# Organización del computador

## Buses y protocolos

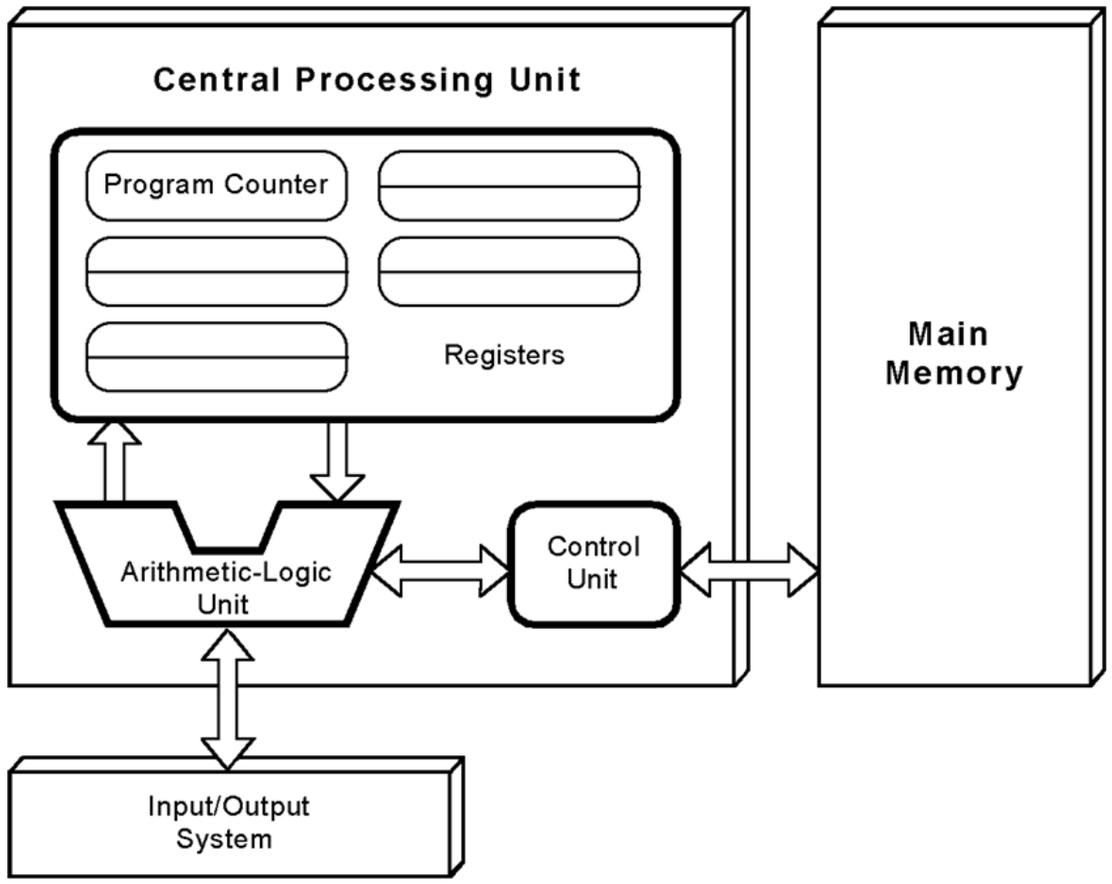
# Jerarquía de máquina

| | | |
|---|---|---|
| Nivel 6 | Usuario | Programa ejecutables |
| Nivel 5 | Lenguaje de alto nivel | C++, Java, Python, etc. |
| Nivel 4 | Lenguaje ensamblador | Assembly code |
| Nivel 3 | Software del sistema | Sistema operativo, bibliotecas, etc. |
| Nivel 2 | Lenguaje de máquina | Instruction Set Architecture (ISA) |
| Nivel 1 | Unidad de control | Microcódigo / hardware |
| Nivel 0 | Lógica digital | Circuitos, compuertas, memorias |

- Cada nivel funciona como una máquina abstracta que oculta la capa anterior
- Cada nivel es capaz de resolver determinado tipo de problemas a partir de comprender un tipo de instrucciones específico
- La capa inferior es utilizada como servicio

# Arquitectura de von Newmann

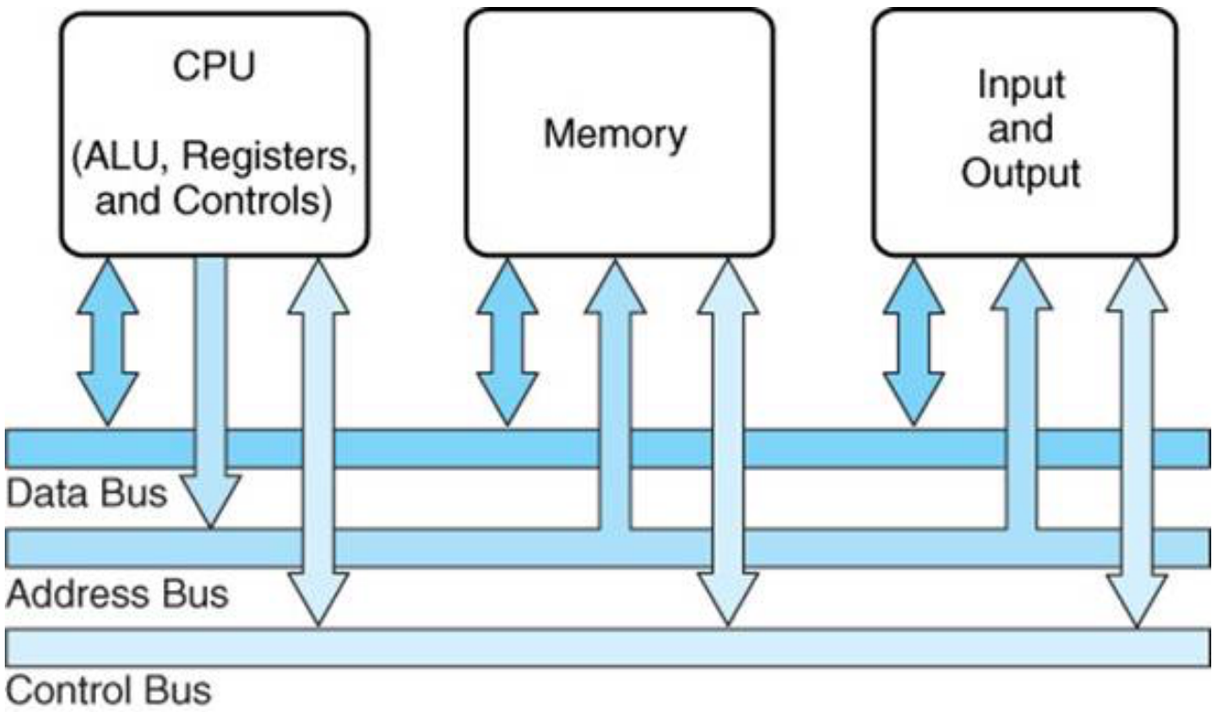
- 3 componentes principales:
  - **CPU**: Unidad de control, Unidad Aritmética lógica, Registros
  - **Memoria**: Almacenamiento de programas y datos
  - **Sistema** de Entrada y Salida
- Procesamiento secuencial de instrucciones
- Datos almacenados en sistema binario
- Sistema de interconexión de componentes:
  - Conecta la Unidad de Control con la Memoria mediante un camino único
  - La unicidad del camino fuerza la alternación entre ciclos de lectura / escritura y ejecución
  - Esta alternación se llama cuello de botella de von Newmann (von Newmann bottleneck)[*]




---

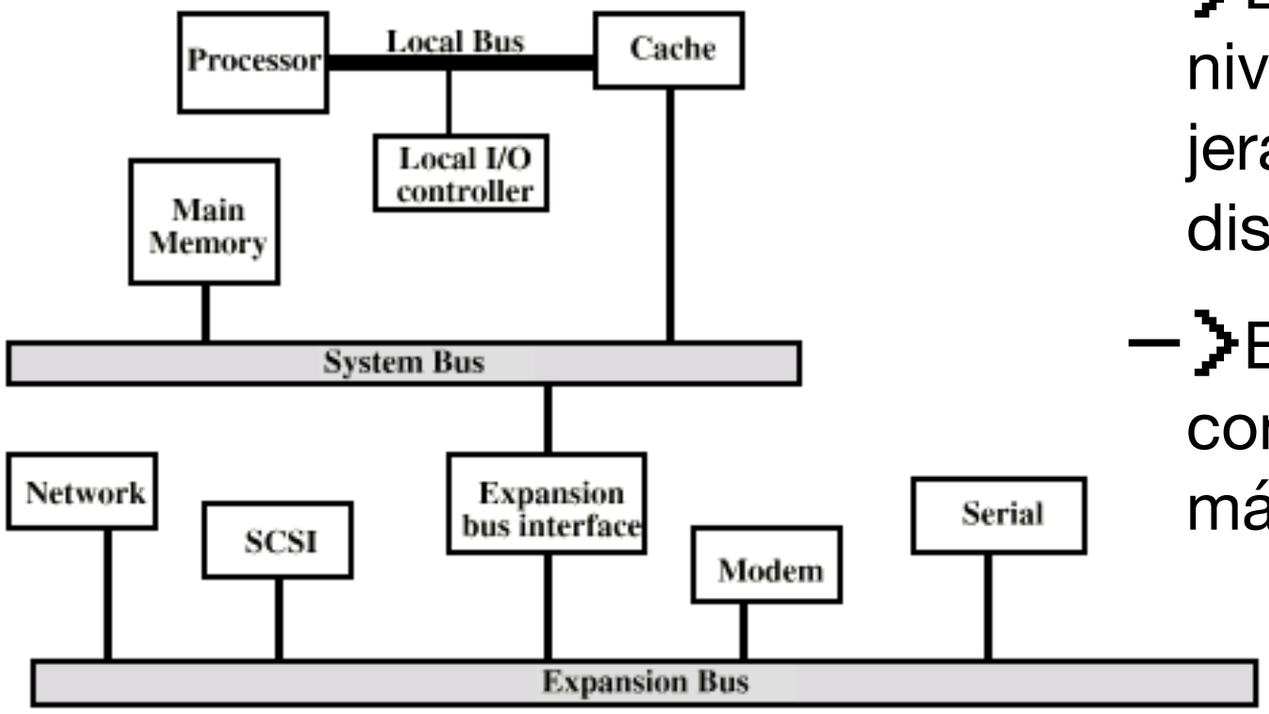
**[*]** El término “cuello de botella de von Neumann” fue acuñado por John Backus en su conferencia de la concesión del Premio Turing ACM de 1977.

# Buses

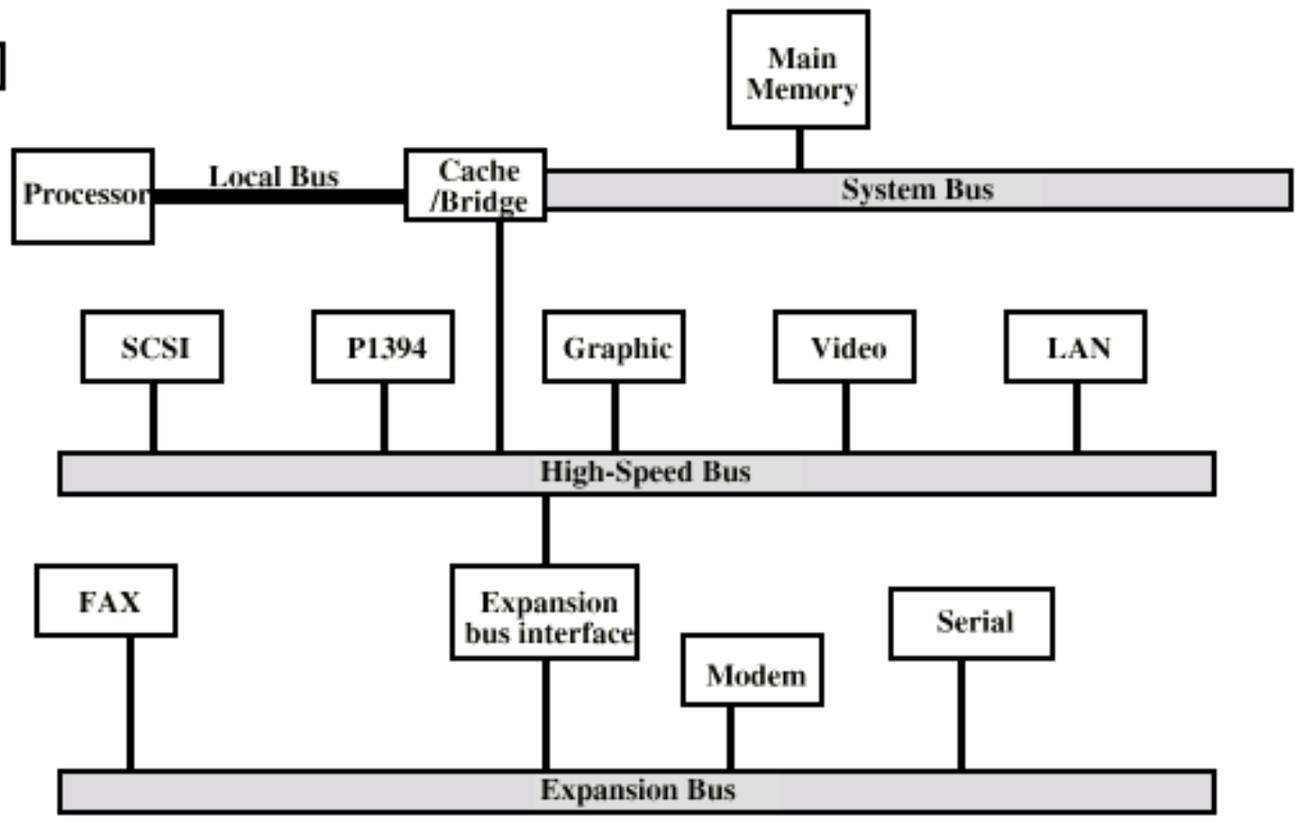



- Un **bus** está formado por un conjunto de señales con un propósito específico
- Sirven a los efectos de conectar dos o más componentes y por consiguiente es un recurso compartido
- En general se cuentan con señales de direcciones (**Bus de direcciones**), señales de datos (**Bus de datos**) y señales de control (**Bus de control**)

# Buses



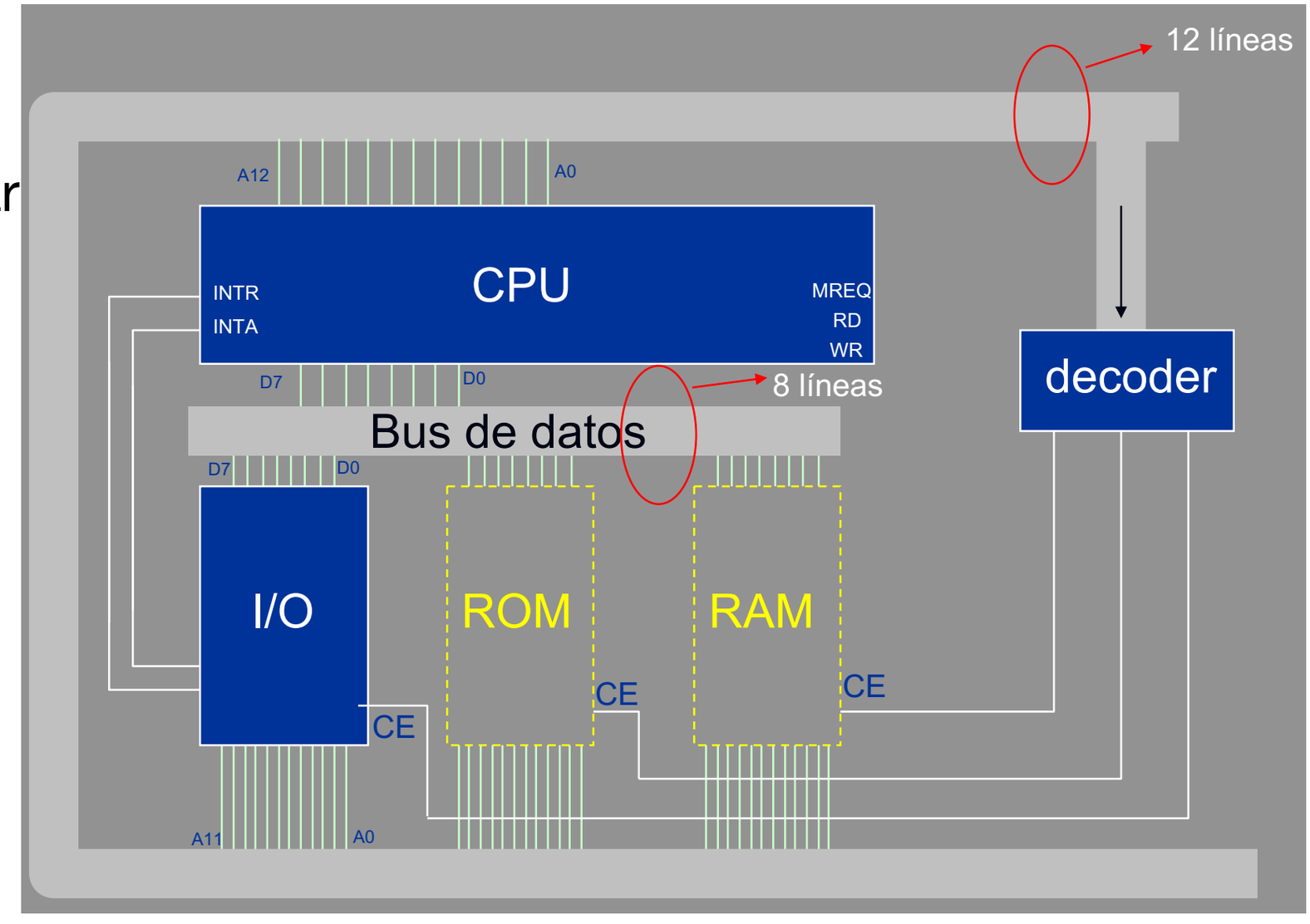

- Los buses pueden estructurarse a varios niveles con el objetivo de priorizar jerárquicamente el acceso a ciertos dispositivos
- En general se cuenta con un **local bus** que conecta el procesador con sus componentes más cercanas como ser la memoria Cache
- La conexión entre buses de distintos niveles se suele denominar **bridge**
- A medida que nos alejamos del procesador los buses adquieren propósitos más específicos y pierden accesibilidad al procesador



# Diseño de un bus

## Tipos de líneas

- Las líneas de un bus se dicen que son **dedicadas** cuando es posible racionalizar su rol en el sistema de forma que este sea siempre el mismo
- **Dedicación física:** conectan siempre el mismo conjunto de componentes
- **Dedicación funcional:** realizan siempre la misma función

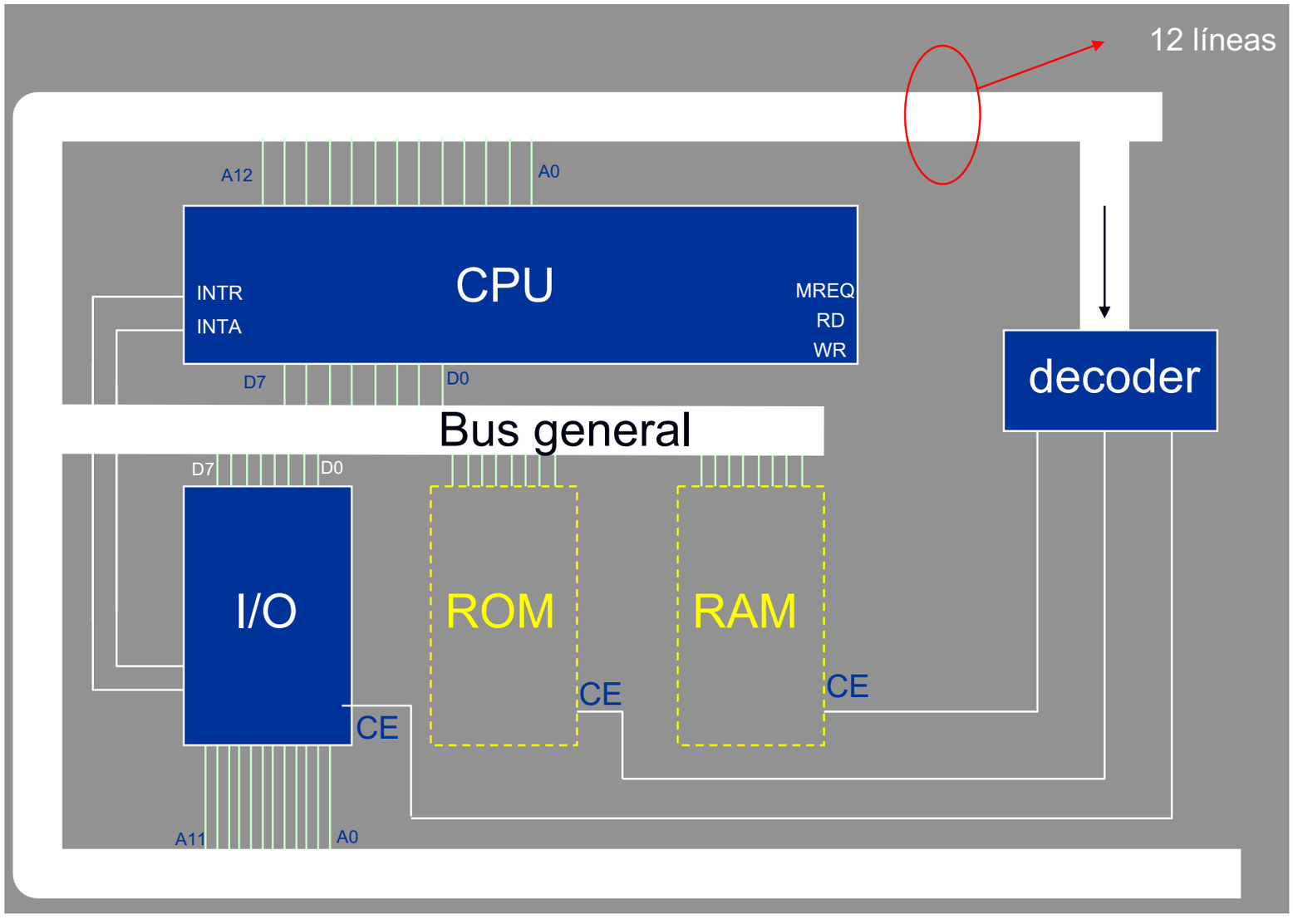



- La dedicación de las líneas implica menor cantidad de disputas por su uso al mismo tiempo que incrementa el tamaño y costo del sistema

# Diseño de un bus

## Tipos de líneas

- Las líneas de un bus se dicen que son **multiplexadas** cuando cumplen más de un rol en el sistema
- Un ejemplo clásico de multiplexación de bus es cuando el bus de datos y de direcciones comparten las mismas líneas físicas
- La multiplexación de las líneas implica menor cantidad de líneas físicas reduciendo el tamaño y costo del sistema pero complejizando la lógica de administración necesaria para resolver los accesos al recurso



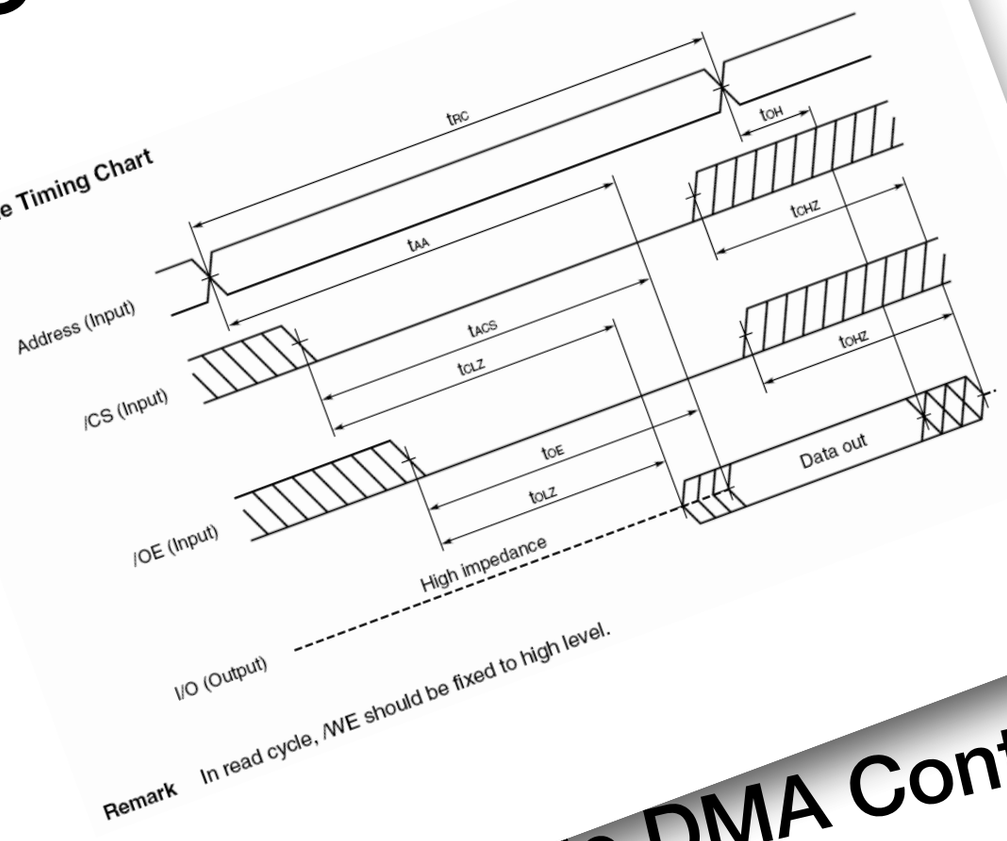
# Diseño de un bus

## Ancho del bus

- Se denomina **ancho de un bus** a la cantidad de líneas físicas que este utiliza con el objetivo de desarrollar su rol en el sistema
- La cantidad de líneas físicas de los buses es uno de los principales elementos que afecta su desempeño:
  - El ancho del **bus de direcciones** determina el espacio de memoria direccionable y por lo tanto su tamaño máximo
  - El ancho del **bus de datos** determina la cantidad de accesos a memoria necesarios para transferir la misma cantidad de información (el tamaño de la palabra)

# Memoria SRAM

**Read Cycle Timing Chart**

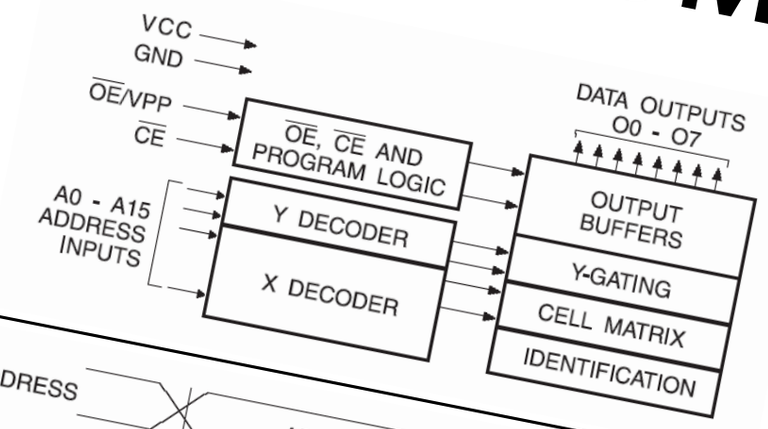


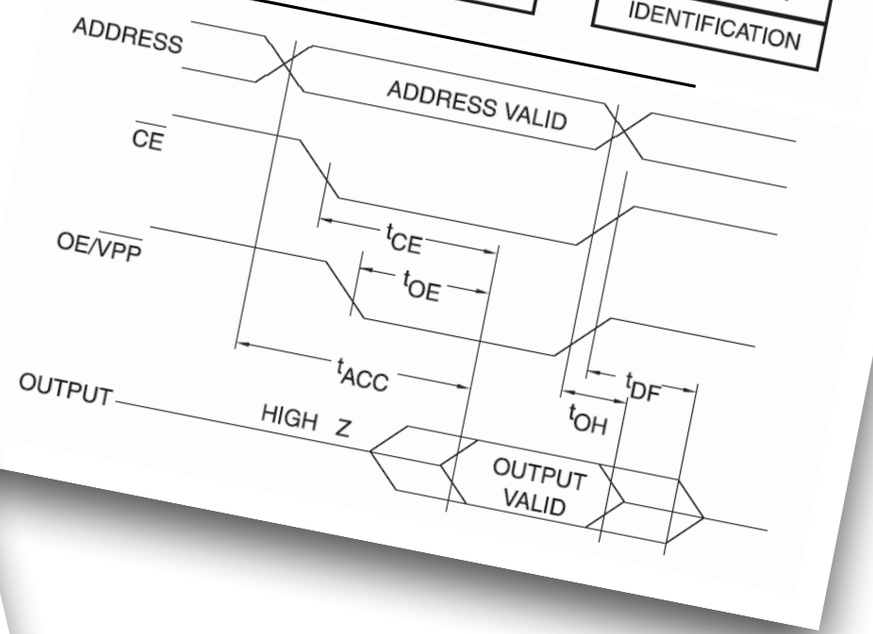

**Remark** In read cycle, /WE should be fixed to high level.

**ama de tiempos**

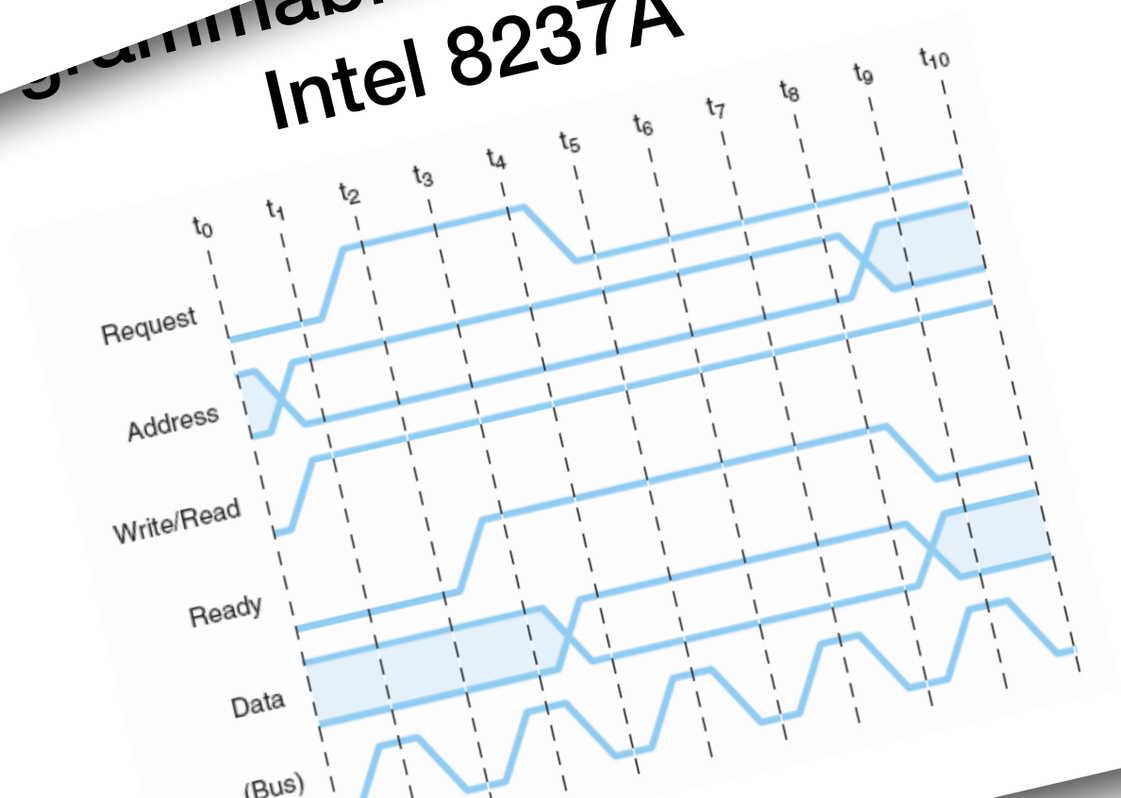
# Memoria OTP[*] EPROM

**grama de bloques**



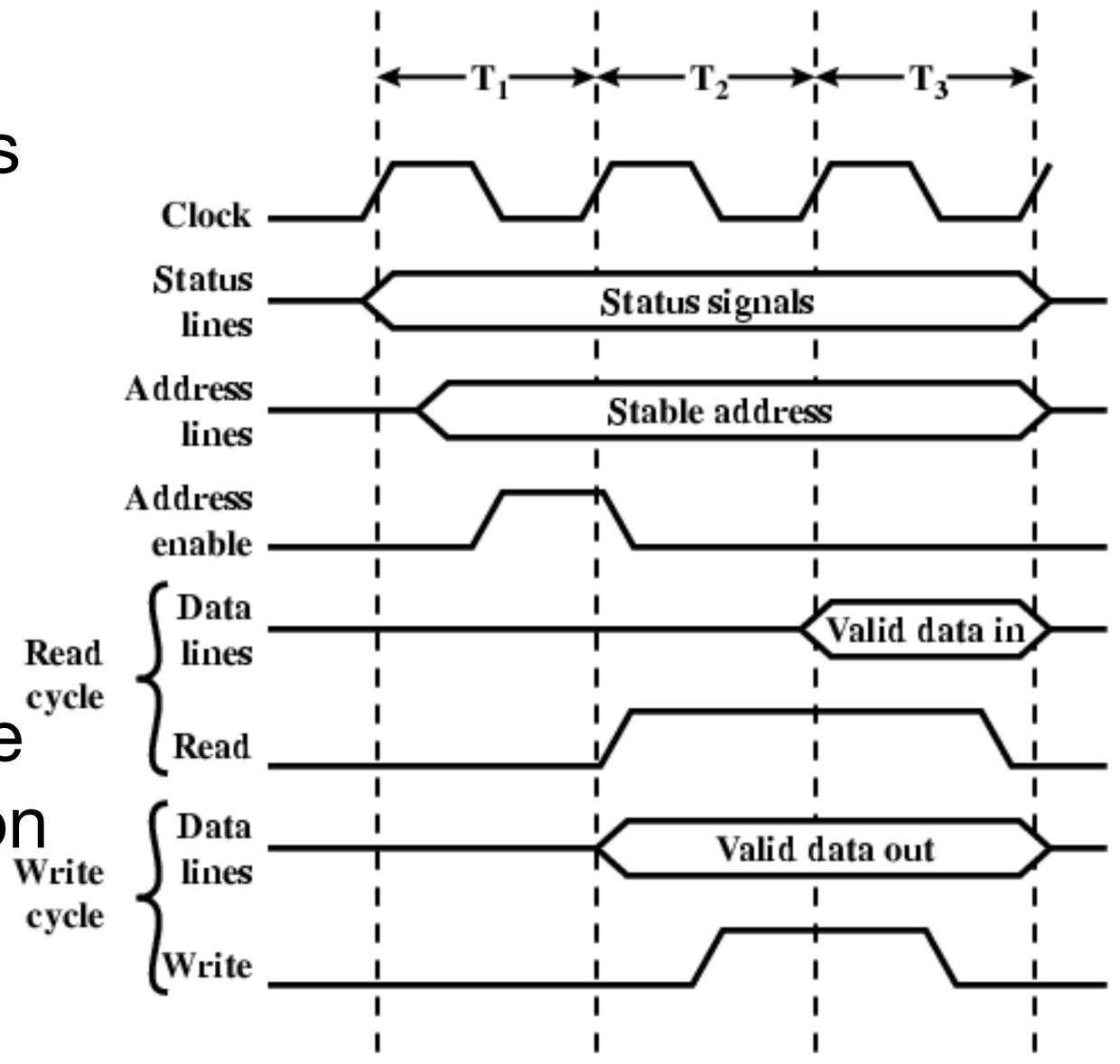

# Programmable DMA Controller Intel 8237A



# Diseño de un bus

## Temporización sincrónica

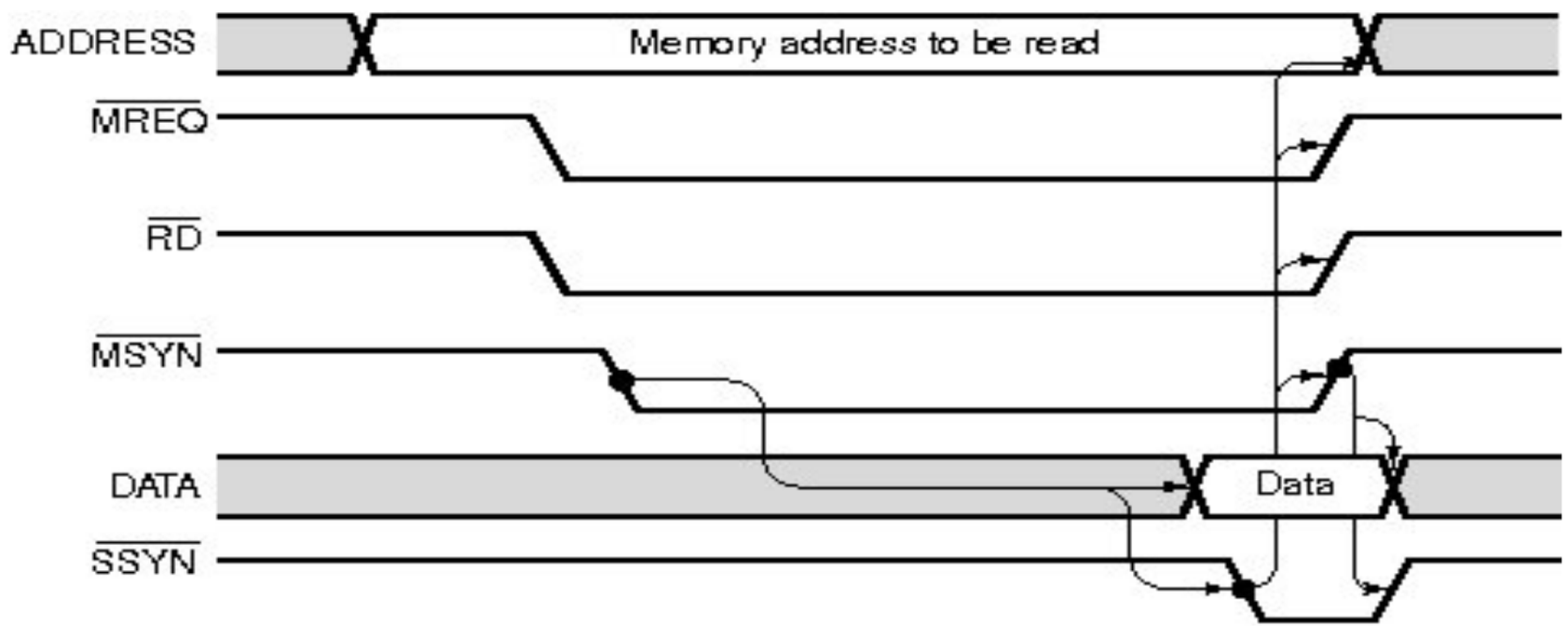
- El correcto desempeño de un bus requiere que ciertos eventos se encuentren coordinados
- La coordinación de eventos impone la necesidad de temporización, es decir, de un reloj
- **En la imagen**, las señales que conforman las líneas de dirección deben tener un valor estable cuando se habilita la lectura



# Diseño de un bus

**Temporización asincrónica**

- El correcto desempeño de un bus requiere que ciertos eventos se encuentren coordinados
- La coordinación de eventos no se encuentra regulada por un reloj sino por una relación de precedencia entre los eventos. La ocurrencia de ciertos eventos dispara otros eventos
- **En la imagen**, la señal MSYN en 1 dispara la colocación de los datos en DATA y 1 en la línea SSYN; esta a su vez lo hace sobre MREQ, RD y MSYN



# Diseño de un bus

**Líneas dedicadas**

- **Escritura:** La componente **A** que escribe coloca la dirección en el bus de direcciones y los datos en el bus de datos en un mismo ciclo de reloj del bus; luego notifica de este hecho a la componente **B**
- **Lectura:** La componente **A** que lee coloca la dirección en el bus de direcciones y notifica de este hecho a la componente **B** en un ciclo de reloj del bus; luego, la componente **B** coloca los datos en el bus de datos y notifica a la componente **A** de esto en otro ciclo de reloj

# Diseño de un bus

### Líneas multiplexadas

- **Escritura:** La componente **A** que escribe coloca la dirección en el bus de direcciones y lo notifica a la componente **B** en un ciclo de reloj del bus, la componente **B** notifica que la dirección ya fue leída; luego la componente **A** coloca los datos en el bus de datos en otro ciclo de reloj del bus y luego notifica de este hecho a la componente **B.**
- **Lectura:** La componente **A** que lee coloca la dirección en el bus de direcciones y notifica de este hecho a la componente **B** en un ciclo de reloj del bus; luego, la componente **B** coloca los datos en el bus y notifica a la componente **A** de esto en otro ciclo de reloj

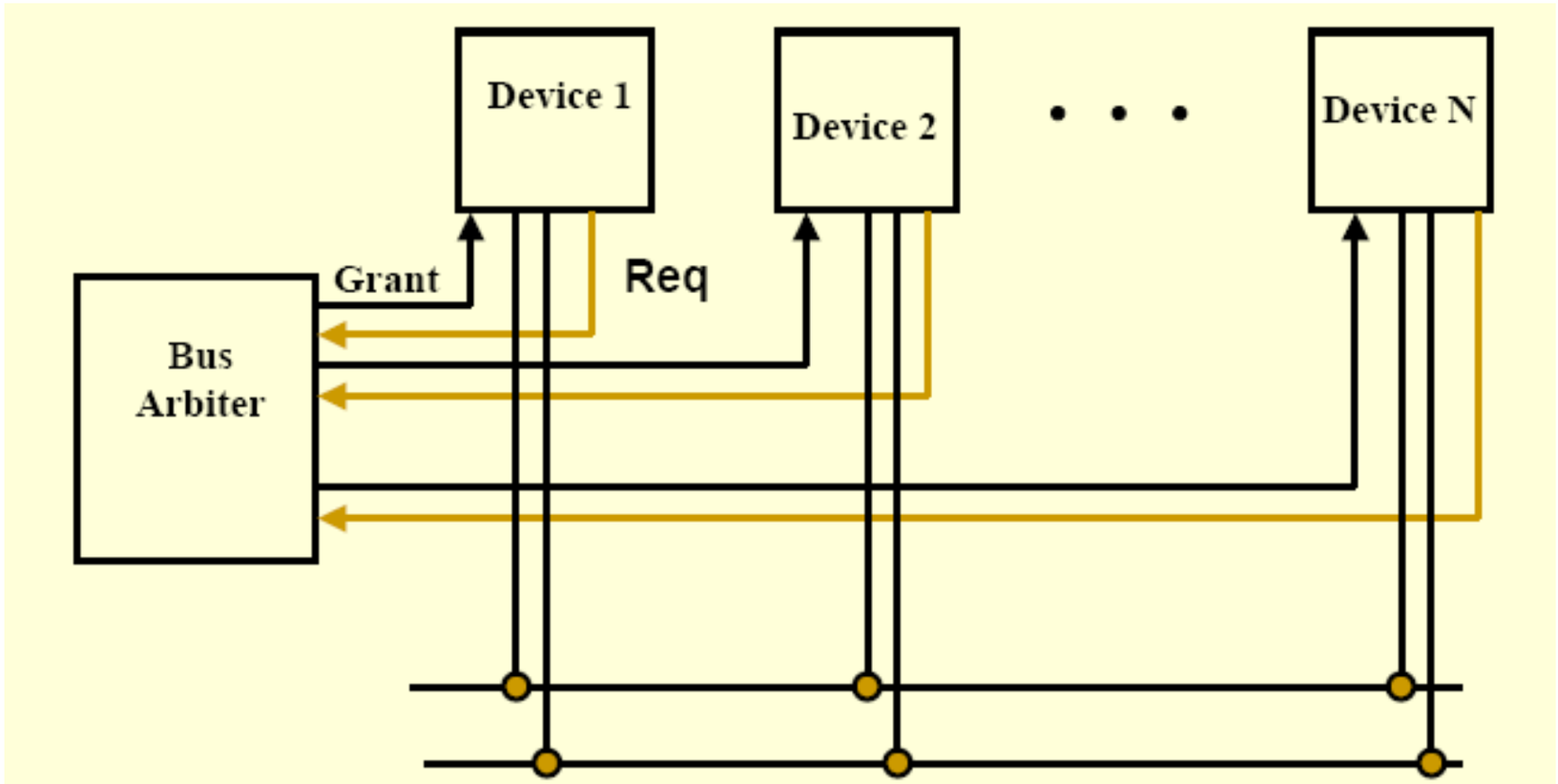
# Diseño de un bus

## Arbitraje

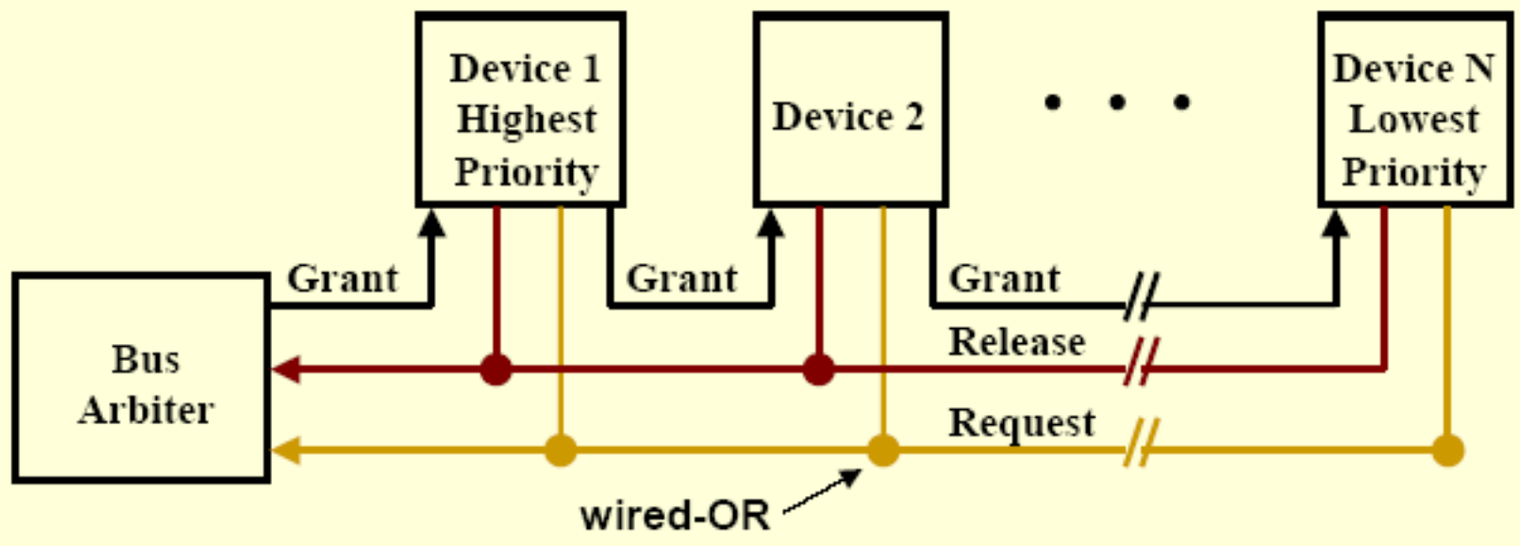
- Los buses son recursos compartidos requeridos por muchas componentes al mismo tiempo, por ejemplo, la CPU puede necesitar datos de la memoria al mismo tiempo que un dispositivo de entrada/salida puede necesitar leer/escribir en esta
- Para decidir qué componente recibirá el acceso se lleva a cabo un **arbitraje** del bus:
  - **Centralizado:** requiere de un controlador que implementa la lógica de arbitraje; las componentes solicitan acceso y el controlador es quien lo concede
  - **Distribuido:** las componentes implementan un subsistema de control de acceso y entre todos estos subsistemas administran el recurso

# Diseño de un bus

**Arbitraje centralizado**




**Arbitraje distribuido**



# PC Bus

- Estándar de IBM para la IBM PC que utilizaba un Intel 8088
- 62 líneas:
  - 20 líneas de direcciones: 1 Mb de memoria direccionable
  - 8 líneas de datos: palabra de 8 bits (1 byte)
  - Lineas de control entre las que se encuentran: lectura/escritura en memoria, lectura/escritura en entrada/salida, interrupciones, DMA, etc.

- Intel lanza el 80286 con bus de direcciones de 24 bits (16 Mb direccionables) y bus de datos de 16 bits. IBM lo estandariza bajo el nombre PC-AT

# Bus ISA

**(Industrial Standard Architecture)**

- Se rediseña el bus integrando la extensión PC-AT al PC Bus de forma que sea compatible
- 62 líneas:
  - 24 líneas de direcciones (A0–A19 - A20-A23): 16 Mb de memoria direccionable
  - 16 líneas de datos (D0–D7 - D8–D15): palabra de 16 bits (2 byte)
  - Lineas de control entre las que se implementa a acceso a dos controladores de interrupciones (IRQ8–IRQ15) y dos de DMA, etc.

# Evolución de bus ISA

**(Industrial Standard Architecture)**

- **Bus EISA:** Intel introduce el 80386 con un bus de datos de 32 bits y de direcciones de 32 bits.

  Electrónicamente complejo, poco escalable y con una velocidad máxima de 33 Mhz que resulta un cuello de botella respecto del aumento de velocidad del procesador

- **VESA Bus Local:** Desarrollado por un consorcio de fabricantes de controladores de video.

  Se extiende ISA pero con líneas de control dedicadas para video. Con velocidad de entre 25 - 50 Mhz, pero no contempla otro tipo de dispositivos

# Bus PCI

**(Peripheral Component Interconnect)**

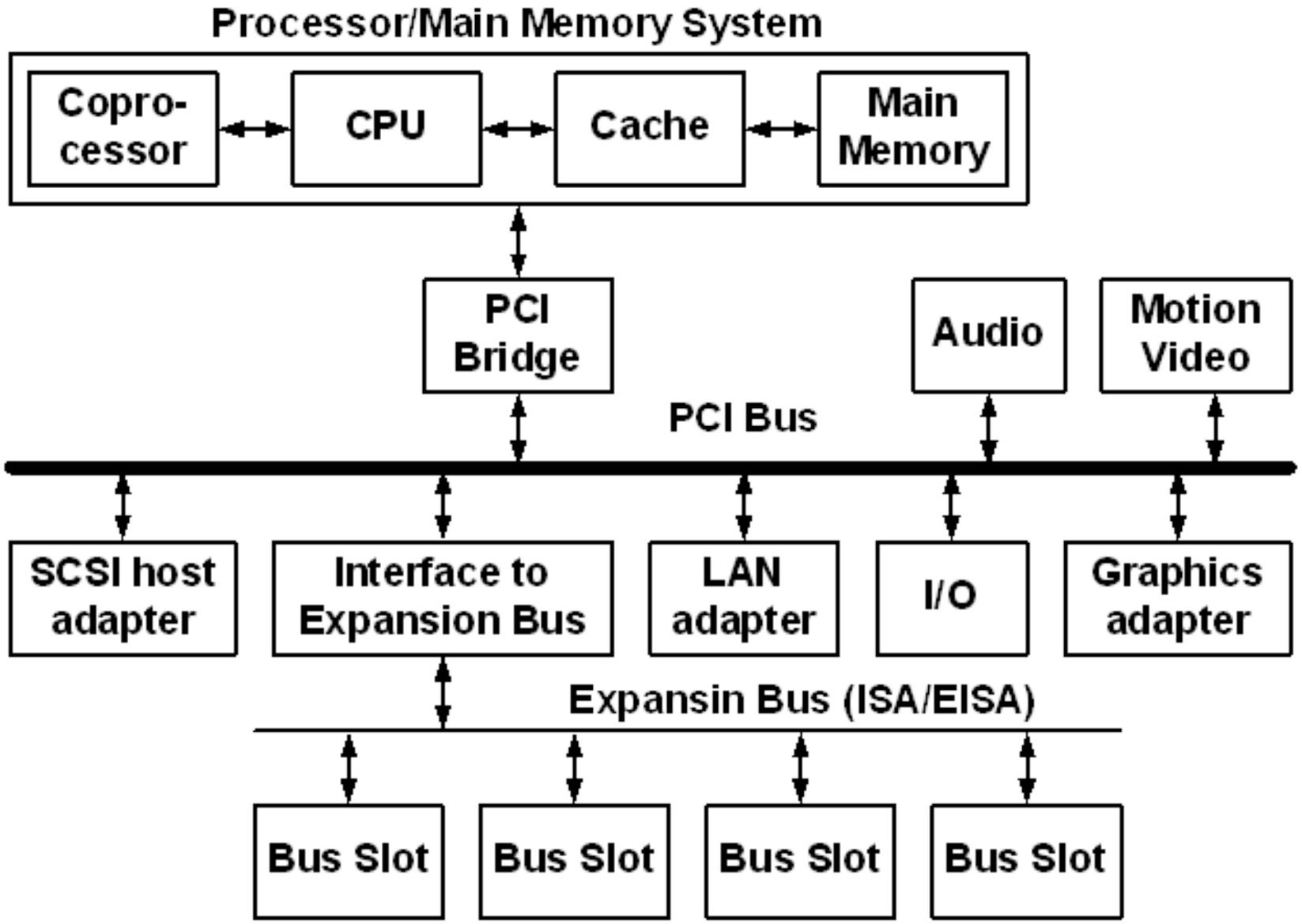
- Se desarrolla para dar soporte de alta velocidad, general y escalable frente a las opciones EISA y VESA Local Bus.

- En 1990 lo estandariza Intel en su **versión 1.0**:
  - 32 líneas de datos
  - 33 Mhz de velocidad
  - De implementación sencilla e incluye la posibilidad de interconectar con otros buses como ISA
- **Versión 2.0:** 66 Mhz (4.2 Giga bits p/seg. = 528 Mb p/seg.)
- **Versión 2.1:** 64 líneas de datos

# Bus PCI

## (Peripheral Component Interconnect)

### Diagrama de bloques



# Bus PCI

## (Peripheral Component Interconnect)

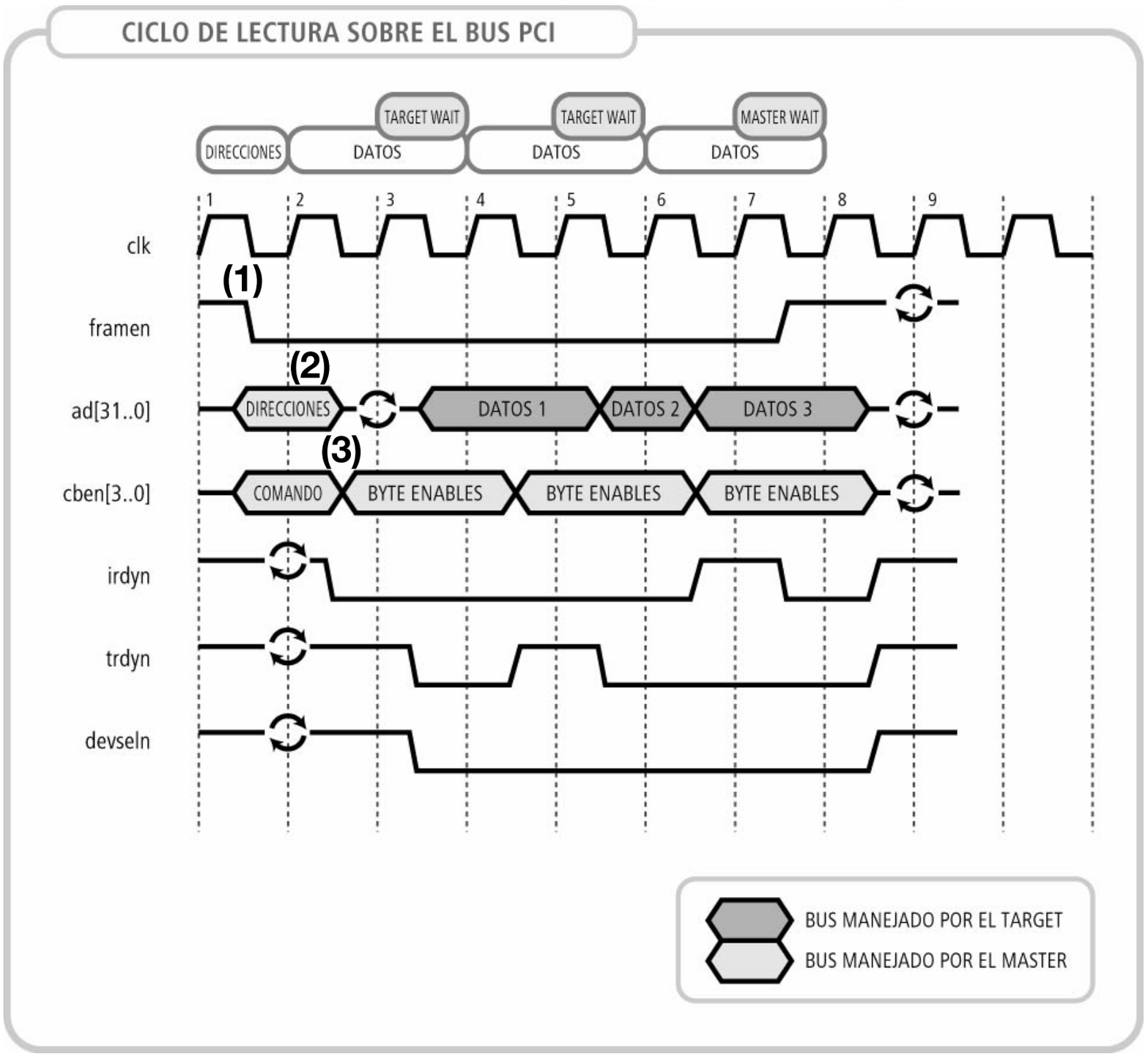
### 49 señales obligatorias

| Terminal | Línea | Nº | Habilita | Descripción |
|---|---|---|---|---|
| Sistema | CLK | 1 | Externo | Reloj de frecuencia 33 ó 66 MHz |
| | RST# | 1 | Externo | Restablece el sistema y los dispositivos |
| Datos y direcciones | AD | 32 | Maestro/destino | Líneas de dirección y datos multiplexadas |
| | C/BE# | 4 | Maestro/destino | Indica que líneas transportan información |
| | PAR | 1 | Maestro/destino | Bit de paridad de dirección o datos |
| Control | FRAME# | 1 | Maestro | Transferencia (AD y C/BE preparadas) |
| | IRDY# | 1 | Maestro | Lectura: maestro acepta datos/ Escritura: datos en AD |
| | TRDY# | 1 | Destino | Lectura: datos en AD/ Escritura: destino acepta datos |
| | STOP# | 1 | Destino | Detener transacción |
| | IDSEL | 1 | Maestro | Selección de inicio de dispositivo |
| | DEVSEL# | 1 | Destino | Dispositivo escuchando |
| Arbitraje | REQ# | 1 | Externo | Solicitud de bus |
| | GNT# | 1 | Maestro | Bus concedido |
| Error | PERR# | 1 | Maestro/destino | Se ha detectado un error de paridad en los datos |
| | SERR# | 1 | Todos | Error crítico o error de paridad en la dirección |

Entre las **señales opcionales** encontramos soporte de interrupciones para cada dispositivo, soporte de cache, 32 líneas de datos y direcciones multiplexadas adicionales, etc.

# Bus PCI

**(Peripheral Component Interconnect)**



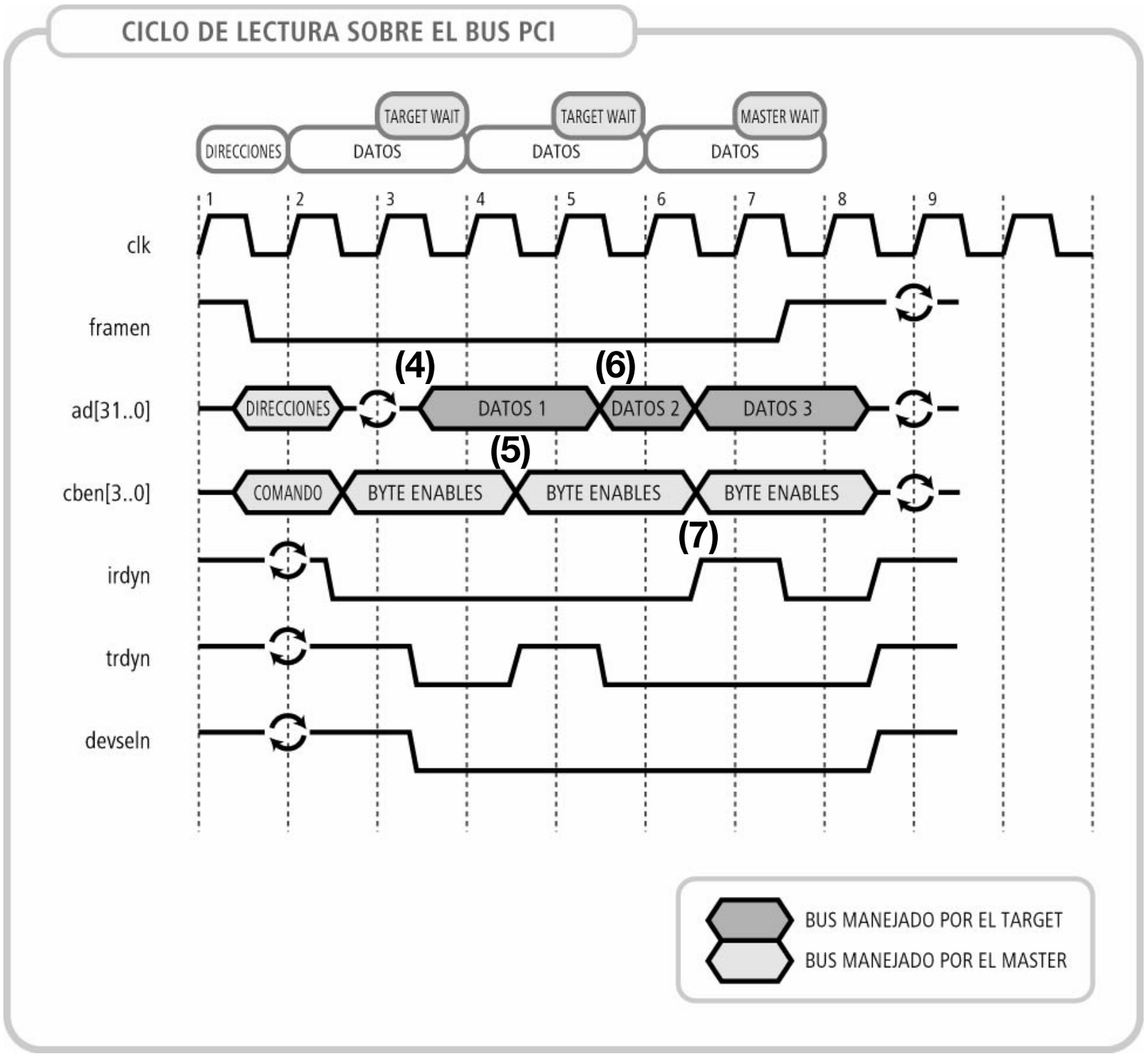

1. El *master* obtiene el control del bus lo notifica **framen** = 0, coloca la dirección en **ad**[31..0] en el flanco del ciclo de reloj y establece la operación a realizar con las líneas **cben** (lectura o escritura den memoria o en entrada/ salida)

2. Al comienzo del siguiente ciclo de reloj el *slave* reconoce la dirección

3. El *master* abandona las líneas **ad** y determina cuales de ellas se usaran para la transmisión usando **cben**. Luego activa la señal **irdyn** (initiator ready)

# Bus PCI

**(Peripheral Component Interconnect)**




4. El *slave* activa **devseln** para notificar que reconoció la dirección, coloca los datos en **ad** y activa **trdy** (target ready) para notificar que hay un dato válido en el bus

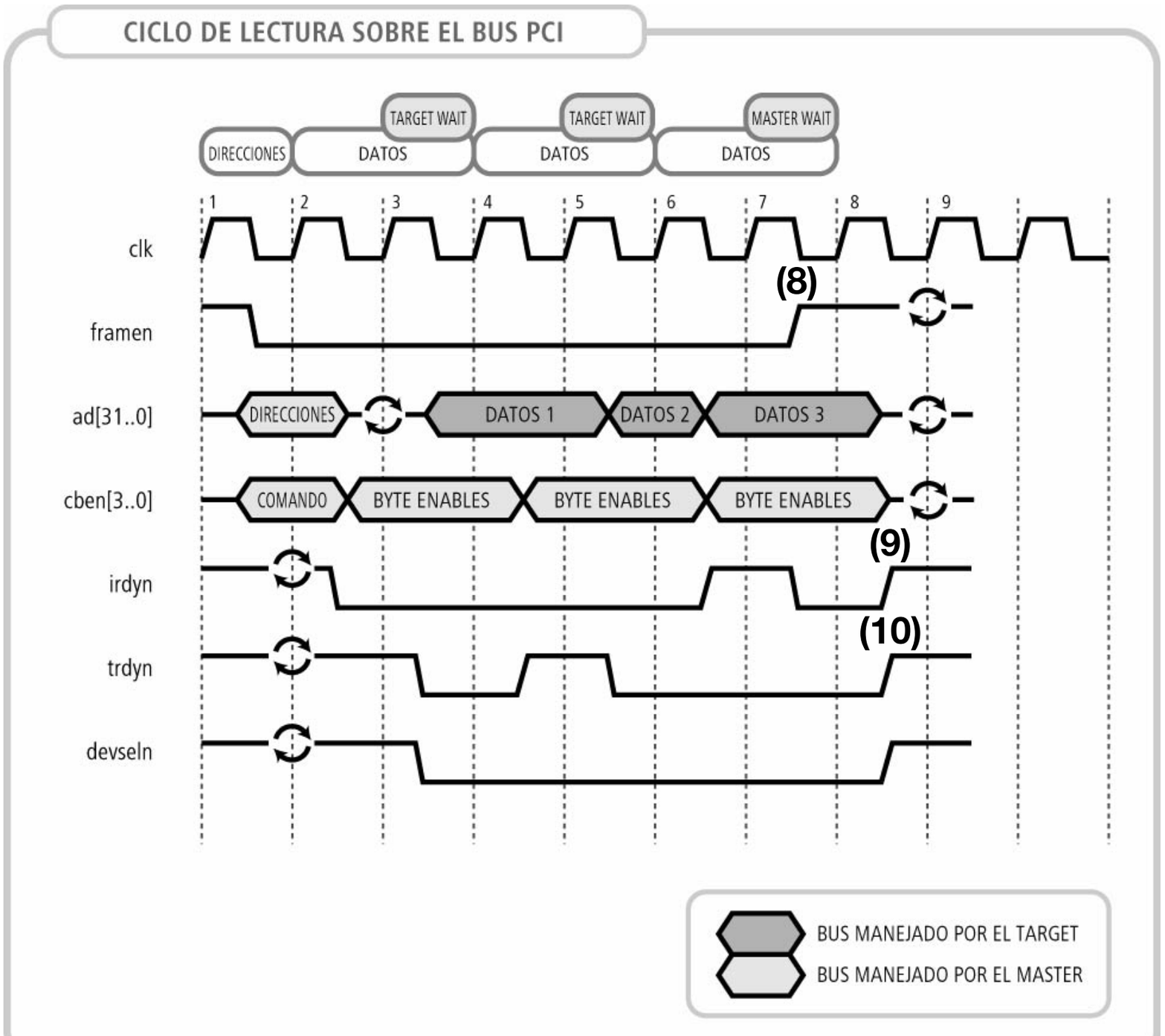
5. El *slave* cambia la línea **trdy** para notificar que se encuentra preparando la transmisión del siguiente bloque

6. El *slave* coloca el dato y cambia la línea **trdy** notificando al *master* que ya puede leer

7. Si el *master* estuviera ocupado y no pudiera leer usa la señal **irdyn** para notificar al *slave* que no se encuentra leyendo

# Bus PCI

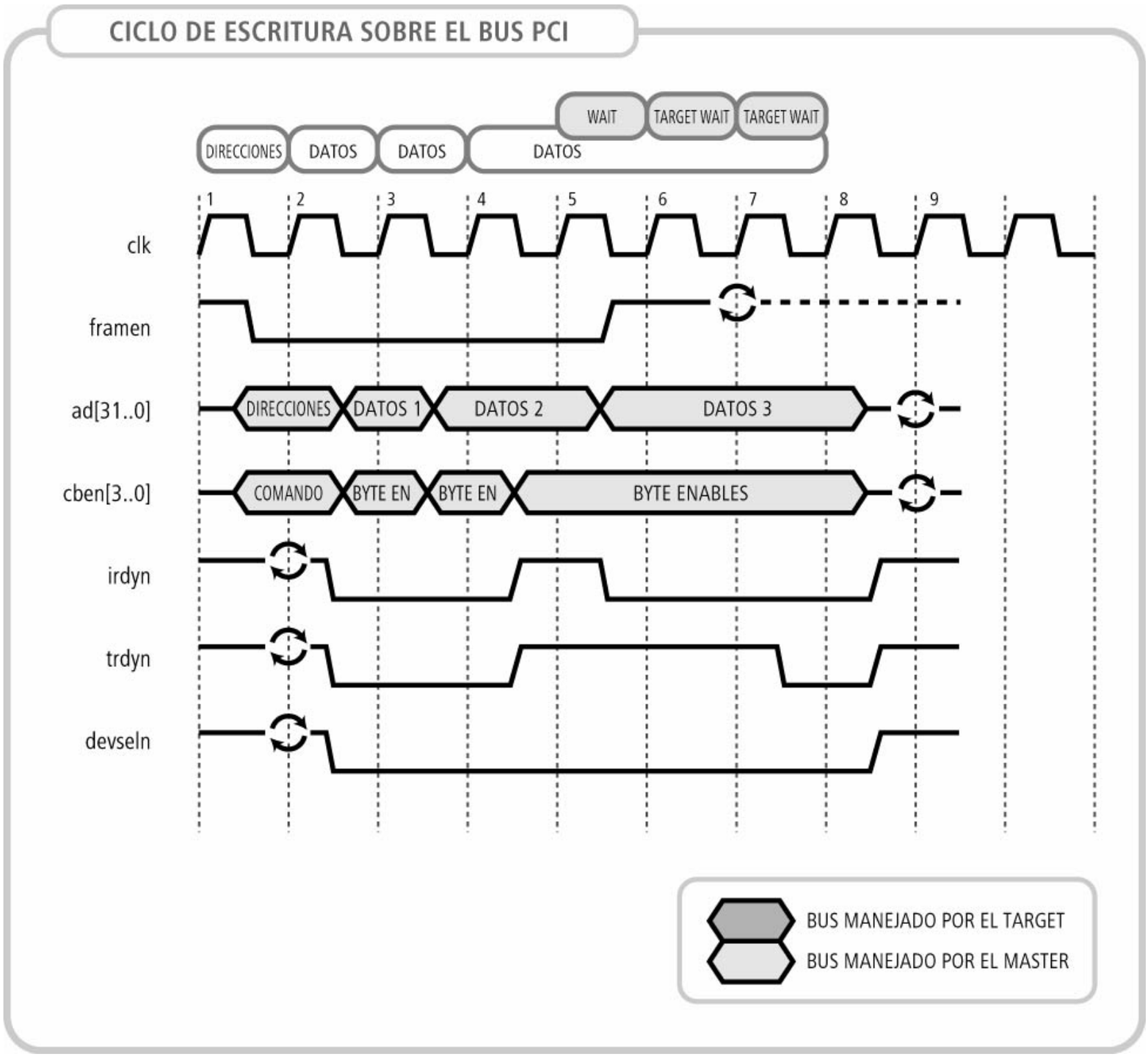
**(Peripheral Component Interconnect)**




8. El *master* sabe que el tercer dato es el último y lo expresa desactivando **framen**
9. El *master* desactiva la línea **irdyn** para finalizar la operación y liberando el bus
10. El *slave* desactiva la línea **trdyn** y **devsel** finalizando la operación y liberando el bus

# Bus PCI

## (Peripheral Component Interconnect)

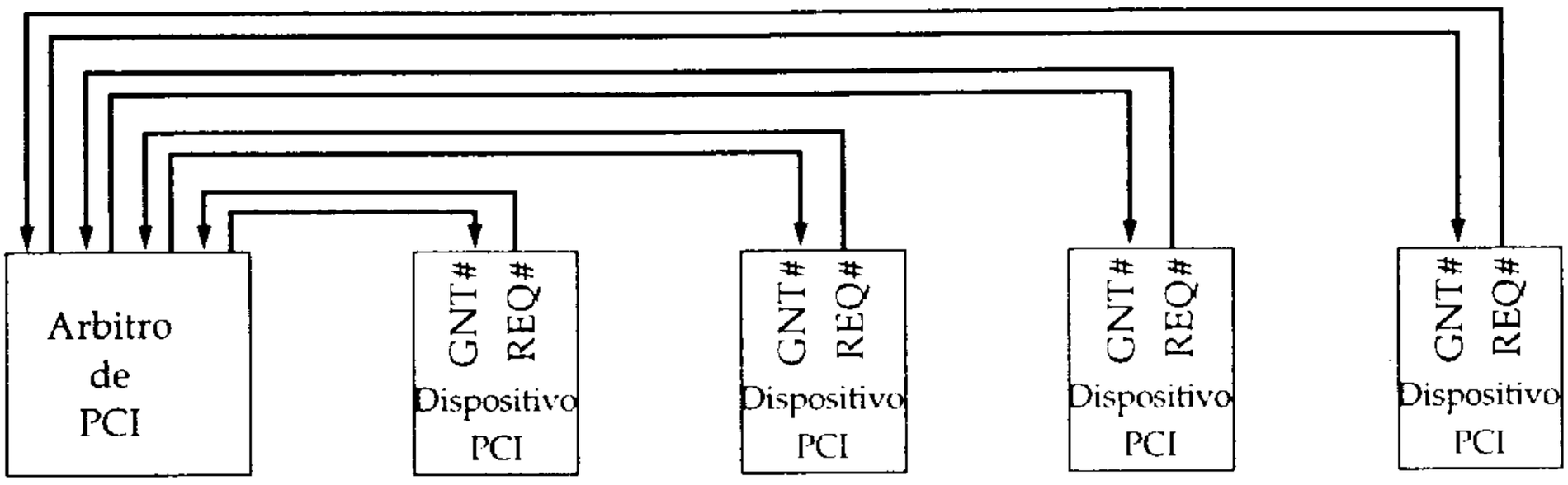


# Bus PCI

**(Peripheral Component Interconnect)**

**Arbitraje**

- Arbitraje centralizado en el controlador de bus PCI a través de dos líneas REQ (require) y GNT (grant)
- El dispositivo levanta la señal REQ para solicitar el acceso y espera por el arbitro que levante la señal GNT
- El dispositivo usa el bus mientras la señal GNT permanezca alta



# Interfaz serie

- **RS-232 (Recommended standard 232):** es una norma de intercambio de datos entre un DTE, o dispositivo terminal (por ejemplo, computadora) y un DCE, o equipo de comunicación (por ejemplo, un modem)
- La comunicación comienza con un *handshake* en que DTE y DCE acuerdan el intercambio de datos
- La transmisión se realiza en serie, es decir, uno tras otro sobre la misma línea organizada por frames con la forma:

**[**bit de arranque**][**bits de datos**][**bit de paridad**][**bit de parada**]**

# Interfaz serie

1 El DTE informa al DCE que desea transmitir datos levantando la señal DTR

2 El DCE informa al DTE que acepta la comunicación levantando la señal DSR

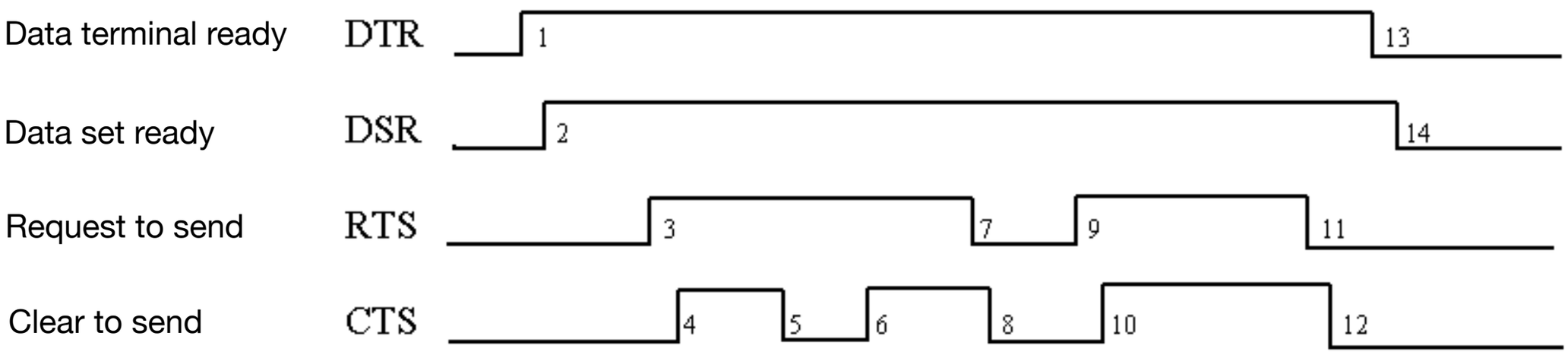
3 El DTE solicita permiso para enviar la información levantando la señal RTS

4 El DCE informa al DTE que está listo para recibir datos levantando la señal CTS

5 Si el DCE tiene el buffer interno lleno baja la señal CTS para informárselo al DTE; el DTE deja de enviar infomación

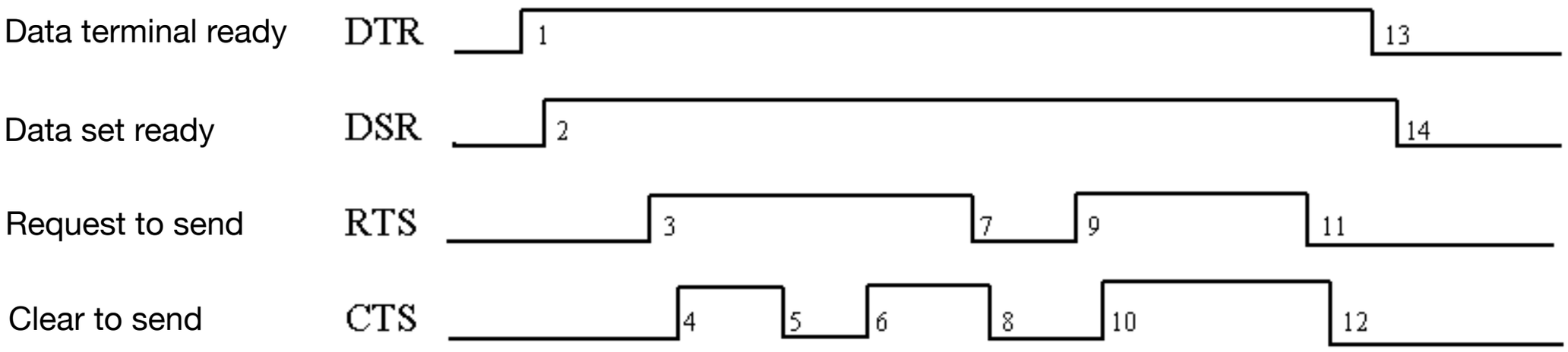
6 Una vez que el buffer del DCE fue vaciado, se informa al DTE que puede continuar levantando nuevamente la señal CTS

7 Si el DTE tiene su buffer lleno o no desea enviar más información, lo notifica al DCE bajando la señal RTS. Hay DCEs que se encuentran configurados para que la baja de esta señal indique que toda transmisión ha cesado

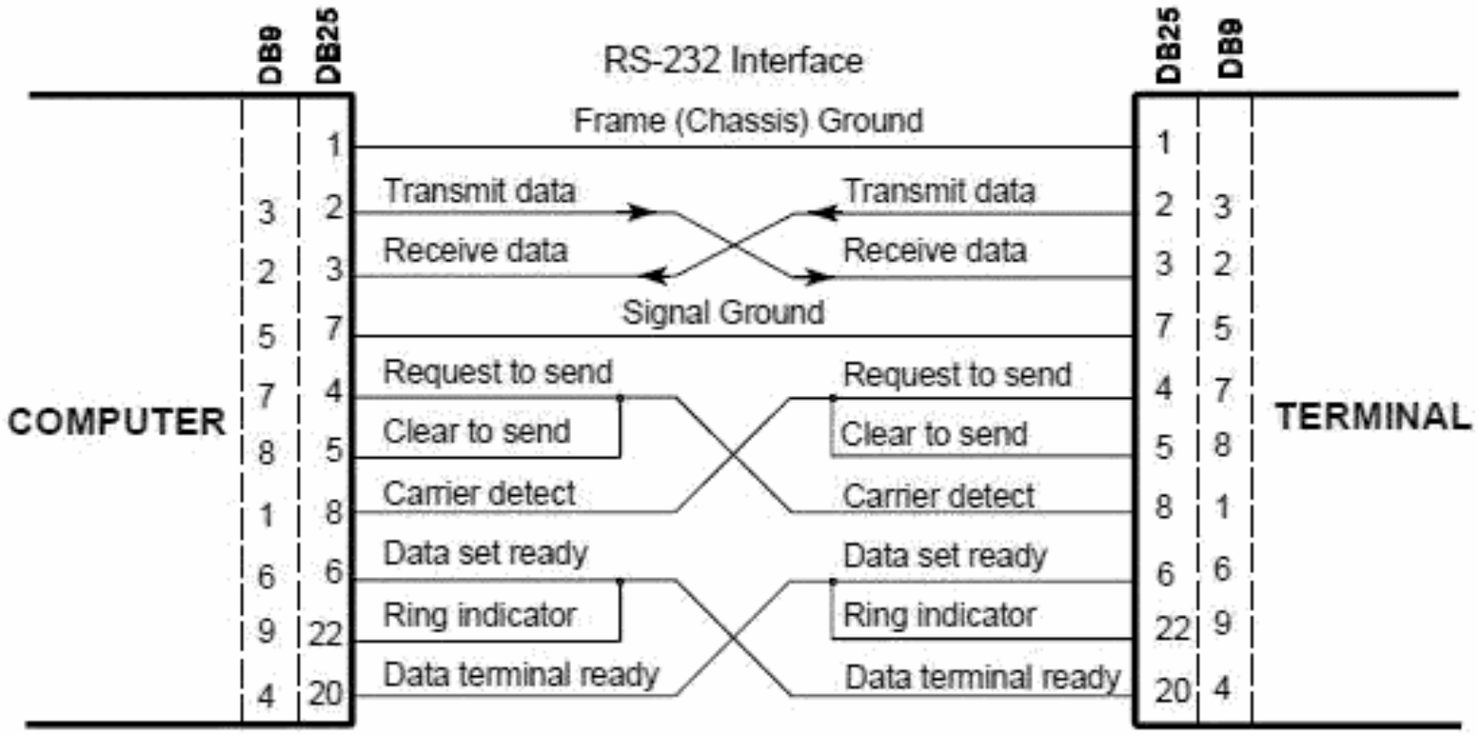


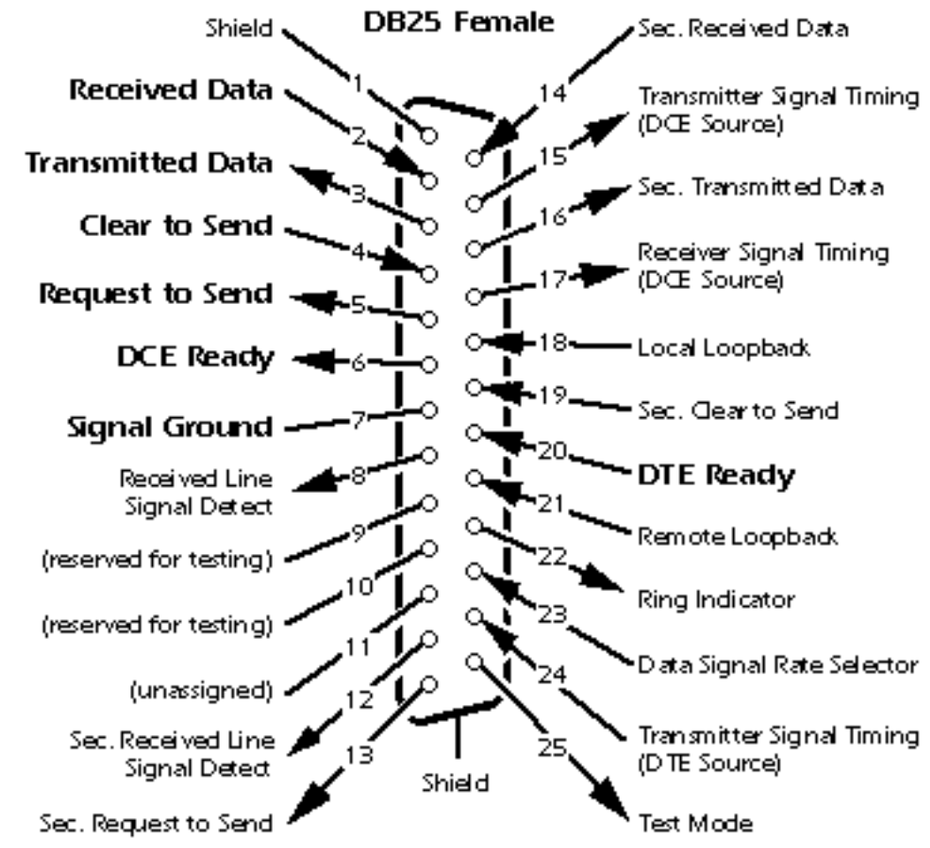
# Interfaz serie

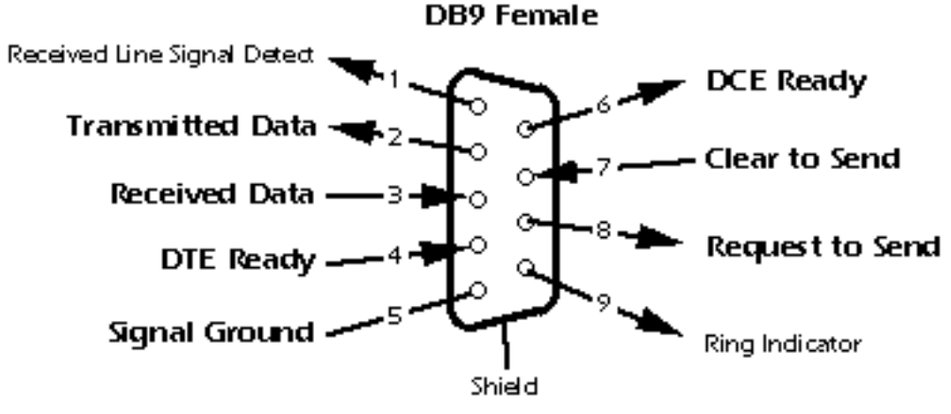
| | |
|---|---|
| 8 | El DCE reconoce la baja de la señal RTS bajando la señal CTS |
| 9 | El DTE levanta nuevamente la señal RTS indicando que desea continuar con la transmisión |
| 10 | El DCE levanta la señal CTS indicando que está listo para continuar |
| 11 | El DTE baja la señal RTS indicando que no desea transmitir más información |
| 12 | El DCE baja la señal indicando que reconoció la señal del DTE |
| 13 | El DTE baja la señal DTR informando al DCE que desea terminar la comunicación |
| 14 | El DCE reconoce la señal del DTE y lo informa bajando la señal DSR |

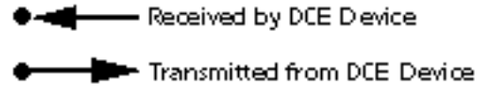


# Interfaz serie



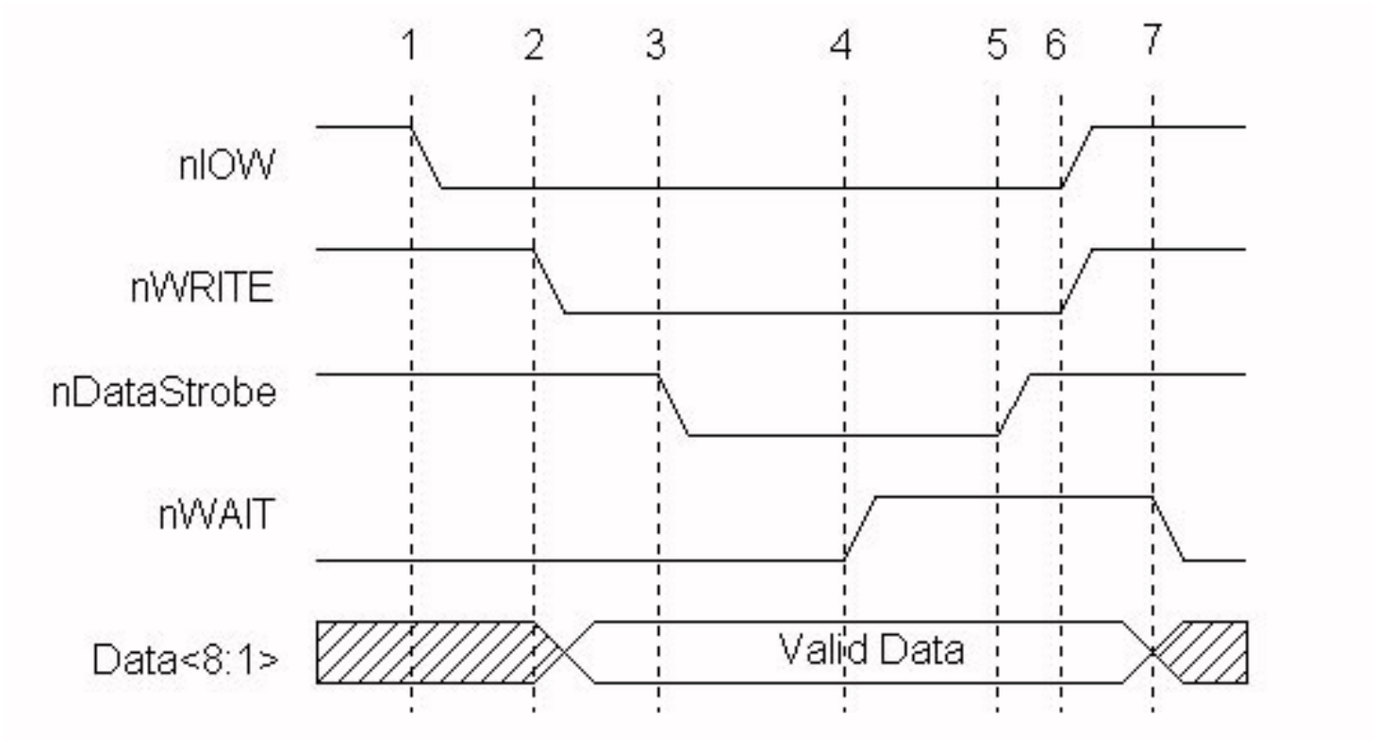
# Interfaz paralelo

- El **puerto paralelo** es una interfaz de comunicación entre una computadora y dispositivos, originalmente ideado para la conexión de dispositivos de impresión
- La comunicación comienza con un *handshake* en que la computadora y el dispositivo acuerdan el intercambio de datos
- Existen varios estándares de comunicación paralela: Centronics, IBM, Bi-Tronics, EPP y ECP, y Dataproducts, todos ellos fabricantes de controladores de puerto paralelo.

# Interfaz paralelo

## Data Write cycle

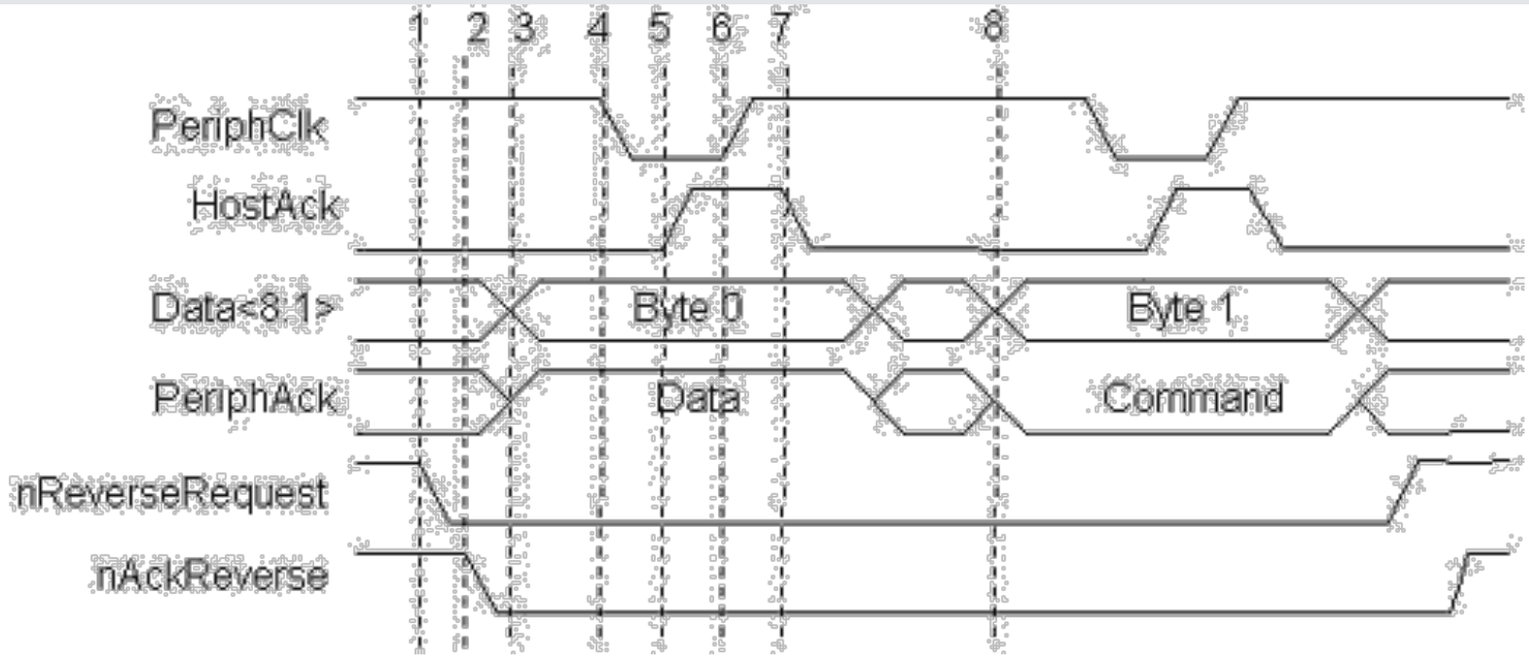
1. The program executes an I/O write cycle by lowering signal nIOW
2. The nWrite line is lowered and the data is output to lines D0—D7
3. The nDataStrobe is asserted, since nWAIT is asserted low
4. The program waits for the acknowledge from the peripheral (nWAIT deasserted)
5. The program deasserts nDataStrobe to finish the cycle
6. The ISA I/O cycle ends
7. The peripheral deasserts nWAIT to indicate that the next cycle may begin



# Interfaz paralelo

## Data Read cycle

1. The peripheral lower the signal nReverseReq asking for a reverse communication
2. The program lower the signal nAckReverse acknowledging the request
3. The peripheral puts data in D0—D7 raise signal PeriphAck
4. The peripherals clock lowers
5. The program raises the signal HostAck acknowledging the read
6. The peripherals clock raises
7. The program lowers the signal HostAck
8. A command cycle follows analogously but signal PeripheralAck is lowered



# Interfaz paralelo