencillo de implementar



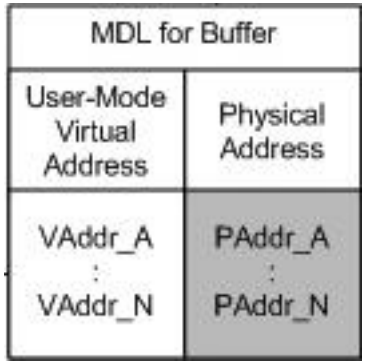
DMA con E/S directa (Direct I/O)

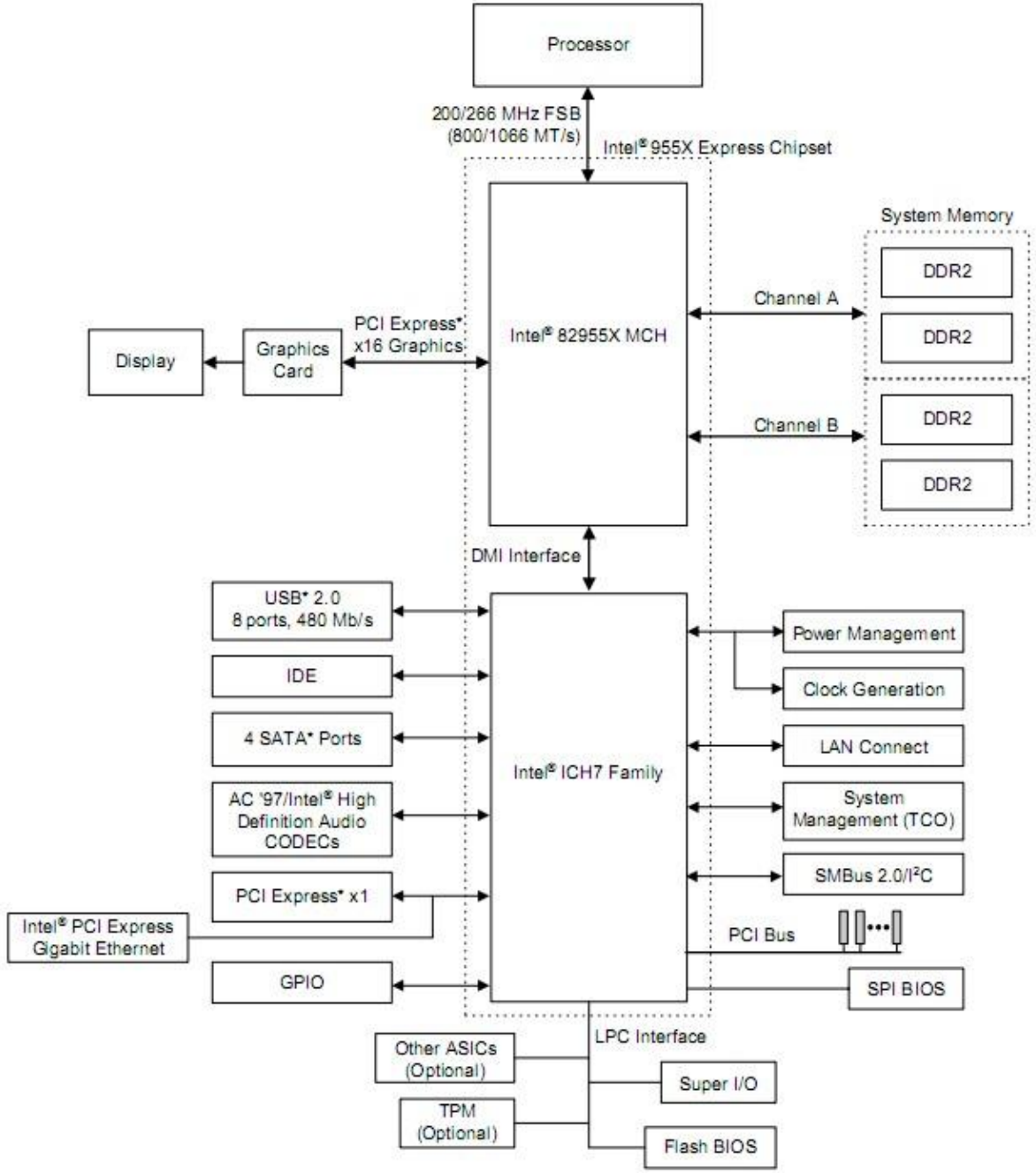
- En modo Direct I/O, transferencias DMA se realizan directamente entre memoria principal y periférico
  - No se emplea buffer intermedio
  - Es, por tanto, más rápida, al no requerir copia memoria-memoria para preparar el buffer o usar sus contenidos



DMA con E/S directa (Direct I/O)

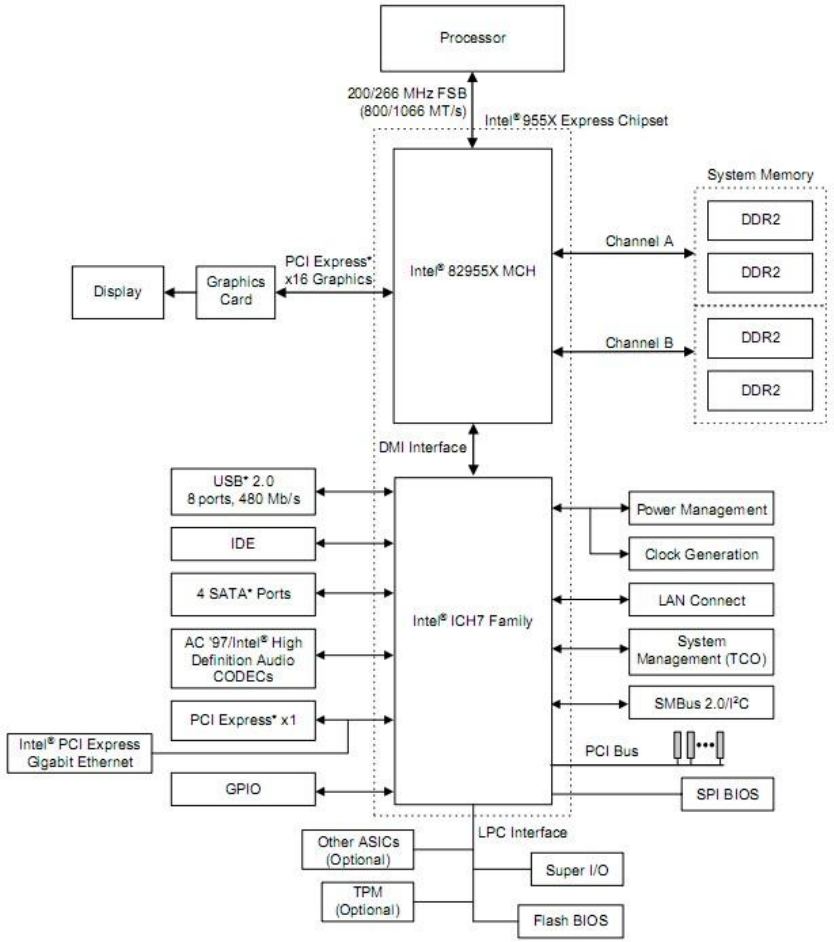
- Direct I/O requiere más sofisticación en controlador y driver DMA
  - Debe ser capaz de gestionar en una única transferencia, sin solución de continuidad, un conjunto de bloques no contiguos de memoria
  - Esta capacidad se denomina *scatter-gather*
  - Controlador DMA necesita como parámetro de la transferencia una lista de scatter-gather
    - Listado de direcciones de inicio y longitudes de los bloques




Processor
200/266 MHz FSB
(800/1066 MT/s)
Intel® 955X Express Chipset
System Memory
DDR2
DDR2
DDR2
DDR2
Channel A
Channel B
Intel® 82955X MCH
PCI Express*
x16 Graphics
Graphics
Card
Display
DMI Interface
USB* 2.0
8 ports, 480 Mb/s
IDE
4 SATA* Ports
AC '97/Intel® High
Definition Audio
CODECs
PCI Express* x1
Intel® PCI Express
Gigabit Ethernet
GPIO
Intel® ICH7 Family
Power Management
Clock Generation
LAN Connect
System
Management (TCO)
SMBus 2.0/I²C
PCI Bus
SPI BIOS
LPC Interface
Other ASICs
(Optional)
TPM
(Optional)
Super I/O
Flash BIOS

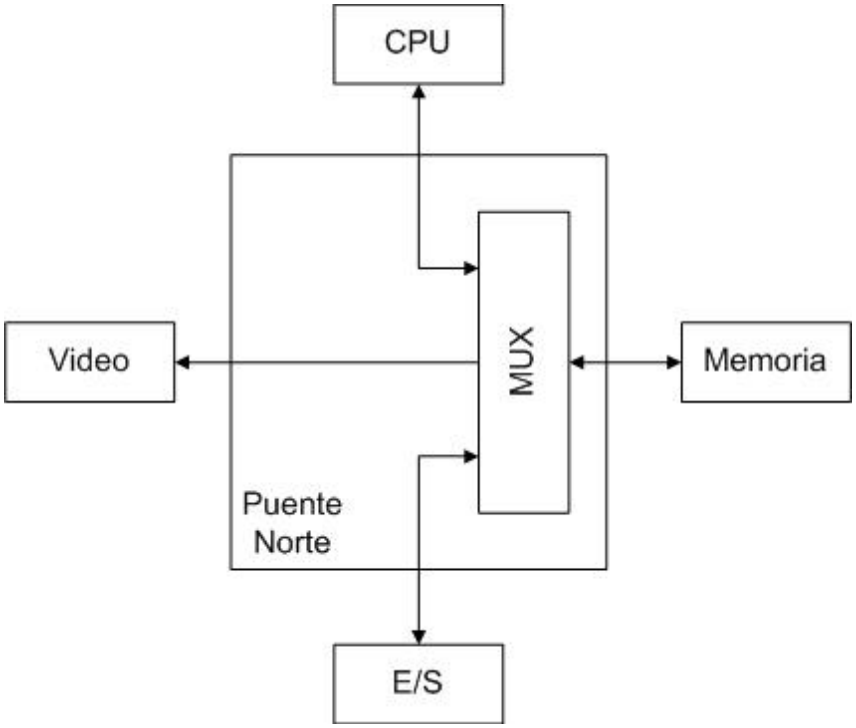
Implementación DMA en chipsets modernos

- DMA en arquitecturas modernas ya no impacta tanto en CPU
  - No se necesita detener CPU durante acceso DMA a memoria
- Chipsets introducen un controlador de memoria intermediario entre procesador y DRAM
  - Llamado *Puente Norte* (*North Bridge*) o *Memory Controller Hub* (MCH)
- Puente Norte proporciona buses diferenciados para:
  - accesos a CPU
  - accesos a memoria
  - accesos a E/S



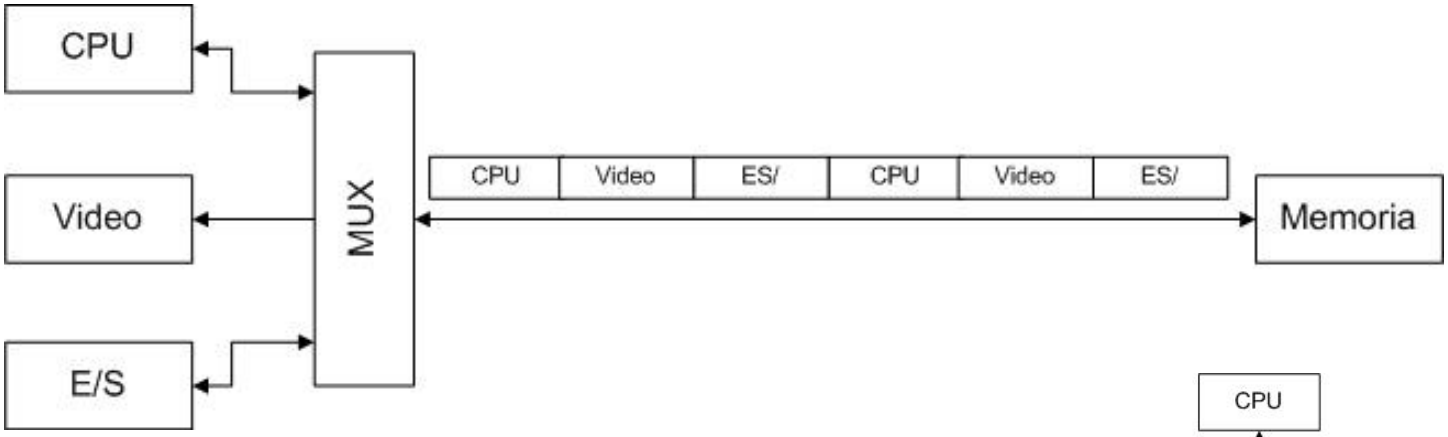
Implementación DMA en chipsets modernos

- Puente Norte desacopla acceso a memoria, acceso a procesador y acceso a E/S
  - Permite la operación concurrente de todos estos accesos

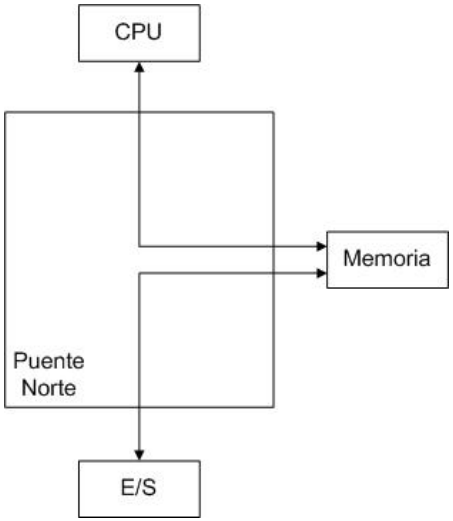


Implementación DMA en chipsets modernos

- Bus acceso a memoria es mucho más rápido que los otros buses
  - Permite la multiplexación en el tiempo de todos los accesos

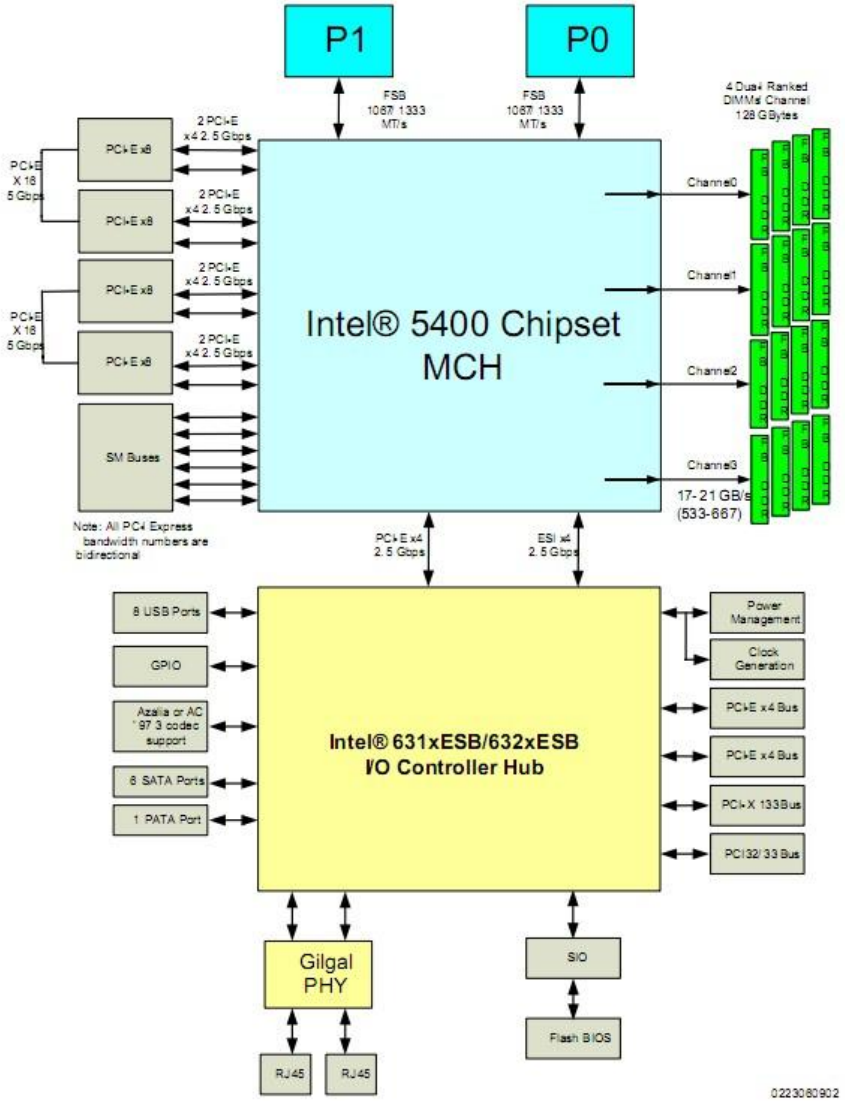


  - A efectos prácticos, el resultado es como si existieran caminos paralelos simultáneos para acceder concurrentemente a memoria desde CPU y E/S



Implementación DMA en chipsets modernos

- Ya no es necesario interferir en la operación de la CPU para hacer transferencias DMA
  - Especialmente importante en multiprocesadores, donde operación ordenada de las CPUs es imprescindible para sincronizar el sistema
  - Controladores de acceso a memoria (MCH) modernos también vigilan la coherencia de caché en los accesos a DMA



II – Buses e interfaces

Diferenciación entre buses e interfaces

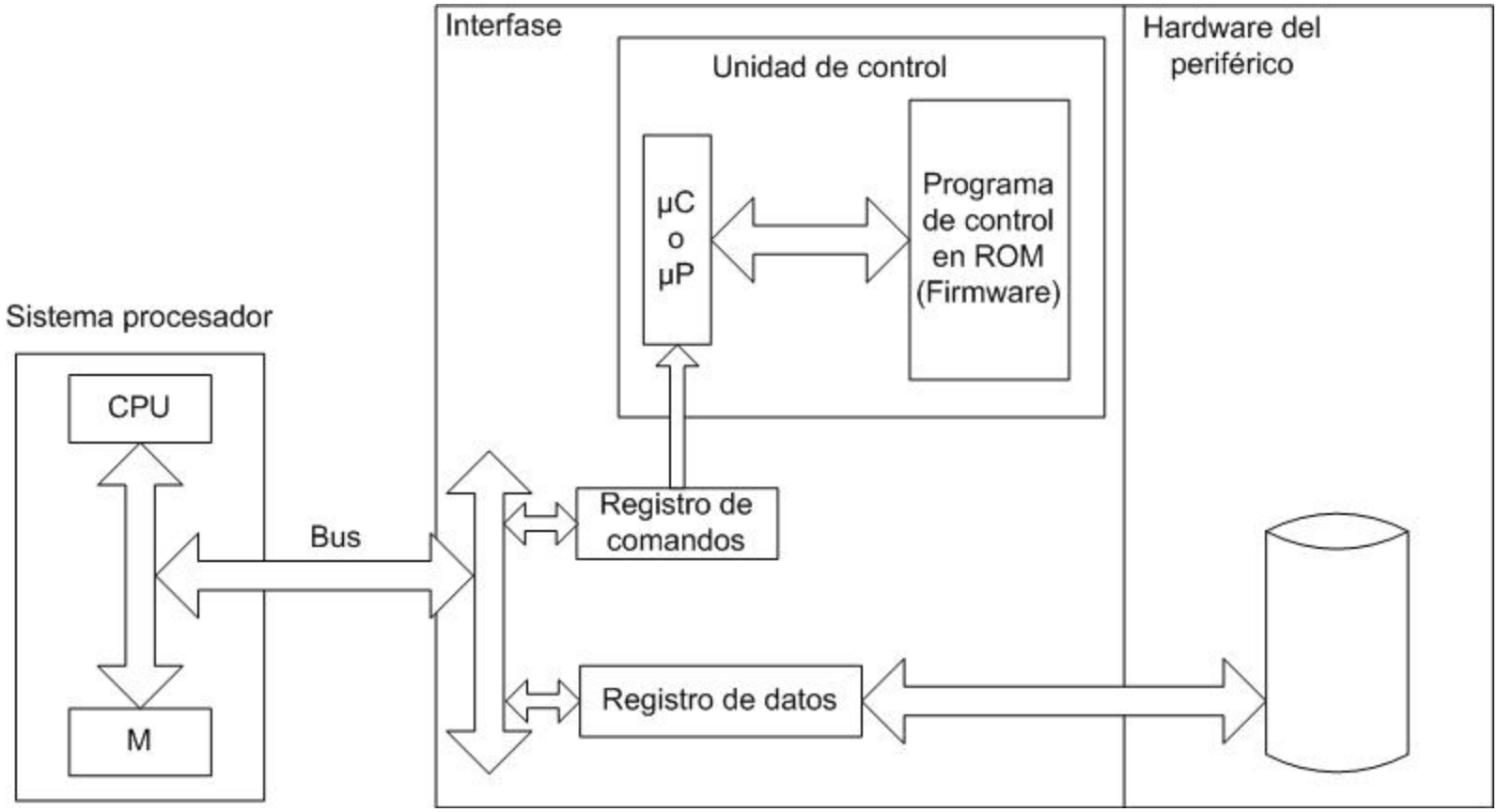
- Diferencia entre buses e interfaces no es obvia, y existe mucha confusión en el uso cotidiano de estos términos
  - Confusión aumenta porque un interfaz siempre necesita ir montado sobre un bus
  - Ejemplo: el término "*SCSI*" se usa rutinariamente para identificar:
    - un cierto bus (llamado el "bus paralelo SCSI")
    - o un interfaz, usado por muchos tipos distintos de dispositivos de almacenamiento, los cuales emplean buses muy variados (USB, SAS, FibreChannel, Firewire, etc...)

- Es imprescindible aprender a distinguir bien entre bus e interfaz
  - Diferencia fundamental entre ellos es que, en interfaz, el periférico tiene inteligencia suficiente para ejecutar autónomamente comandos, recibidos a través de un bus, y devolver por el bus el resultado de dicha ejecución

Bus

- Un bus es un subsistema que interconecta dos o más dispositivos a través de un conjunto de líneas comunes, y cuya definición está formada por:
  - La especificación de la señalización física y las líneas (incluyendo conectores) que unen a estos subsistemas
  - La especificación (funcional y de temporización) de los intercambios de datos por estas líneas, y de los mecanismos que monitorizan que se realicen adecuadamente
- Muy importante: en la definición de un bus no se hace ninguna interpretación de los datos transferidos por el bus
  - La temporización, o protocolo de intercambio de señales, nos permite distinguir en el bus entre señales de control y de datos, pero nunca se interpreta el contenido de esos datos

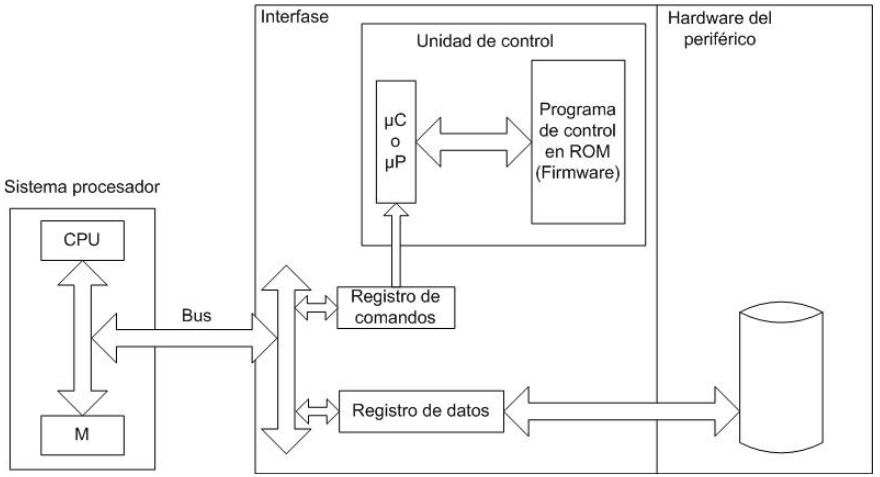
Interfaz


Interfase
Unidad de control
Hardware del periférico
µC o µP
Programa de control en ROM (Firmware)
Sistema procesador
CPU
Bus
Registro de comandos
M
Registro de datos

Interfaz

- Periférico que usa un interfaz dispone de la capacidad de procesar comandos de forma autónoma

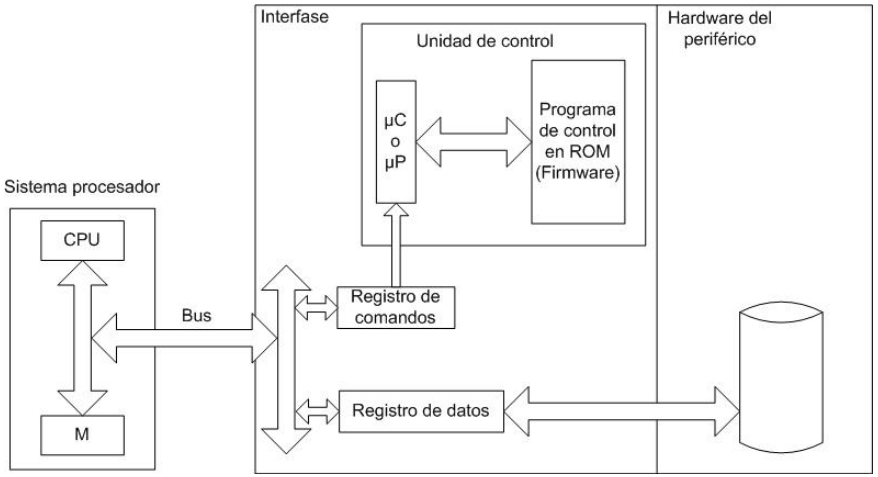
- Su interfase contiene una unidad de control compleja, con:
  - Un microprocesador simple o microntrolador avanzado
  - Un programa en ROM (*firmware*), con el código que ejecutará ese microprocesador para implementar los comandos



Interfaz

- La unidad de control recibe la petición de ejecutar un cierto comando a través del registro de comandos

- Esta petición será un código de operación (codop)
  - Cada operación tiene su propio codop
  - Codop debe ser escrito al registro de comandos desde el exterior a través de un bus



Interfaz

- En el caso más sencillo, operación a través de la interfaz se realiza en estos pasos:

1. Sistema procesador escribe en el registro de comandos, a través de un bus, el codop del comando a ejecutar
2. Unidad de control del periférico identifica el comando, y comienza su ejecución autónoma
3. Bajo supervisión de la unidad de control, el hardware del periférico realiza la operación (lectura, escritura o control). Los datos a mover son intercambiados con el sistema procesador a través del bus
4. Periférico termina la ejecución del comando devolviendo a través del bus códigos de estado y/o error

Interfaz

- Vemos, pues, que un interfaz hardware se caracteriza por:
  - Involucra un periférico "inteligente", es decir, con capacidad de procesamiento autónomo de comandos
  - Requiere interpretación de los datos transferidos a través del bus
    - En particular, comandos y códigos de estado serán campos de bits que deben ser decodificados para su identificación e interpretación
  - Siempre requiere un bus subyacente, a través del cual se intercambian códigos de control y datos entre sistema procesador y periférico
    - Esta presencia de un bus bajo el interfaz es la que induce a confusión entre ambos

Interfaz

- Por tanto, un interfaz es una implementación funcional en capas

- Interfaz = Bus + "inteligencia" = Bus + procesamiento autónomo de comandos

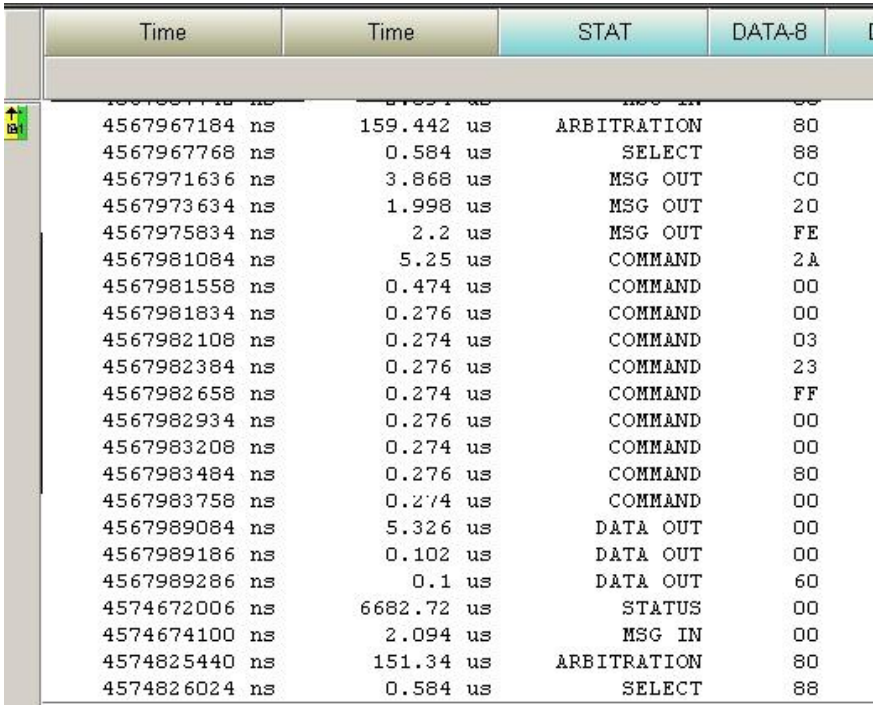
Interfaz

- Ejemplo: transacción de escritura en interfaz SCSI sobre bus paralelo SCSI

Time	Time	STAT	DATA-8
4567967184 ns	159.442 us	ARBITRATION	80
4567967768 ns	0.584 us	SELECT	88
4567971636 ns	3.868 us	MSG OUT	C0
4567973634 ns	1.998 us	MSG OUT	20
4567975834 ns	2.2 us	MSG OUT	FE
4567981084 ns	5.25 us	COMMAND	2A
4567981558 ns	0.474 us	COMMAND	00
4567981834 ns	0.276 us	COMMAND	00
4567982108 ns	0.274 us	COMMAND	03
4567982384 ns	0.276 us	COMMAND	23
4567982658 ns	0.274 us	COMMAND	FF
4567982934 ns	0.276 us	COMMAND	00
4567983208 ns	0.274 us	COMMAND	00
4567983484 ns	0.276 us	COMMAND	80
4567983758 ns	0.274 us	COMMAND	00
4567989084 ns	5.326 us	DATA OUT	00
4567989186 ns	0.102 us	DATA OUT	00
4567989286 ns	0.1 us	DATA OUT	60
4574672006 ns	6682.72 us	STATUS	00
4574674100 ns	2.094 us	MSG IN	00
4574825440 ns	151.34 us	ARBITRATION	80
4574826024 ns	0.584 us	SELECT	88

Interfaz

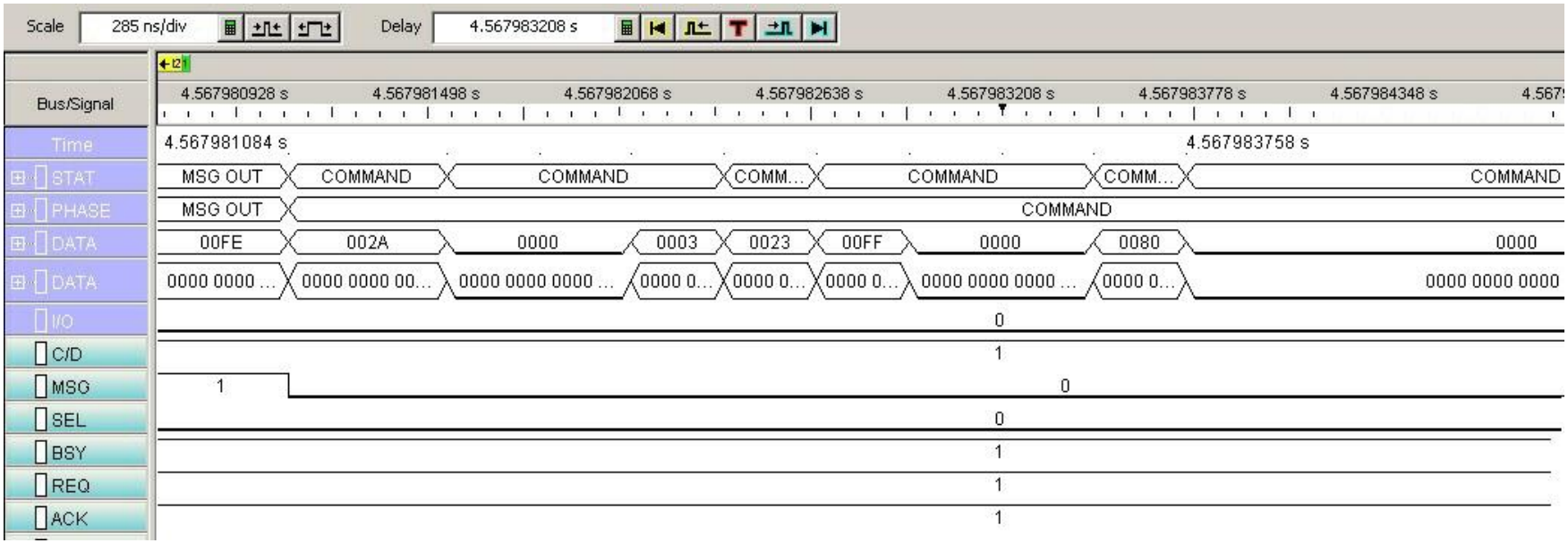
- Figura muestra listado de contenidos transferidos a través del bus para ordenar a un disco duro un comando de escritura
  - Necesaria la interpretación de los datos para identificar el comando, extraer parámetros y determinar si la ejecución ha terminado con éxito
  - Estandarización de interfaz SCSI permite determinar, sin ambigüedad ninguna, cómo interpretar en cada momento el contenido del bus



Time	Time	STAT	DATA-8
4567967184 ns	159.442 us	ARBITRATION	80
4567967768 ns	0.584 us	SELECT	88
4567971636 ns	3.868 us	MSG OUT	C0
4567973634 ns	1.998 us	MSG OUT	20
4567975834 ns	2.2 us	MSG OUT	FE
4567981084 ns	5.25 us	COMMAND	2A
4567981558 ns	0.474 us	COMMAND	00
4567981834 ns	0.276 us	COMMAND	00
4567982108 ns	0.274 us	COMMAND	03
4567982384 ns	0.276 us	COMMAND	23
4567982658 ns	0.274 us	COMMAND	FF
4567982934 ns	0.276 us	COMMAND	00
4567983208 ns	0.274 us	COMMAND	00
4567983484 ns	0.276 us	COMMAND	80
4567983758 ns	0.274 us	COMMAND	00
4567989084 ns	5.326 us	DATA OUT	00
4567989186 ns	0.102 us	DATA OUT	00
4567989286 ns	0.1 us	DATA OUT	60
4574672006 ns	6682.72 us	STATUS	00
4574674100 ns	2.094 us	MSG IN	00
4574825440 ns	151.34 us	ARBITRATION	80
4574826024 ns	0.584 us	SELECT	88

Interfaz

- Figura muestra los datos que componen el comando, y sus parámetros, siendo transferidos a través del bus paralelo SCSI
  - Se observa que el bus posee su propia señalización de control, realizada en paralelo al paso de los datos



Interfaz

- La misma secuencia de datos puede pasarse a través de otro bus (FibreChannel, Ethernet, SAS, USB), y el comando SCSI interpretado por el periférico seguirá siendo el mismo, independientemente del bus usado
  - El comando reside en el nivel del interfaz SCSI

Time	Time	STAT	DATA-8
4567967184 ns	159.442 us	ARBITRATION	80
4567967768 ns	0.584 us	SELECT	88
4567971636 ns	3.868 us	MSG OUT	C0
4567973634 ns	1.998 us	MSG OUT	20
4567975834 ns	2.2 us	MSG OUT	FE
4567981084 ns	5.25 us	COMMAND	2A
4567981558 ns	0.474 us	COMMAND	00
4567981834 ns	0.276 us	COMMAND	00
4567982108 ns	0.274 us	COMMAND	03
4567982384 ns	0.276 us	COMMAND	23
4567982658 ns	0.274 us	COMMAND	FF
4567982934 ns	0.276 us	COMMAND	00
4567983208 ns	0.274 us	COMMAND	00
4567983484 ns	0.276 us	COMMAND	80
4567983758 ns	0.274 us	COMMAND	00
4567989084 ns	5.326 us	DATA OUT	00
4567989186 ns	0.102 us	DATA OUT	00
4567989286 ns	0.1 us	DATA OUT	60
4574672006 ns	6682.72 us	STATUS	00
4574674100 ns	2.094 us	MSG IN	00
4574825440 ns	151.34 us	ARBITRATION	80
4574826024 ns	0.584 us	SELECT	88

Transacciones de bus

- Llamamos *transacción de bus* a la secuencia de operaciones que componen una transferencia de datos completa

- Dirección de la transacción usa el criterio "CPU-céntrico":
  - Transacción de lectura: transferencia de datos desde periférico hacia sistema procesador
  - Transacción de escritura: transferencia de datos desde sistema procesador hacia periférico

Transacciones de bus

- Transacciones de bus siempre compuestas por dos tipos de fases:
  - Una fase de direccionamiento (*Dir*): En ella el dispositivo que inicia la transacción coloca en las líneas del bus valores para identificar al dispositivo con quien quiere intercambiar datos y el tipo de transacción a realizar
  - Una o varias fases de datos (*D*): En ellas se realiza la transferencia de los contenidos útiles
    - Dependiendo del tipo de transacción, el dispositivo que la inició actúa en estas fases como origen (escritura) o destino (lectura) de los datos transferidos

Maestro y esclavo de bus

- Una transacción de bus siempre involucra a un maestro de bus y a un esclavo de bus

- Maestro de bus (Bus master): Dispositivo con capacidad de iniciar una transacción de bus
  - Direcciona en el bus a un dispositivo, indicándole que actúe como esclavo de bus
  - Adicionalmente, le señaliza qué tipo de transacción se va a realizar

- Esclavo de bus (Bus slave): Dispositivo que carece de la capacidad de iniciar transacciones de bus
  - Espera pasivamente a ser direccionado por un maestro de bus al inicio de una transacción de bus

Maestro y esclavo de bus

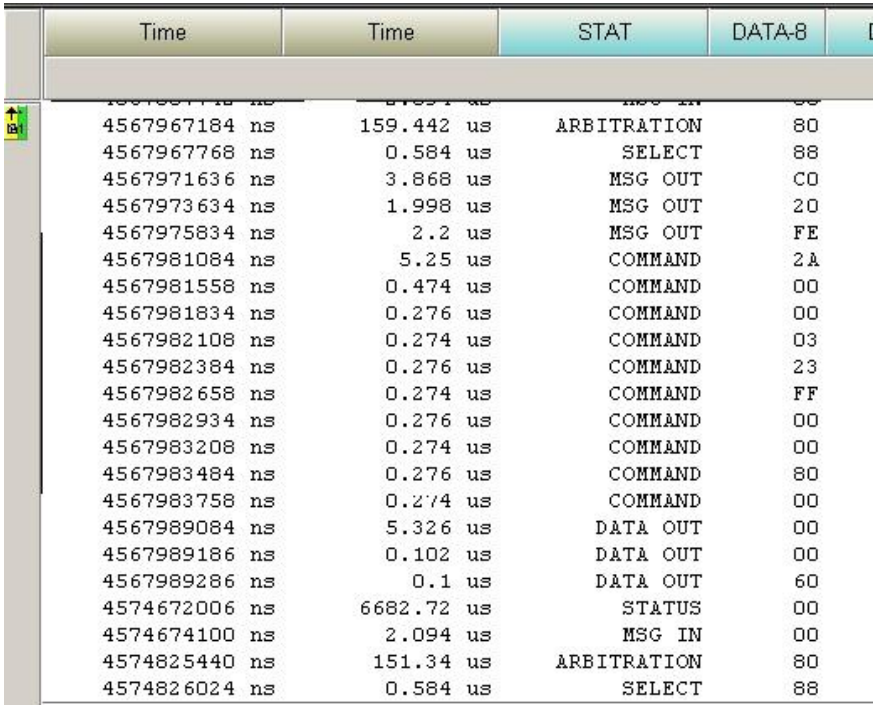
- Durante una transacción de bus sólo puede existir un único maestro de bus activo
  - Un mecanismo de arbitración determinará qué maestro de bus estará activo en cada transacción
- Ejemplos típicos de maestros de bus son los controladores DMA y la CPU

Relación entre transacción de bus e interfaz

- Las transacciones de bus son realizadas por la capa encargada del transporte de los datos, no de su interpretación
  - Durante la fase de direccionamiento de la transacción de bus **NO** se transfiere el comando a nivel de interfaz
  - Los códigos de operación y parámetros de un comando a nivel de interfaz se envían en las fases de datos de una única transacción de bus

Relación entre transacción de bus e interfaz

- En transacción de bus de figura, fase direccionamiento está compuesta por los estados de bus "ARBITRATION" y "SELECT"
- Resto de estados del bus corresponden a fases de datos de la transacción de bus
  - A nivel de interfaz, esos datos se van interpretando como códigos de estado, códigos de operación, parámetros y datos.
  - El estándar define cómo interpretar cada byte de acuerdo con su posición en la secuencia temporal



Time	Time	STAT	DATA-8
4567967184 ns	159.442 us	ARBITRATION	80
4567967768 ns	0.584 us	SELECT	88
4567971636 ns	3.868 us	MSG OUT	C0
4567973634 ns	1.998 us	MSG OUT	20
4567975834 ns	2.2 us	MSG OUT	FE
4567981084 ns	5.25 us	COMMAND	2A
4567981558 ns	0.474 us	COMMAND	00
4567981834 ns	0.276 us	COMMAND	00
4567982108 ns	0.274 us	COMMAND	03
4567982384 ns	0.276 us	COMMAND	23
4567982658 ns	0.274 us	COMMAND	FF
4567982934 ns	0.276 us	COMMAND	00
4567983208 ns	0.274 us	COMMAND	00
4567983484 ns	0.276 us	COMMAND	80
4567983758 ns	0.274 us	COMMAND	00
4567989084 ns	5.326 us	DATA OUT	00
4567989186 ns	0.102 us	DATA OUT	00
4567989286 ns	0.1 us	DATA OUT	60
4574672006 ns	6682.72 us	STATUS	00
4574674100 ns	2.094 us	MSG IN	00
4574825440 ns	151.34 us	ARBITRATION	80
4574826024 ns	0.584 us	SELECT	88

Iniciador y diana

- Existencia simultánea de transacciones de bus y de comandos a nivel de interfaz hace necesario identificar quien inicia la transacción de bus y quien inicia el comando
  - Aparecen dos nuevos conceptos: iniciador y diana

- Iniciador: Es un dispositivo con capacidad de iniciar un comando a nivel de interfaz

- Diana: Es un dispositivo que carece de la capacidad de iniciar autónomamente un comando a nivel de interfaz
  - Sólo puede esperar pasivamente a que un iniciador le ordene (a través de una transacción de bus) un comando, ejecutarlo y devolver el resultado

Iniciador y diana

- No tiene que existir una correlación directa entre "Iniciador" y "Maestro de bus", ni entre "Diana" y "Esclavo de bus"
  - Un iniciador necesariamente tendrá que actuar como maestro de bus cuando vaya a generar un comando a nivel de interfaz
  - Sin embargo, es frecuente que, durante la ejecución del comando, una diana pueda adoptar el papel de maestro de bus o de esclavo de bus (ej: bus SCSI)
    - Cuando la diana actúa como maestro de bus, el iniciador actuará como esclavo de bus
  - Independientemente de su papel a nivel de bus, a nivel de interfaz la diana es siempre pasiva

Iniciador y diana

NOTA IMPORTANTE: Desgraciadamente, el uso de los términos iniciador y diana no es coherente en los distintos buses e interfaces, como muestra el siguiente ejemplo:

- En el bus PCI se denomina "iniciador" al maestro de bus, y "diana" al esclavo de bus

- Por ello, cuando PCI-X introdujo lo que conceptualmente funciona como comandos a nivel de interfaz se necesitaron términos nuevos, en la forma:
  - *Requester* = dispositivo que inicia comandos = equivalente a un iniciador a nivel de interfaz
  - *Completer* = dispositivo que ejecuta comandos = equivalente a diana a nivel de interfaz

Transferencia en modo ráfaga

- La transacción de bus más sencilla está compuesta por una fase de direccionamiento, seguida por una única fase de transferencia de datos
  - Cuando sólo se pueda usar este tipo de transacción de bus, el intercambio de un bloque de datos con un periférico tendrá la forma de la figura, con todos los direccionamientos involucrando al mismo dispositivo periférico:

  - Es fácil darse cuenta que este modo de operación es el que aparece cuando se usa PIO, o controladores DMA primitivos
- Transacciones monofase de datos usan bus ineficientemente
  - Las fases de direccionamiento son redundantes, y no transportan carga útil
  - Suele haber tiempos muertos entre transacciones consecutivas

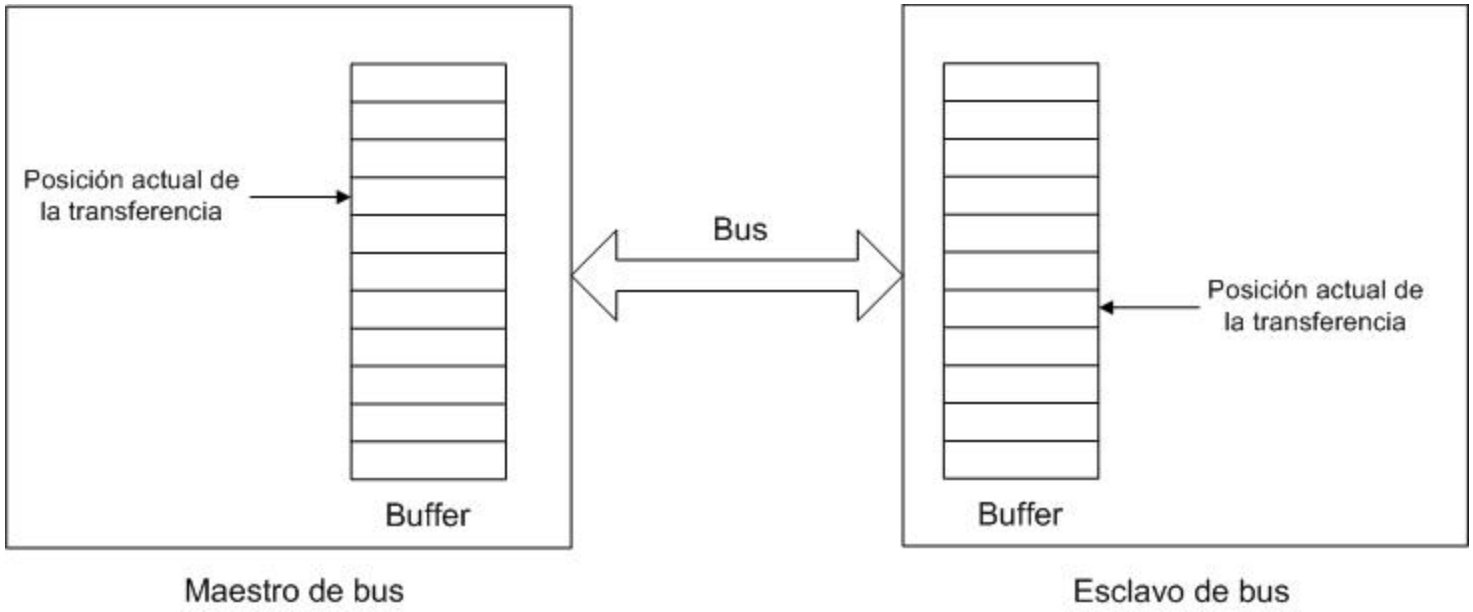
Transferencia en modo ráfaga

- Para mejorar drásticamente el rendimiento, todos los buses modernos trabajan con transferencias en *modo ráfaga* (*burst mode*)

- En transferencia en modo ráfaga, transacción de bus consta de una única fase de direccionamiento, seguida por múltiples fases de datos consecutivas

  - Bus se usa de forma mucho más eficiente, al desaparecer las fases adicionales de direccionamiento, así como los tiempos muertos intermedios

Transferencia en modo ráfaga

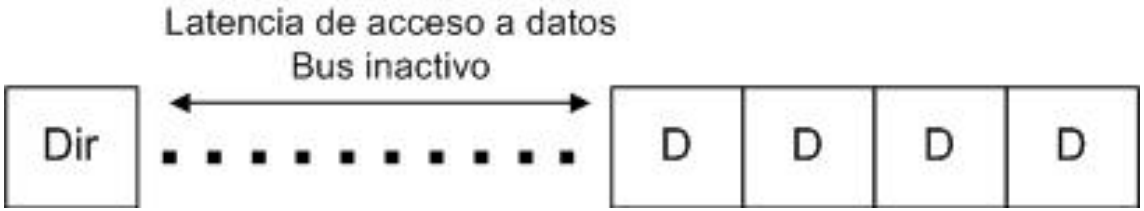


- Transferencia en modo ráfaga se suele realizar entre buffers de memoria, ubicados en interfases de maestro y esclavo de bus
  - Modo ráfaga requiere que ambos dispositivos mantengan un puntero a la posición del buffer involucrada en el movimiento de datos, y la autoincrementen tras cada transferencia de datos
  - Modo ráfaga es, por tanto, esencialmente una variante de DMA

Transacciones de bus bloqueantes: eficiencia

- En las implementaciones sencillas de los buses, las transacciones de bus son bloqueantes
  - Sólo puede existir en el sistema una única transacción de bus en curso
  - Una vez comenzada una transacción de bus, el bus queda bloqueado
    - No se liberará el bus hasta que termine la última fase de transferencia de datos de la transacción en curso
    - Con el bus bloqueado no es posible iniciar ninguna nueva transacción de bus, aun cuando la transacción actual tenga el bus inactivo un largo tiempo por falta de datos
- La implementación bloqueante de un bus es sencilla, pero presenta problemas de eficiencia, especialmente en combinación con transferencias en modo ráfaga

Transacciones de bus bloqueantes: eficiencia

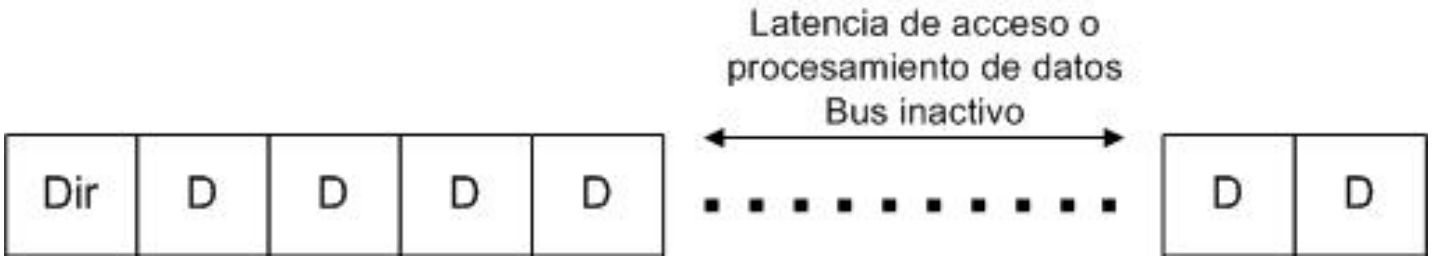


1) Latencias de acceso inicial a datos

- Este problema aparece cuando, tras la fase inicial de datos, el orígen de datos tarda mucho tiempo en acceder físicamente a los datos a transferir y prepararlos en el buffer
  - Ejemplo: acceso de lectura a un disco duro que tenga el motor parado por ahorro de energía

- Durante el tiempo de espera el bus está inactivo y desaprovechado
  - Dependiendo del tipo de dispositivo, este tiempo puede ser largo

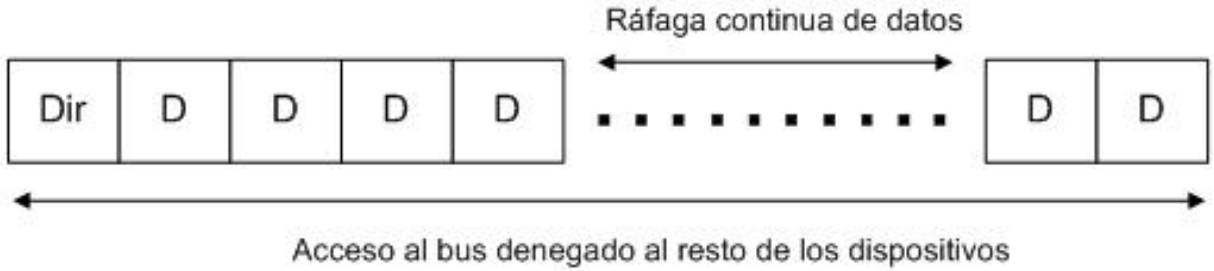
Transacciones de bus bloqueantes: eficiencia



2) Latencias de procesamiento o acceso intermedio a datos

- Ocurre si, durante la transacción, origen se queda sin datos en buffer, y necesita obtener físicamente más datos
  - Ej: lectura agota datos en caché de disco duro, y necesita leer físicamente un nuevo sector de la media
- También si, durante la transacción, destino necesita detener transferencias para procesar los datos ya almacenados en su buffer
  - Escritura a disco ha llenado su buffer, y necesita vaciarlo escribiendo físicamente datos a la superficie del disco
- En ambos casos aparece un tiempo muerto de espera, durante el cual se desaprovecha el bus
  - Dependiendo del tipo de dispositivo, este tiempo puede ser largo

Transacciones de bus bloqueantes: eficiencia



3) Monopolización del bus

- Problema específico de transacciones bloqueantes con transferencias en modo ráfaga

- Una vez iniciada transacción, ningún maestro de bus puede comenzar otra transacción hasta que se transfiera el último dato de la ráfaga actual
  - Si ráfaga es larga (o hay latencias en la transacción), tiempo de denegación de accesos al bus para los demás dispositivos puede ser muy grande
  - Problema serio si hay dispositivos que requieran atención inmediata a su petición de E/S (ej: tarjeta de red)
  - Reduce también la eficiencia del sistema, al penalizar la operación concurrente de sus dispositivos

Transacciones divididas (“split transactions”)

- La solución a los problemas de eficiencia descritos es usar transacciones no bloqueantes

- Diremos que un bus soporta transacciones divididas (o interrumpibles) si la transferencia de un bloque de datos puede ser:

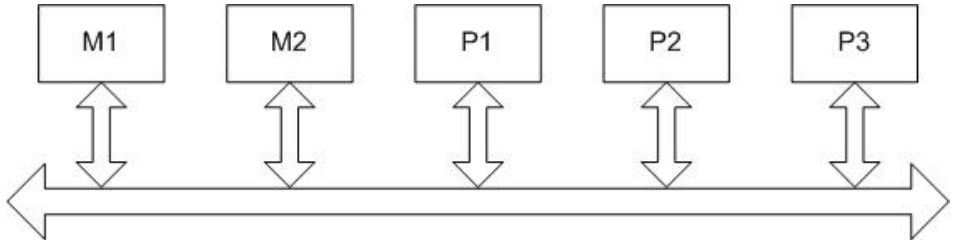
  - interrumpida para intercalar una o más nuevas transacciones de bus

  - y luego continuada desde donde se dejó, hasta terminar la transferencia de los datos

Transacciones divididas (“split transactions”)

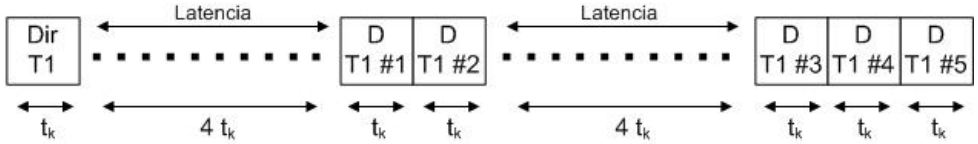
- Características de las transacciones divididas:
  - La interrupción de la transferencia puede producirse entre la fase de direccionamiento y la primera fase de datos, o entre dos fases de datos consecutivas
  - Necesita que tanto maestro como esclavo de bus mantenga información de estado de la transacción en el momento de la interrupción, incluyendo la dirección del próximo ítem de datos a transferir
  - La reanudación de la transacción requiere una nueva fase de direccionamiento, en la que maestro y esclavo de bus reconozcan la continuación de la transacción interrumpida
- Las transacciones divididas son especialmente interesantes a nivel de interfaz
  - Permiten intercalar la ejecución de múltiples comandos
  - Los conceptos básicos a nivel de interfaz son idénticos a los ya descritos, pero cambiando “maestro de bus” y “esclavo de bus” por “iniciador” y “diana”, respectivamente

Transacciones divididas (“split transactions”)

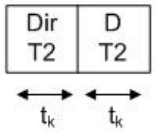


Ejemplo: Suponemos un bus con dos maestros de bus (M1, M2) y tres esclavos de bus (P1, P2, P3), en el que se necesitan realizar estas tres transacciones de bus:

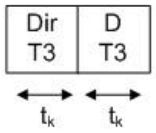
T1: M1 a P1, 5 fases de datos con las siguientes latencias y duraciones



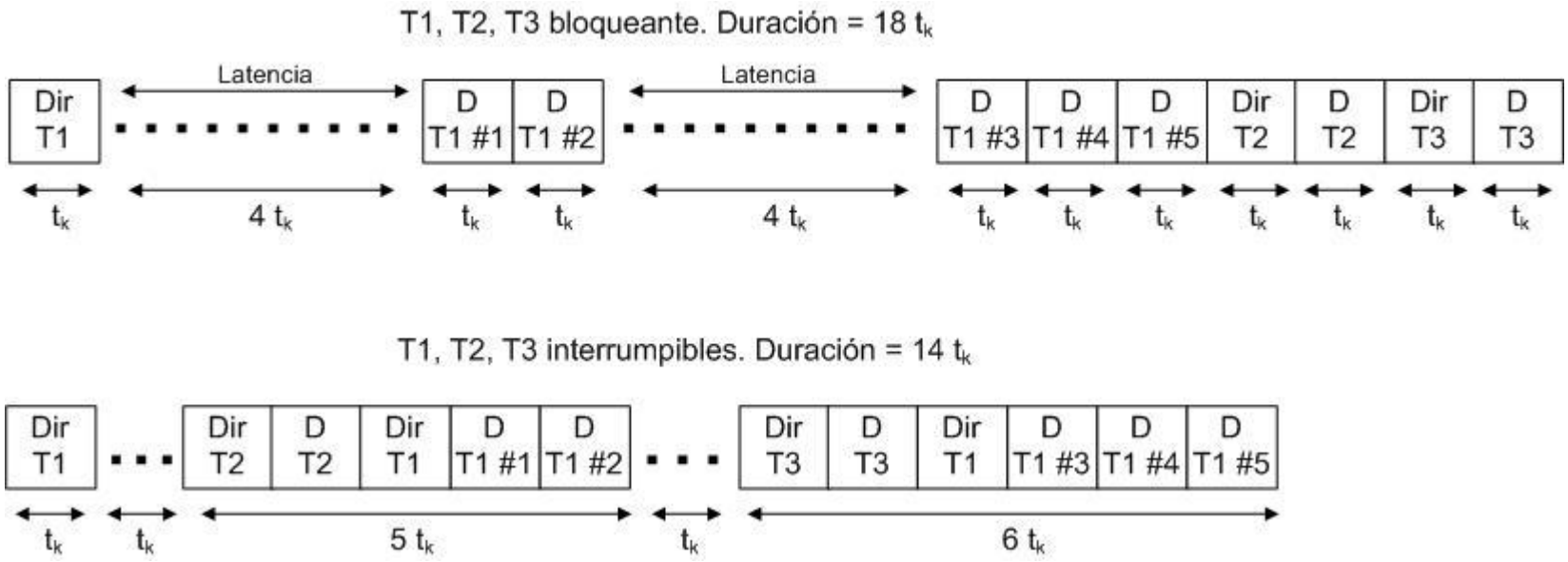
T2: M2 a P2, 1 fase de datos



T3: M2 a P3, 1 fase de datos



Transacciones divididas ("split transactions")



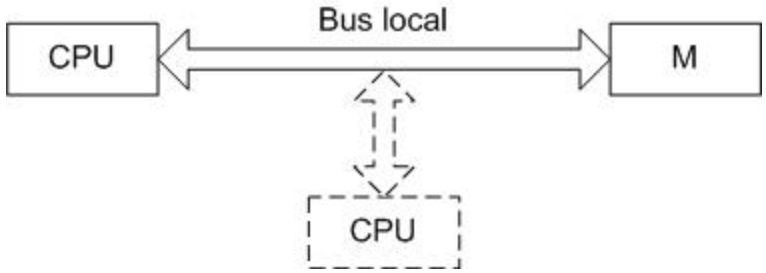
- La figura muestra los cronogramas con transacciones bloqueantes y con transacciones divididas
- Con transacciones bloqueantes la duración es menor, así como el tiempo que han tenido que esperar M2, P2 y P3 para realizar sus operaciones de E/S

Transacciones divididas (“split transactions”)

- Ventajas:
  - Permiten aprovechar las latencias, evitando inactividad del bus
  - Evitan monopolización del bus, fragmentando las transferencias largas
  - Mejoran la concurrencia del sistema y reducen los tiempos de respuesta

- Inconvenientes:
  - El diseño del sistema es más complejo
  - Requiere unidades de control relativamente complejas en maestro y esclavo de bus
    - Esto no es problema en periféricos que emplean un interfaz hardware, porque en ellos ya tenemos, de todas formas, una unidad de control compleja

Clasificación de los buses según su propósito



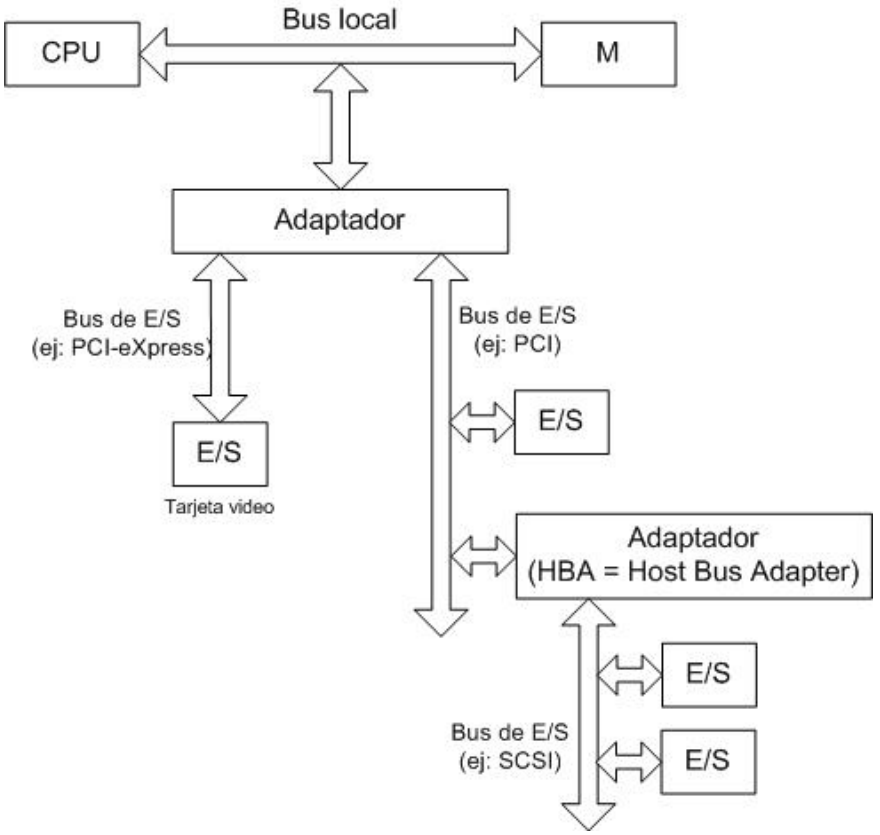
Bus CPU – Memoria (Bus local)

- Diseñado para interconectar, con el máximo rendimiento posible, CPU y memoria principal
  - Buses de alta velocidad, trabajando a la máxima frecuencia posible
  - Necesariamente sus líneas deben ser muy cortas
  - Diseño muy optimizado y ligado al tipo de CPU y subsistema de memoria
  - Puede contemplar el uso de múltiples CPUs
- En diseños actuales, este bus está desacoplado del resto del sistema por el "Puente Norte" del chipset

Buses de E/S, o de expansión

- Su propósito es la conexión de dispositivos de E/S, o de adaptadores para estos dispositivos

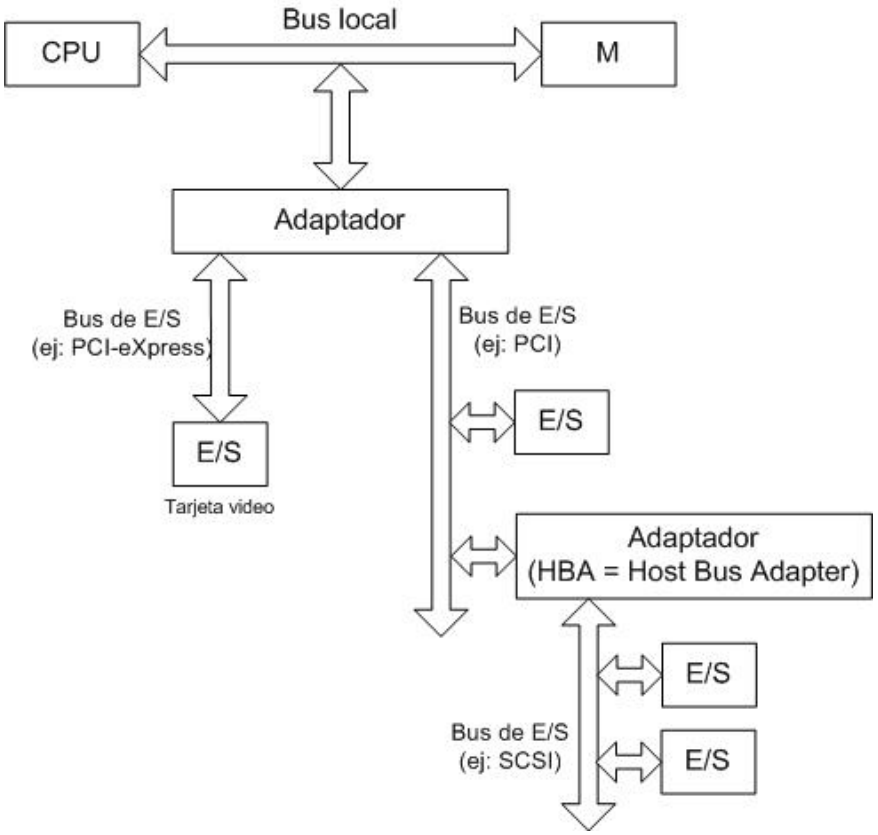
- No tienen conexión directa al bus local, requiriendo siempre un circuito intermedio de adaptación



Buses de E/S, o de expansión

Tienen características variadas:

- Los hay rápidos, optimizados para máxima velocidad
  - Por tanto, cortos y con sólo un dispositivo conectado (ej: AGP, PCI-eXpress)
- Otros permiten la conexión de múltiples dispositivos, con longitudes moderadas y velocidades media o media-alta
  - Ej: PCI, SCSI, USB
- Los hay también optimizados para largas distancias, a velocidades bajas
  - Ej: bus serie RS-422



Clasificación de buses según su temporización

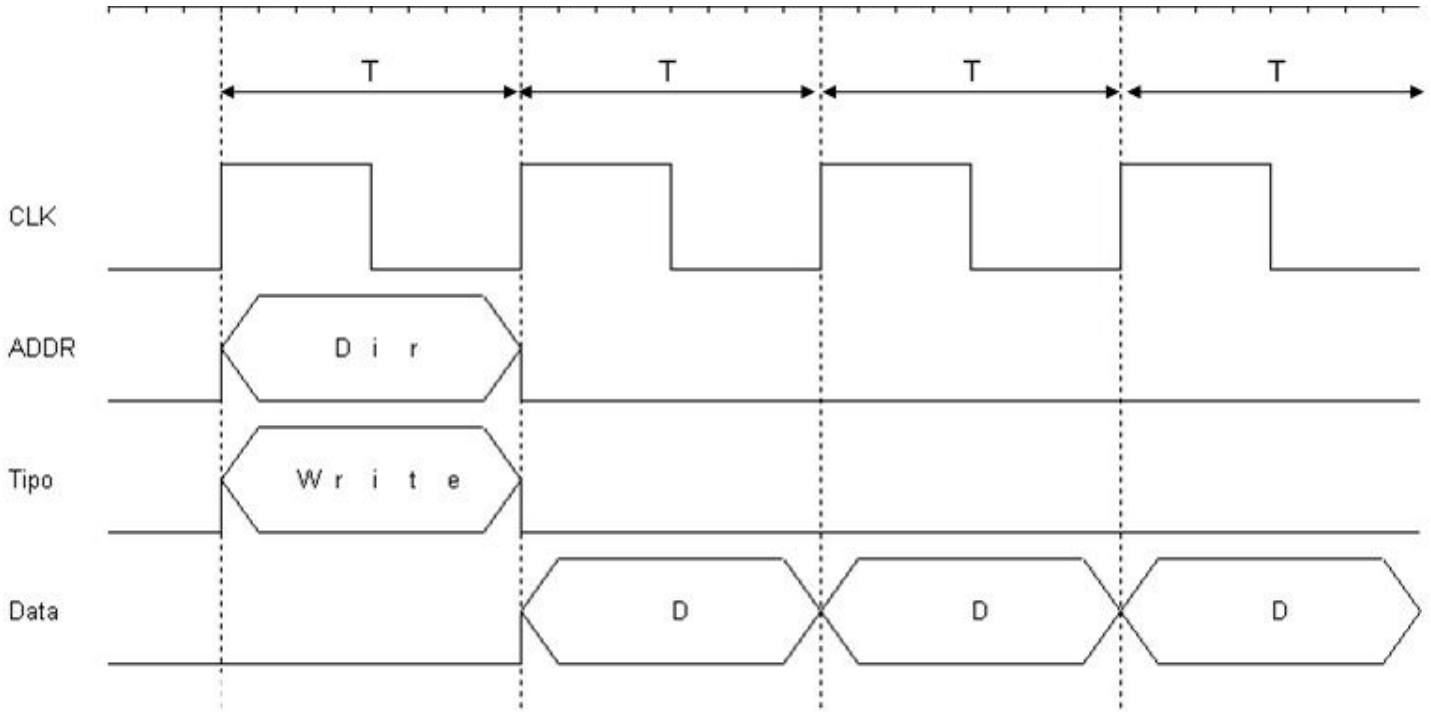
- En una transacción de bus es imprescindible el uso de señales de sincronismo comunes al maestro y al esclavo de bus, para que ambos sepan con exactitud:
  - La fase de la transacción en que se encuentran
  - El momento exacto en que se pueden leer e interpretar con seguridad los valores presentes en las líneas del bus
- Dependiendo del origen e interpretación de las señales de sincronismo, los buses se clasifican como síncronos o asíncronos

Bus síncrono

- En bus síncrono existe una señal periódica de sincronismo (típicamente llamada CLK) común a todos los dispositivos del bus
  - CLK es generada por un dispositivo externo, distinto del maestro o esclavo de bus
  - Los ciclos de la señal CLK delimitan con precisión intervalos de tiempo (llamados *ciclos de reloj*, o *time-slots*)

- Cada fase de una transacción de bus sólo puede producirse en un cierto time-slot
  - Dicho time-slot es fijado con precisión por las especificaciones del bus
  - Por tanto, el instante en que comienza o termina ese time-slot puede ser determinado con precisión tanto por el maestro como por el esclavo de bus, simplemente contando el número de ciclos CLK transcurridos desde el inicio de la transacción

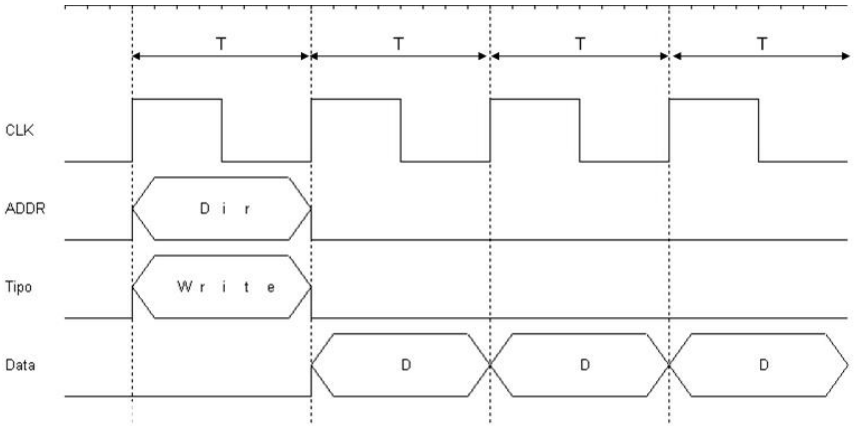
Bus síncrono



- Figura muestra ejemplo de transacción de escritura en bus síncrono, con una fase de direccionamiento y tres de datos
- Existe una señal de reloj CLK, cuyos ciclos marcan la evolución de la transacción

Bus síncrono

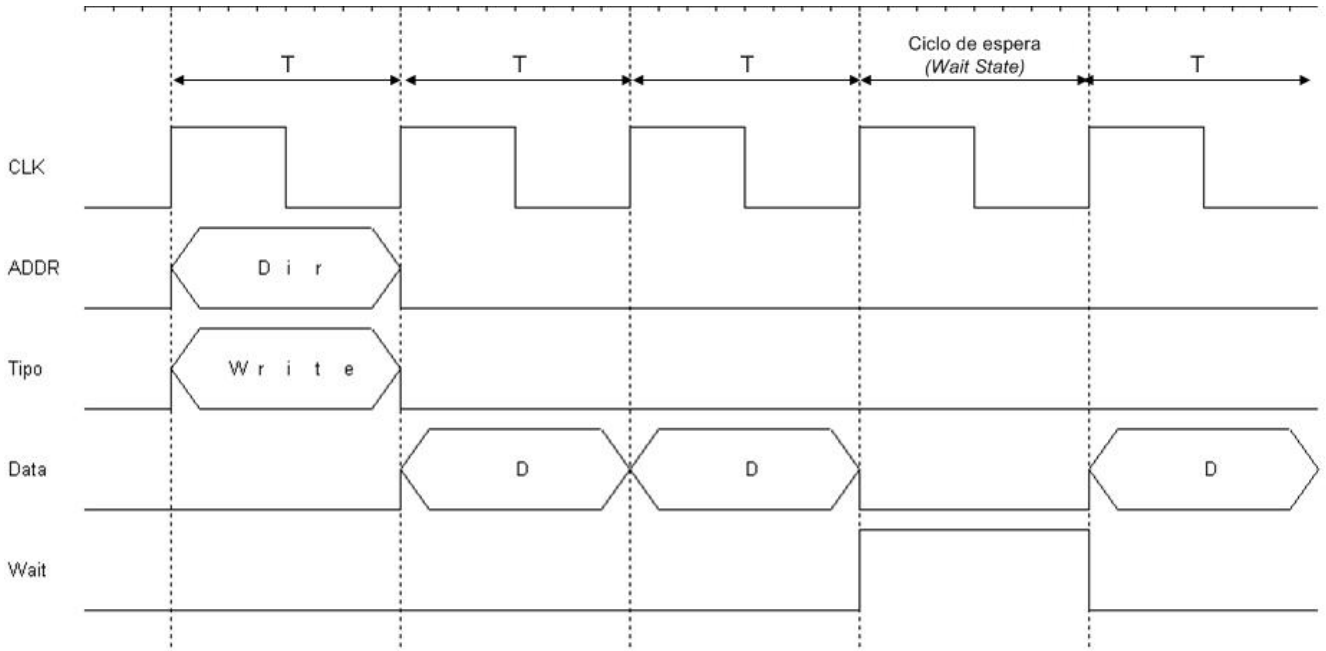
- Fase direccionamiento ocurre obligatoriamente en primer ciclo CLK, en el que:
  - Maestro bus debe obligatoriamente señalizar dirección y tipo de transacción
  - Esclavo bus debe obligatoriamente interpretar esta señalización
- Fases de datos se producen en segundo y subsiguientes ciclos CLK
  - Cada movimiento de datos requiere un CLK
  - En cada CLK, maestro está obligado a colocar en bus los datos a transferir, y esclavo a leerlos y almacenarlos



Bus síncrono

- La temporización de las transacciones en un bus síncrono es muy estrícta, y debe respetarse escrupulosamente
  - Si maestro o esclavo necesitan tiempo adicional para completar alguna fase de la transacción, deben señalizarlo explícitamente
    - Señalización consiste en la activación de una línea específica del bus, llamada típicamente "**WAIT"** o "**READY**"
  - Tiempo adicional se consigue mediante la inserción de ciclos de espera (WAIT STATES)
    - En un ciclo de espera CLK sigue oscilando, pero sus cambios dejan de contar a efectos de la evolución de la transacción

Bus síncrono



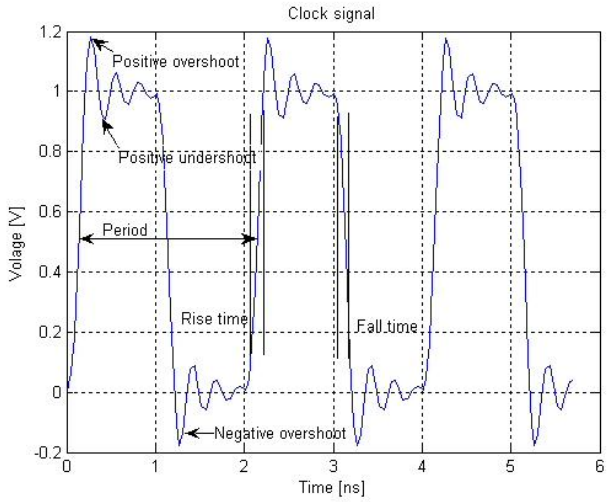
- Figura muestra la misma transacción de escritura, pero con un ciclo de espera
  - Esclavo necesita la inserción de un estado de espera entre segunda y tercera escritura, para poder procesar los dos primeros ítem de datos
  - Activación de WAIT permite que un ciclo completo de CLK pase sin contar a efectos del control de la transacción

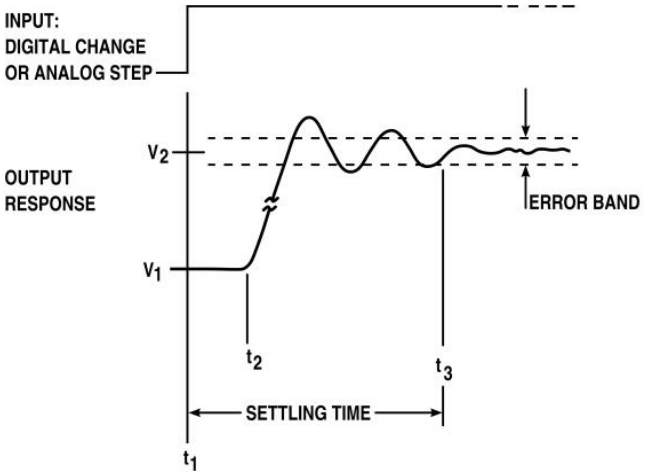
Bus síncrono

- Ventajas del bus síncrono:
  - Su implementación en los maestros y esclavos de bus es relativamente sencilla, al no requerir señalización bidireccional
  - En cada transferencia de datos requiere sólo señalización unidireccional, por lo que permite transferencias en modo ráfaga a la máxima velocidad posible

Bus síncrono

- Inconvenientes:
  - Para que su eficacia sea máxima, es necesario que todos los dispositivos conectados al bus sean capaces de trabajar a velocidades similares
    - Mezcla de dispositivos rápidos y lentos penaliza el rendimiento del bus, por frecuente inserción de estados de espera
  - Requiere distribuir por todo el bus una señal de reloj común
    - Necesita diseño cuidadoso (y costoso) de placa madre para evitar deformación de señal CLK en forma de rebotes ("ringing") y alargamiento de los flancos ("clock skew")

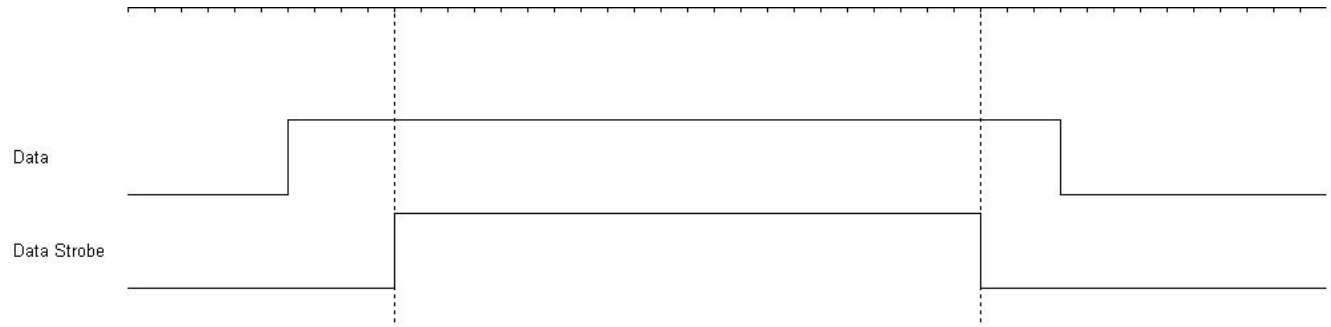




Bus asíncrono

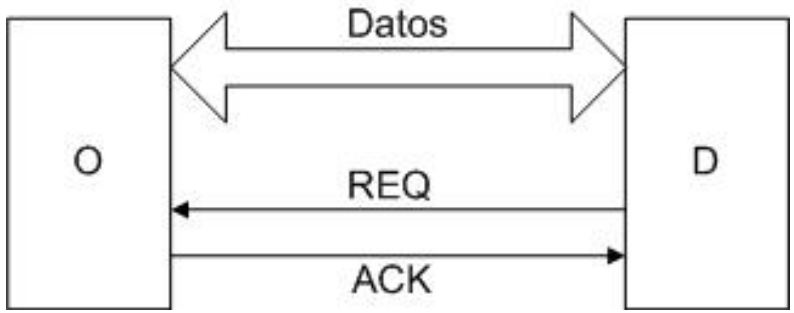
- En este bus la sincronización se consigue mediante el intercambio de señales de control (“handshaking”) centre el maestro y el esclavo de bus
  - No se necesita, por tanto, una señal global CLK para controlar la temporización del bus
- Señalización puede ser unidireccional (“one-way”) o bidireccional (“two-way”)

Bus asíncrono



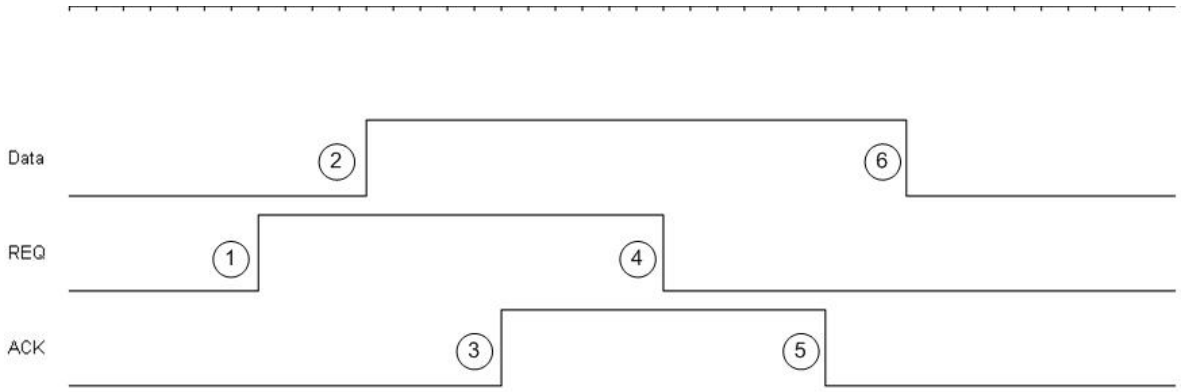
- Figura muestra posible implementación de señalización unidireccional
  - Usa un strobe de datos para delimitar intervalo de tiempo durante el que los valores en el bus de datos son estables
    - Típicamente, datos se leen en flanco subida o flanco bajada del strobe
  - Datos y strobe son generados por el sistema origen de los datos
  - Datos deben escribirse al bus antes de activar el strobe, y retirarse del bus después de haber desactivado el strobe

Bus asíncrono



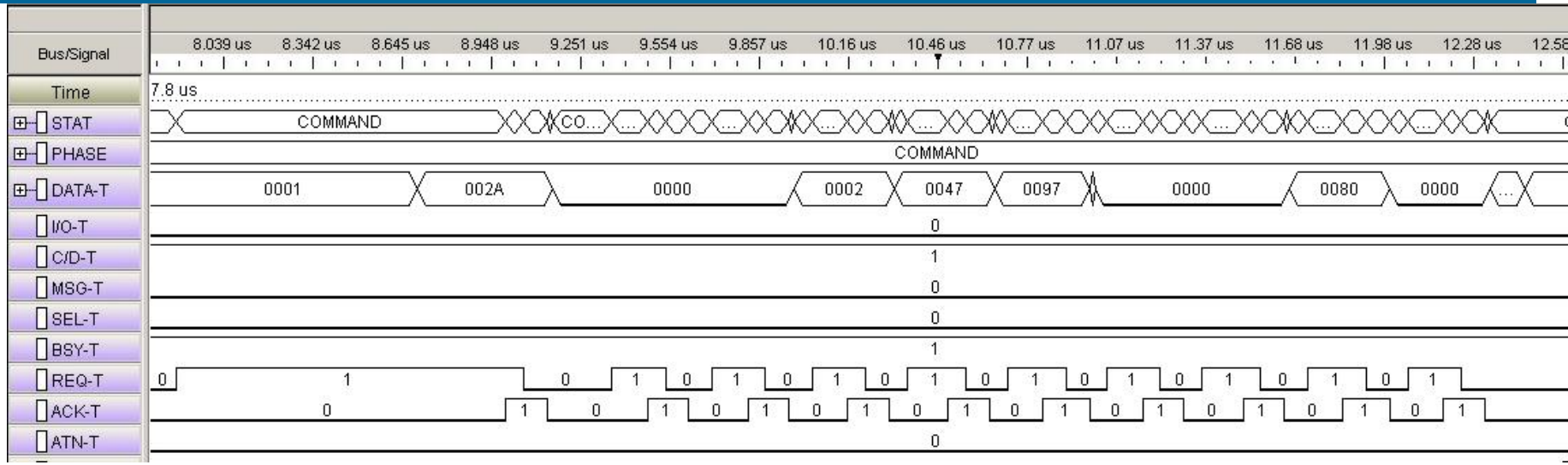
- Figura muestra implementación típica de señalización bidireccional
  - Sistema D = destino de datos
    - Utiliza señal REQ para solicitar que le transfieran datos
  - Sistema O = origen de datos
    - Utiliza señal ACK para indicar cuando hay datos válidos en el bus

Bus asíncrono



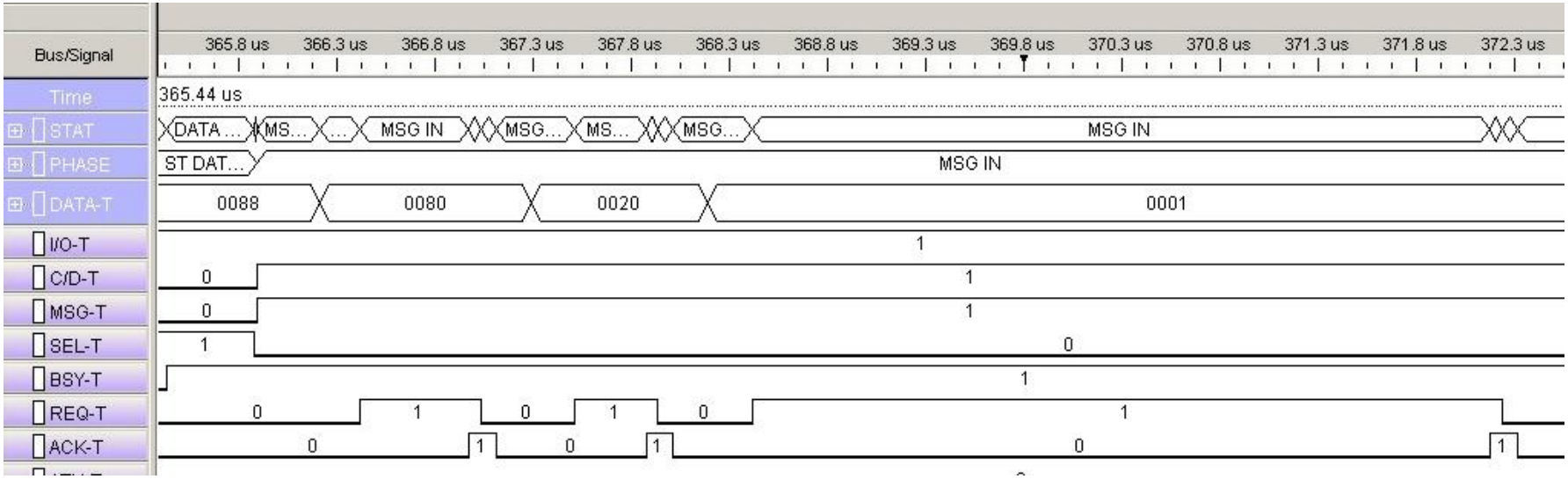
- Cronograma muestra secuencia de eventos durante transferencia
  1. D activa REQ solicitando se le envíe un ítem de datos
  2. O escribe los datos en el bus
  3. Una vez los valores del bus son estables, O activa ACK para indicar presencia en el bus de datos válidos. D lee los datos en flanco subida ACK
  4. Tras leer los datos, D desactiva REQ, indicando lectura finalizada
  5. O desactiva ACK en respuesta a la desactivación de REQ
  6. O retira los datos del bus

Bus asíncrono



- Figura muestra ejemplo real de handshaking asíncrono bidireccional en el bus paralelo SCSI
  - Tarjeta SCSI (= iniciador = O) está enviando comando del interfaz SCSI a disco duro (= diana = D)
  - Disco solicita con REQ el envío de un byte de comando
  - Iniciador confirma con ACK la escritura del byte de comando

Bus asíncrono



- En este segundo ejemplo se han invertido los papeles de orígen y destino de datos
  - Disco duro (= diana = **O**) está enviando al iniciador códigos de control durante la ejecución de un comando
  - Tarjeta SCSI (= iniciador = **D**) recibe esos códigos de estado
  - También se han invertido funciones de las señales REQ y ACK
    - Disco indica con REQ la escritura al bus de un byte de código de control
    - Iniciador indica con ACK lectura del byte de código de control

Bus asíncrono

- Ventajas:
  - Proporciona mayor flexibilidad para sincronizar la operación de dispositivos con distintas velocidades de operación
  - El estado de la transacción no evoluciona hasta que se genera explícitamente la señal de control
    - No requiere, por tanto, la inserción de estados de espera
- Inconvenientes:
  - Es más complejo de implementar
  - En señalizaciones bidireccionales, la necesidad de esperar a la señal ACK penaliza y reduce la capacidad del bus