

Arquitectura de Computadores

5. Sistemas de Entrada/Salida

1. Estructura de un Sistema de E/S
2. Métodos de E/S
3. E/S por sondeo (*polling*)
4. E/S por interrupciones
5. E/S por Acceso Directo a Memoria /DMA)

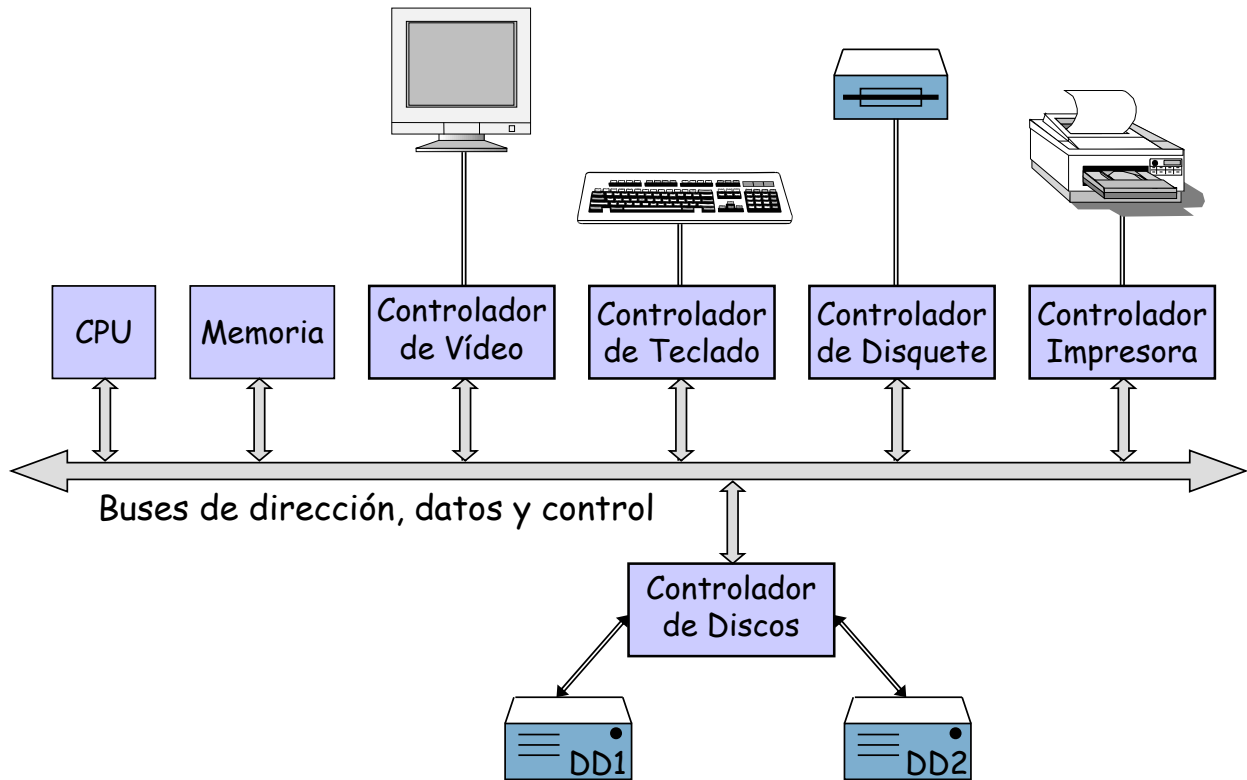
En los capítulos anteriores hemos visto los dos componentes básicos de un ordenador: el procesador y la memoria. No obstante, la actividad del procesador carecería de sentido si no estuviese relacionado con el mundo exterior. Además, es necesario que la tarea que se ejecuta en la CPU esté gobernada por los comandos y datos que hay en el exterior del ordenador. Igualmente, para que los resultados derivados de esa tarea tengan alguna utilidad, deben poder enviarse a algún medio externo al ordenador en el que puedan guardarse o representarse en un formato reconocible por el usuario.

Para la relación con el entorno exterior del ordenador se dispone de los dispositivos de entrada/salida, tales como terminales, impresoras, pantallas, etc. Algunos de estos dispositivos se utilizan como almacenamientos de memoria, como los discos y cintas magnéticas. También hay otro tipo de dispositivos periféricos utilizados en aplicaciones de control industrial, como los sensores de temperatura, de presión, alarmas sonoras, dispositivos de cierre y apertura de válvulas, etc.

La recogida de un dato externo es similar a la operación de lectura en memoria e, igualmente, la entrega de un dato al entorno exterior es equivalente a la escritura en una posición de memoria. No obstante el comportamiento de los dispositivos de E/S es distinto a la memoria; por ejemplo, los datos en memoria siempre están disponibles, mientras que en los dispositivos externos pueden no estar siempre, y hay que esperar a que ocurra un cierto suceso o evento para que aparezca el dato. Por esto, veremos que los sistemas y métodos de entrada/salida son distintos a los utilizados para la memoria.

En este capítulo no nos vamos a ocupar del estudio de los distintos dispositivos periféricos que hay en el mercado. Lo que vamos a estudiar son los métodos y sistemas utilizados para realizar las operaciones generales de entrada/salida.

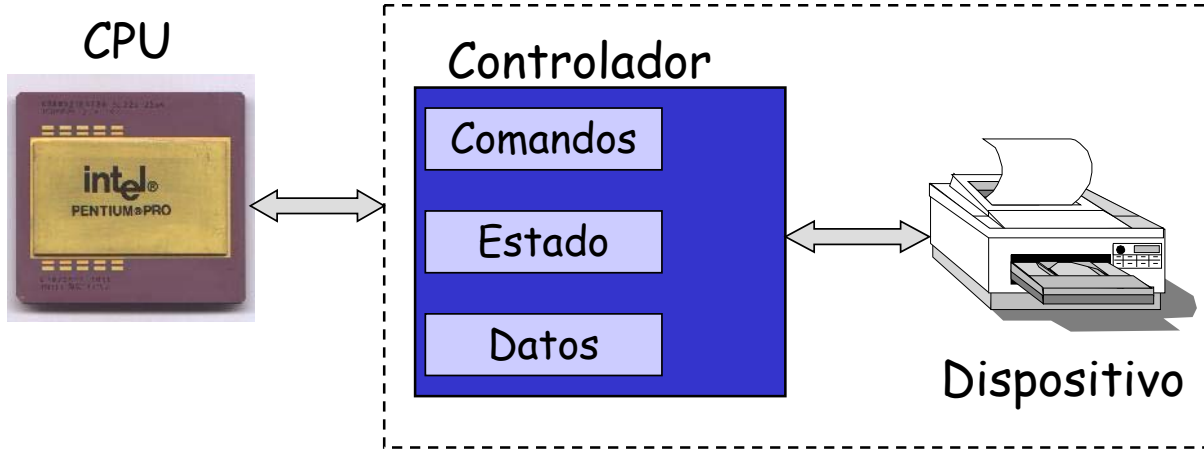
Estructura de un Sistema de E/S



Un sistema de entrada/salida requiere tanto los componentes hardware necesarios, como una estructura de programas que manejen adecuada y eficientemente los dispositivos. Veamos entonces la estructura tanto del hardware como de los programas implicados en las operaciones de entrada/salida.

Los dispositivos de E/S se comunican con la CPU y la memoria por los buses del sistema que ya conocemos. Estos dispositivos son heterogéneos y de características muy variadas, por lo que resultaría muy costoso que la CPU se encargara de manejarlos, tanto los dispositivos electrónicos (monitores de vídeo o sensores), como los que disponen de mucha mecánica en su funcionamiento (discos o cintas magnéticas).

Por esto, normalmente, cada dispositivo o grupo de dispositivos periféricos cuenta con un **controlador de dispositivo**, cuya electrónica incluye controladores y chips especializados en el manejo de ese tipo de dispositivos. Estos controladores de dispositivos admiten órdenes o comandos muy abstractos que les puede enviar la CPU, y que una vez recibidos se encargan de llevarlos a cabo, liberando así al procesador principal de realizar todas las tareas de bajo nivel, y con cualquier tipo de dispositivo.



Así, ya tenemos que el controlador de dispositivo actúa de interfaz entre la CPU y el dispositivo de E/S, permitiendo la realización de operaciones más o menos complejas con relativa facilidad. Cada controlador puede ocuparse de uno o varios dispositivos del mismo tipo y, en algunos casos, de realizar operaciones de distinta naturaleza (transmisión/recepción, *timer*, ...). Nosotros vamos a ocuparnos en este capítulo de cómo se trabaja con los controladores de dispositivo, que son los que tienen interfaz directa con el procesador principal. Así, para abreviar, cuando hablemos aquí de un dispositivo nos referiremos a algo formado por el propio dispositivo más el controlador correspondiente.

La forma que habitualmente ofrecen los controladores de dispositivos para recibir instrucciones del procesador, o de ofrecer información de su estado es mediante unos **registros** o **puertos**. Un controlador dispone de unos cuantos puertos (entre 2 y 30) que pueden ser de lectura, de escritura o de lectura-escritura. Los de escritura son los que se utilizan para suministrarle datos y órdenes o comandos de las operaciones que tiene que realizar, mientras que los de lectura sirven para leer datos e informar del estado interno.

Responsable del control de uno o más dispositivos y del intercambio de datos entre tales dispositivos y la CPU o la memoria.

Funciones de un Controlador de Dispositivo

- ✓ Control y temporización del flujo de datos
- ✓ Comunicación con la CPU
 - Decodificación de comandos
 - Intercambio de los datos de E/S con la CPU
 - Informar del estado del dispositivo
 - Reconocimiento de la dirección del dispositivo
- ✓ Comunicación con el dispositivo
- ✓ Almacenamiento temporal de datos (buffer)
- ✓ Detección de errores

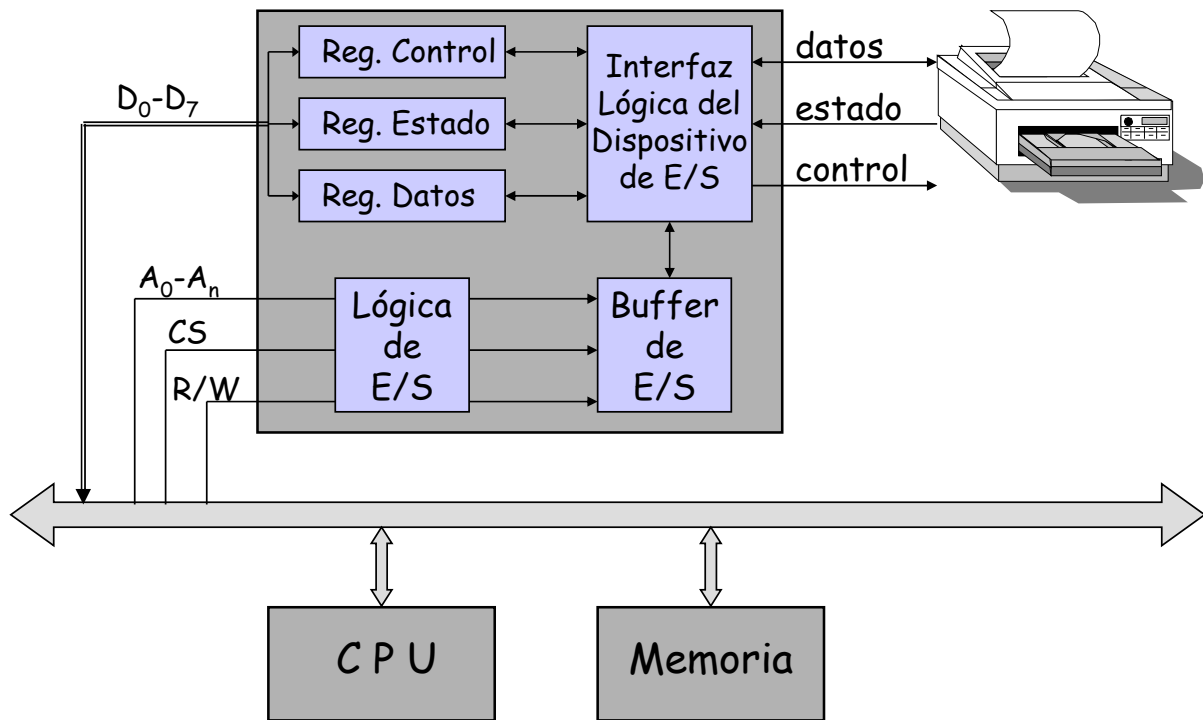
Para describir los controladores de dispositivos vamos a abordar su cometido y la estructura que suelen presentar. Comencemos por su cometido.

Un controlador de E/S es la entidad responsable del control de uno o más dispositivos del mismo tipo, y del intercambio de datos entre tales dispositivos y la memoria principal o los registros de la CPU. Así pues, un controlador tiene dos interfaces: una interfaz interna al ordenador, la que le comunica con la memoria y la CPU, y otra externa al computador, la que le comunica con el dispositivo periférico.

Las principales funciones de un controlador de entrada/salida son estas:

- **Control y Temporización**, para coordinar el flujo de tráfico entre la CPU o memoria, y el dispositivo periférico.
- **Comunicación con la CPU**, que implica las siguientes tareas:
 - 1) Decodificación de los comandos que vienen de la CPU (LEER SECTOR, ESCRIBIR SECTOR, POSICIONARSE EN PISTA n, ...)
 - 2) Intercambio de los datos de E/S con la CPU.
 - 3) Informar del estado del dispositivo (OCUPADO, PREPARADO, ...).
 - 4) Reconocimiento de la dirección del dispositivo, es decir, darse cuenta que los datos que vienen por los buses van dirigidos a este dispositivo y no a otro.
- **Comunicación con el Dispositivo**, es decir, envío de comandos, intercambio de datos y recepción de la información de estado.
- **Almacenamiento temporal de datos (buffer)**. Ya que la velocidad de acceso de la memoria es mucho más alta que la que proporcionan los dispositivos periféricos, el controlador dispone de una memoria local (rápida) con la que se comunica con la memoria y la CPU, así, puede recibir rápidamente un bloque de datos, liberar el bus, y luego escribirlo en el dispositivo a la velocidad que éste proporcione. Esto se aplica igualmente a las operaciones de lectura en el dispositivo, que proporciona lentamente datos que se almacenan en el buffer del controlador, y cuando se dispone de un bloque completo se transmite rápidamente a la memoria principal.
- **Detección de Errores**. Debe ocuparse de detectar y comunicar a la CPU los errores mecánicos o eléctricos del dispositivo.

Estructura de un Controlador



Arquitectura de Computadores

Sistemas de E/S - 5

Los controladores de dispositivos varían mucho en la complejidad y en el número de dispositivos que controlan, aunque suelen estar formados, básicamente, por uno o varios procesadores de propósito específico para el tipo de dispositivo al que está dedicado, de un buffer o memoria local, y de unos registros o puertos para comunicarse con la CPU y con el dispositivo.

La figura de la transparencia muestra el diagrama de bloques general de un controlador; en él podemos apreciar los registros internos y las líneas de comunicaciones.

El manejo de un controlador, desde la CPU, se realiza mediante sus registros internos. Escribiendo y leyendo los valores de estos registros se le pueden dar las ordenes, detectar su estado, recibir o enviar datos, etc. Sus registros más comunes suelen ser estos:

Registro de Control. Escribiendo en este registro se le envían ordenes al controlador, por ejemplo, el tipo de operación (lectura, escritura, ...).

Registro de Estado. Leyendo los valores de este registro se conoce el estado del controlador y del dispositivo, por ejemplo, si hay algún dato disponible, si el dispositivo está preparado u ocupado, si ha habido algún error, etc..

Registros de Datos. Puede haber registros de datos de entrada (por ej. el número de sector a leer), de datos de salida (un carácter que leído), o a veces son multifuncionales, siendo tanto de entrada como de salida.

Como vemos, el controlador tiene líneas de comunicación hacia la CPU y hacia el dispositivo controlado. Las líneas que se conectan al procesador los utiliza éste para realizar las operaciones de lectura/escritura sobre los registros internos del controlador. Los controladores de dispositivos suelen disponer de las siguientes señales:

A₀, A₁, A₂, ... Estas señales seleccionan uno de los registros internos del controlador.

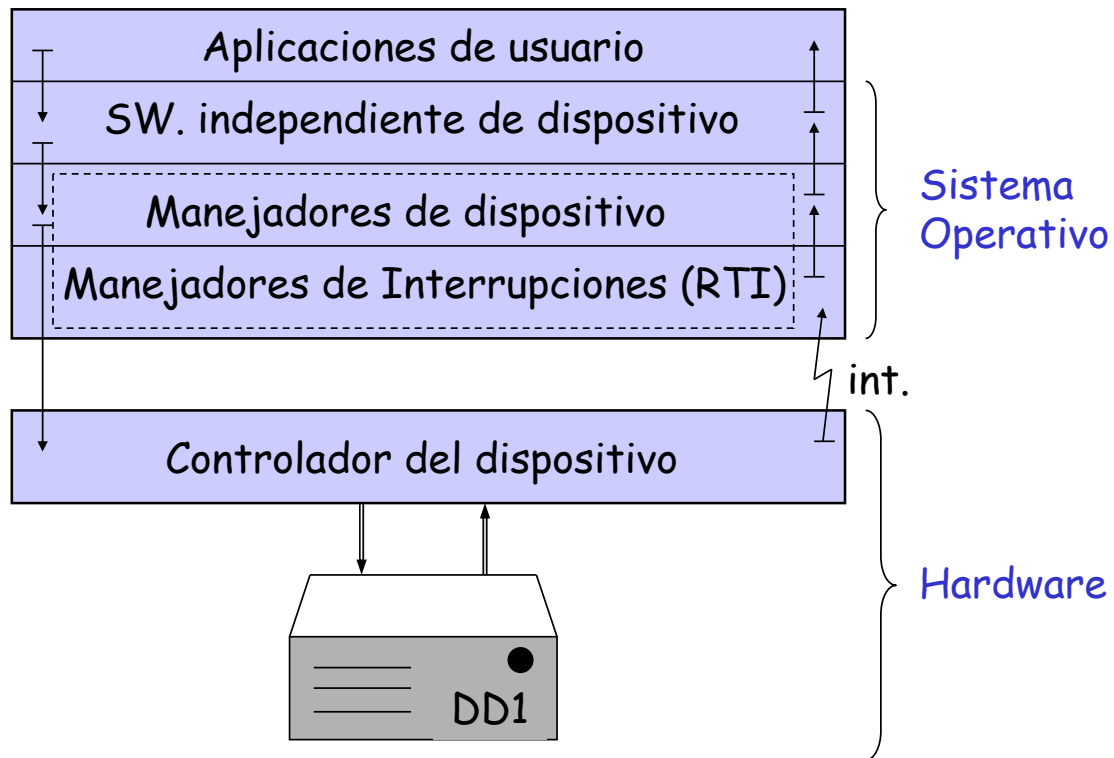
D₀-D₇. Son las señales de datos. Se utilizan para transferir datos entre la CPU y el registro interno indicado por las señales de direcciones.

L/E. Esta señal indica si la operación que se va a realizar sobre el registro seleccionado va a ser de lectura o de escritura. Estas señales pueden ayudar a los hilos de direcciones a seleccionar uno de los registros del controlador, ya que, a veces, algunos de ellos tienen la misma dirección y solamente se distinguen por el hecho de ser de lectura o de escritura.

CS. Esta señal debe activarse cuando el procesador vaya a leer o escribir un dato en uno de los registros del controlador.

Las líneas de comunicación con el dispositivo son de datos, de estado y de control, y varían dependiendo de cada dispositivo concreto.

...Estructura del Software de un Controlador



Los objetivos descritos anteriormente suelen lograrse **estructurando el software** de E/S en los siguientes niveles:

1. **Driver o manejador de dispositivo** (incluye las rutinas de tratamiento de interrupciones). Conoce exactamente el dispositivo que tiene asignado (es software dependiente del dispositivo). La función del *driver* es traducir las ordenes de alto nivel a los comandos de bajo nivel que entiende el controlador del dispositivo.
Es el encargado de darle ordenes, consultar su estado y atender sus interrupciones. Es el primer nivel en el que deben tratar de solucionarse los errores del dispositivo.
2. **Software del sistema operativo independiente de dispositivos**. En caso de que el *driver* no pueda enmascarar un error, se lo comunica al gestor de dispositivos del sistema operativo, que lo intentará por otros medios o mediante más reintentos. Estos gestores deben ser programas independientes de dispositivos concretos, que manejen dispositivos genéricos (discos, impresoras, pantallas, etc.) capaces de comunicarse con la interfaz, igualmente genérica, que le ofrezcan los *drivers* concretos de cada dispositivo.
3. **Aplicación de usuario**. Este nivel es el más abstracto, normalmente es de donde parten las ordenes (también parten del S.O.), y donde se reciben los resultados. Si es posible debe permanecer abstraído de los posibles problemas que puedan producirse en los dispositivos de entrada/salida.

Espacios de Direccionamiento de la CPU

Registros
Internos

Acceso local

`MOVE.B #1,D0`

Memoria
Principal

Acceso mediante
Buses de dirección,
datos y control

`MOVE.B #1,$7FFF`

Hasta el capítulo anterior, hemos visto que la CPU disponía de dos entornos donde podía leer o escribir datos: sus registros internos y la memoria principal, y para ello disponía de distintos métodos de direccionamiento que podía utilizar en las instrucciones para hacer referencia a registros o a posiciones de memoria. Esto quiere decir que disponía de dos espacios de direccionamiento, uno de registros internos y otro de posiciones de memoria.

Para acceder a sus registros internos, la CPU no necesita ningún hardware adicional, sin embargo, para acceder a la memoria principal cuenta con los buses de direcciones, datos y control. Ya sabemos cómo funciona este acceso: la CPU pone una dirección en el bus de direcciones para hacer referencia a una posición de memoria, en una señal de control indica si la operación es de lectura o escritura, y el dato correspondiente va/viene por el bus de datos hacia/desde la posición correspondiente de memoria principal.

¡ Ahora aparece un nuevo espacio de direccionamiento !

Espacio de E/S

¿ Cómo indica la CPU las direcciones de los puertos de E/S ?

- 1.- Espacio de direcciones propio de E/S
- 2.- Compartiendo espacio de direcciones de memoria
(Direcciones *mapeadas* en memoria)

Pues bien, ya hemos visto que ahora disponemos de otro entorno con el que necesita comunicarse la CPU: el conjunto de puertos contenidos en los dispositivos de E/S, y por lo tanto surge la pregunta **¿cómo indica la CPU que un dato va dirigido a un cierto puerto de un dispositivo concreto?**

Lo más inmediato es decir: “conectando los dispositivos de E/S a los buses de direcciones, datos y control”. Pero debemos darnos cuenta de que las direcciones que viajan por el bus de direcciones hacen referencia a posiciones de memoria, es decir, hasta ahora el bus de direcciones estaba pensado para hacer referencia al espacio de direccionamiento de la memoria principal, pero ahora nos ha aparecido un nuevo espacio de direccionamiento: el de los dispositivos de entrada/salida. Es decir, que lo que necesitamos es un nuevo medio para hacer referencia a los puertos de este nuevo espacio de direccionamiento.

En las siguientes transparencias veremos los dos métodos que hay para lograr esta comunicación con los puertos de E/S:

- 1) Disponer de un espacio de direccionamiento separado para los dispositivos de entrada/salida.
- 2) Reservar direcciones de memoria para los dispositivos de E/S. Es lo que se conoce como “direcciones de E/S *mapeadas* en memoria”.

Dos Opciones de Implementación

→ 1.- Nuevo conjunto de buses

→ 2.- Compartir buses de memoria
+ Decodificación de dir.
+ Señal de Control

Si había un espacio de direccionamiento para los registros internos de la CPU (R0, R1, R2, ...) y otro para las posiciones de memoria principal (0, 1, 2, 3, ...), al añadir un nuevo conjunto de dispositivos con los que, a través de sus puertos, también se va a comunicar el procesador, lo que parece razonable es añadir un nuevo espacio de direccionamiento: el **espacio de direccionamiento de los dispositivos de E/S**.

Hay dos posibilidades para implementar este nuevo espacio de direccionamiento. Una consiste en instalar un nuevo conjunto de buses de direcciones, datos y control. La otra, más simple, consiste simplemente en compartir todas las señales existentes en los buses, y simplemente hay que añadir una nueva señal de control que indica si la operación en curso está haciendo referencia al espacio de memoria o al espacio de dispositivos de E/S. Mediante las técnicas de decodificación de direcciones que ya hemos visto, y añadiendo esta nueva señal que indica el espacio de direccionamiento referenciado, resulta fácil de implementar el nuevo espacio de direccionamiento.

Los procesadores de Intel y los PowerPC disponen de este nuevo espacio de direccionamiento aunque, también pueden utilizar otra técnica, que pronto presentaremos, para direccionar los dispositivos “*mapeados*” en memoria.

Referencias a los Espacios de Direccionamiento

Registros: AX, BX, CX, IP, ... FR0-FR31, R0-R31

Direcciones: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D,

...

¿ Cómo se indica en una referencia a un puerto de E/S ?

Intel: Instrucciones explícitas de E/S

```
IN (AX, FFF4);
```

```
OUT (AX, FFF6);
```

PowerPC: Registro de control para conmutar de espacio de direccionamiento

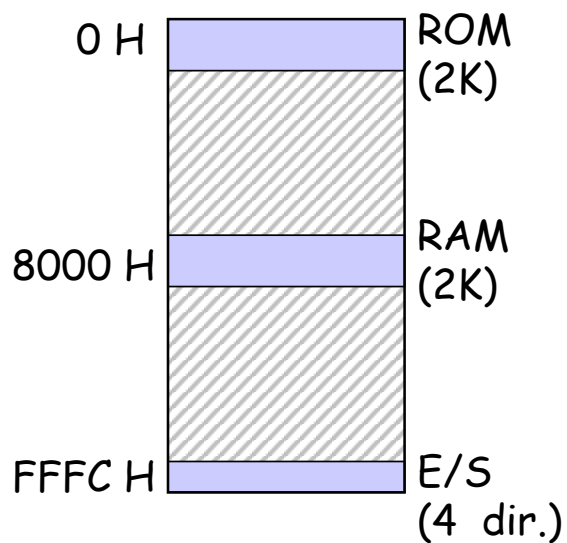
Hemos visto que en las instrucciones se referencian los registros del procesador mediante nombres predefinidos (A0-A7, D0-D7, PC, ... en Motorola; AX, BX, CX, IP, ... en Intel; FR0-FR31, R0-R31, ... en PowerPC), mientras que a las direcciones de memoria basta con indicar un número sin más para saber que se refiere a una posición de memoria. ¿Cómo indicamos ahora en una instrucción que hacemos referencia a un puerto de un dispositivo de E/S?

Para responder a esta pregunta, Intel dispone directamente de instrucciones explícitas de E/S (las instrucciones IN y OUT), que sirven para mover datos entre los puertos de los dispositivos de E/S y los registros de la CPU. Así, cuando se ejecute una de estas operaciones, se activará convenientemente la pata del procesador que indica que la dirección del bus de direcciones se refiere al espacio de direccionamiento de E/S.

PowerPC no ofrece instrucciones explícitas de E/S, pero dispone de un registro de control que permite conmutar de un espacio de direcciones a otro.

A la técnica utilizada cuando el acceso a los dispositivos de E/S se realiza mediante un espacio de direccionamiento separado y se utilizan instrucciones distintas para acceder a los puertos de los dispositivos, también se la conoce con el nombre de **Entrada/Salida explícita**.

Espacio de Direcciones de E/S Mapeadas en Memoria



Se comparte el espacio de direcciones de memoria con las de los dispositivos de E/S

Dirs. E/S \ll Espacio total

```
MOVE .B D0 , $FFFC
```

```
MOV $FFFE , AX
```

Direccionamiento de E/S mapeado en memoria principal.

La otra opción para acceder al nuevo espacio de direcciones de los dispositivos de E/S es compartir un único espacio de direccionamiento entre la memoria principal y los dispositivos de E/S.

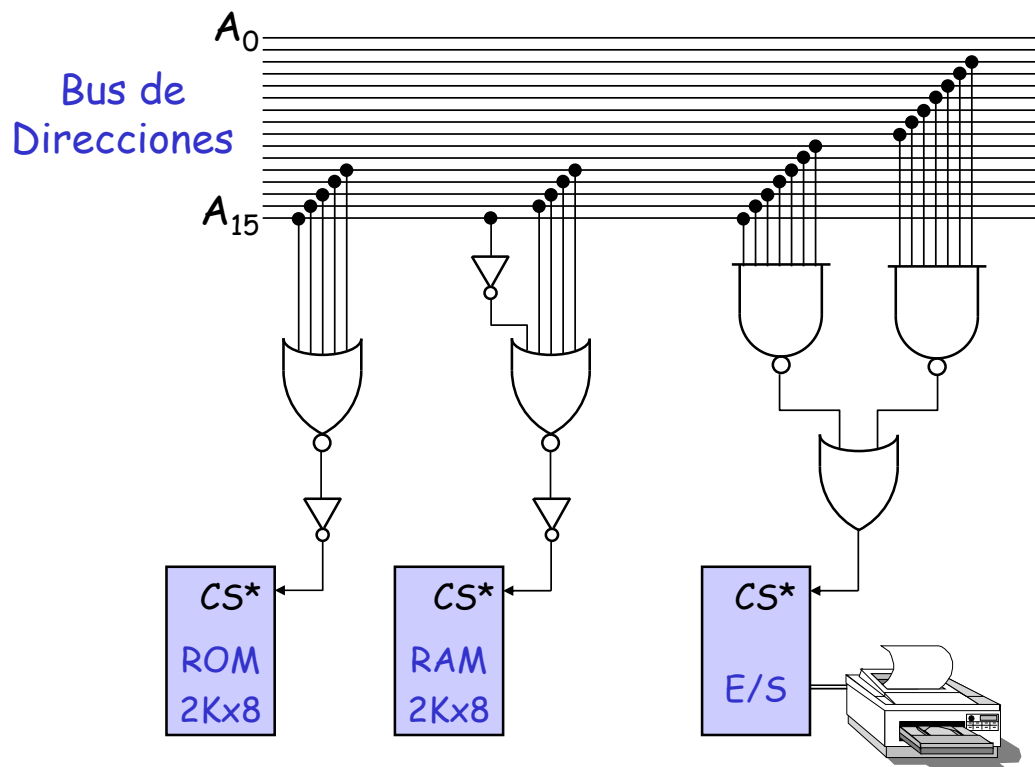
Se trata de disponer de un único espacio de direcciones (el que antes era solamente de la memoria principal), y repartir las direcciones entre posiciones de memoria y puertos de los dispositivos de E/S. En el capítulo anterior ya vimos que hoy día el espacio de direccionamiento no se suele corresponder con una secuencia continua de posiciones de memoria del mismo tipo, sino que el mapa de memoria suele estar compuesto por zonas de memoria RAM, zonas de memoria ROM y por grandes zonas de direcciones que están libres, es decir, que no se corresponden con chips de memoria. Pues bien, aprovechando estos vacíos, y el hecho de que normalmente el conjunto de direcciones requerido por todos los dispositivos de E/S es insignificante frente al vasto espacio de direccionamiento que se suele ofrecer, se trata de reservar en el mapa del espacio de direccionamiento un conjunto de direcciones que se refieren a los puertos de los dispositivos de E/S. Así, simplemente mediante las técnicas de decodificación de direcciones se puede hacer referencia a memoria RAM, ROM, o a dispositivos de E/S.

Esta solución es simple, y no requiere ninguna modificación de la CPU tal y como la hemos conocido hasta ahora, ni en su juego de instrucciones, ni en las señales o patas de que dispone. Para leer o escribir en los puertos de los periféricos simplemente hay que utilizar instrucciones convencionales del tipo `MOVE`.

El único inconveniente que se podría argumentar en contra de esta técnica es que cuanto mayor sea el número de dispositivos que se conecten al sistema, menor será el espacio disponible para memoria principal. Sin embargo, hoy día esta limitación es intrascendente, pues no tiene ninguna importancia frente a los espacios de direccionamiento que ofrecen los buses de direcciones de 32 bits o más (≥ 4 Gbytes) .

Esta técnica de acceso a los dispositivos de E/S es la utilizada por Motorola, aunque también suele utilizarse en arquitecturas basadas en Intel o PowerPC. Esto se debe al hecho de que disponer de un espacio independiente y direcciones explícitas para los dispositivos de E/S no impide que se comparta el espacio de direcciones de memoria con el de los dispositivos de E/S.

...Espacio de Direcciones de E/S Mapeadas en Memoria



En esta figura podemos ver cómo se conecta el controlador de un dispositivo de entrada al bus de direcciones para compartir el espacio de direccionamiento de la CPU con otros "dispositivos", como la memoria.

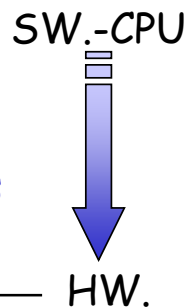
Métodos de Entrada/Salida

El envío/recepción de datos entre la CPU y un dispositivo de E/S se realiza en 2 fases

1º Sincronización CPU - Dispositivo
2º Transferencia del Dato

Estos 2 pasos pueden realizarse:

- ✓ Por *Polling*
- ✓ Por Interrupciones
- ✓ Por DMA



	<i>Polling</i>	Interrupciones	DMA
Sincronización	SW-CPU	HW.	HW.
Transferencia	SW-CPU	SW-CPU	HW.

Una vez desarrollado el hardware que conecta los dispositivos de E/S con la CPU, se debe implementar el software que lo controla. El envío o recepción de un dato a/desde un dispositivo de E/S se debe realizar en dos pasos:

- 1) **Sincronización CPU–dispositivo.** El procesador principal y los dispositivos de E/S realizan tareas distintas y a velocidades muy distintas; por eso, deben sincronizarse antes de realizar la transferencia en cualquier sentido.

Aunque es obvio, no debemos olvidar que en una operación de lectura, no se puede leer el dato del dispositivo hasta que el dato esté en el dispositivo. Por ejemplo, para recibir un carácter de una línea de comunicaciones primero hay que esperar a que el carácter llegue por la línea hasta una pastilla de comunicaciones (USART). De igual manera, en una operación de salida, no se puede enviar el dato al dispositivo si éste no está dispuesto o preparado para recibirlo, pues se perdería.

- 2) **Transferencia del dato.** Una vez que se ha conseguido la sincronización, ya se puede enviar el dato (en cualquier sentido).

Las funciones de estos dos pasos puede realizarse puramente por programa, o con ayuda de algún dispositivo hardware, lo que da lugar a tres métodos principales de entrada/salida:

- Por sondeo (*polling*)
- Por interrupciones
- Por DMA (*Direct Memory Access*)

Generalmente, a medida que se va encargando el hardware de realizar los dos pasos de la E/S, se obtiene mayor eficiencia.

En las siguientes transparencias trataremos cada uno de estos métodos de E/S.

Lectura de Teclado y Eco por Pantalla

```
repeat
  repeat
    Dato = Leer(Reg_Datos_Teclado);
  until Dato /= 0;
  repeat
    Estado := Leer(Reg_Estado_Pantalla);
  until Estado = Preparado;
  Escribir (Dato, Reg_Datos_Pantalla);
until false;
```

Esta es la forma más sencilla de realizar operaciones de E/S. Con este método, el sincronismo entre la CPU y el dispositivo de E/S se consigue preguntando la CPU directamente al dispositivo si tiene algún dato o si está listo para recibirlo.

Aquí arriba tenemos como ejemplo el fragmento de un programa que se dedica a leer caracteres del teclado y a hacer eco por pantalla mediante sondeo o, como más vulgarmente se le conoce, "*polling*".

Un poco más adelante podremos ver la secuencia de pasos que se realizan para llevar a cabo una operación de escritura en un dispositivo mediante este sistema.

Lectura de Teclado y Eco por Pantalla

Con instrucciones explícitas de E/S

```
DATOS_T    EQU    0
DATOS_P    EQU    1
ESTADO_P   EQU    2

LEER       IN     AX, DATOS_T
           JZ     LEER
OCUPADA    IN     BX, ESTADO_P
           JZ     OCUPADA
           OUT   AX, DATOS_P
           JMP   LEER
```

Vamos a mostrar dos ejemplos de código en ensamblador correspondiente al fragmento del programa anterior que se dedica a leer caracteres del teclado y escribirlos en la pantalla.

En el ejemplo superior se muestra el código para un procesador de Intel. En este caso se han aprovechado las instrucciones explícitas de E/S que ofrecen las arquitecturas de la familia 80x86 y Pentium de Intel. (Este código no es estrictamente correcto, pues las instrucciones `IN` y `OUT` siempre utilizan implícitamente el registro `AX` como registro de entrada/salida de datos).

Lectura de Teclado y Eco por Pantalla

Con Dir. de E/S mapeadas en Memoria

```
DATOS_T    EQU    $FFFF01
DATOS_P    EQU    $FFFF03
ESTADO_P   EQU    $FFFF05

LEER       MOVE.B  DATOS_T,D0
           BEQ     LEER

OCUPADA    TEST.B  ESTADO_P
           BEQ     OCUPADA
           MOVE.B  D0,DATOS_P
           BRA     LEER
```

En este ejemplo se muestra el código para una CPU 68000 de Motorola. Al no disponer de espacios de direccionamiento separado para E/S, esta solución utiliza las direcciones de los puertos de E/S *mapeados* en memoria.

CPU

1. Leer Registro de Estado del controlador
3. Comprobar estado.
Si no preparado, ir a Paso 1
4. Escribir dato en el reg. de datos del controlador de pant.
8. Si hay más datos para enviar, ir a Paso 1

Controlador

2. Enviar estado
5. Aceptar dato y poner estado a "No preparado"
6. Esperar a que el dato se escriba en el dispositivo
7. Poner estado a "Preparado"

PROBLEMAS

- ✓ Pérdida de Tiempo (Espera activa)
- ✓ Lentitud (si no se atiende pronto)

Aquí se muestra la secuencia de los pasos que se realizan para llevar a cabo una operación de escritura en un dispositivo de salida (como una pantalla) mediante *polling*.

Los pasos 1 a 3 forman el bucle de sincronismo, en el que la CPU le pregunta al controlador del dispositivo por su estado. Cuando el controlador está preparado para recibir un nuevo dato lo indica en su estado, y en el paso 4 la CPU envía el dato al puerto de datos del controlador. Cuando éste recibe el dato en el paso 5, establece su estado a "no preparado" hasta que haya enviado el dato al dispositivo y esté preparado para recibir un dato más.

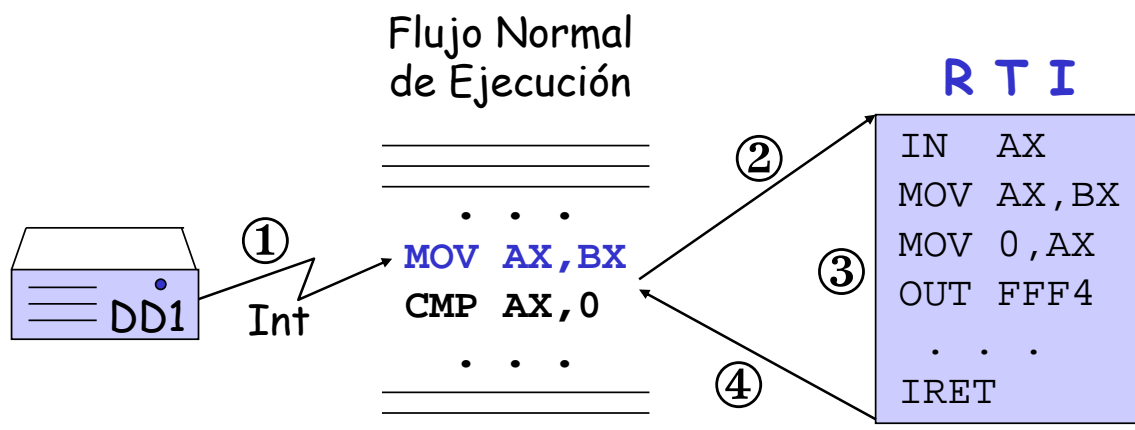
En este sistema, cada uno de los pasos que hay que realizar está programado mediante una o más instrucciones de entrada/salida, por eso a este método también se le conoce con el nombre de **entrada/salida programada**.

Este sencillo método es fácilmente realizable, pero presenta algunos problemas que no siempre permiten utilizarlo. Veamos cuáles son estos inconvenientes:

Pérdida de Tiempo. Cuando el número real de operaciones de lectura que se van a realizar con un dispositivo no es elevado, la mayoría de las veces que se le pregunte si tiene datos disponibles, contestará negativamente (Bucle en pasos 1 a 3). Esto quiere decir que se pierde tiempo en preguntar para no conseguir ningún dato a cambio. Esta situación se agrava si son muchos los periféricos a consultar o cuando las tareas que está realizando el procesador son complejas y deben ejecutarse en un tiempo crítico.

A este problema también se le conoce como "**el problema de la espera activa**", esto es, que la forma que tiene la CPU de esperar a que el dispositivo esté preparado es preguntándolo ella misma mediante instrucciones, y lo realiza en un bucle del que no sale hasta que el dispositivo está preparado. Esto impide que la CPU pueda hacer otra tarea mientras se espera a que el dispositivo esté listo.

Lentitud en la atención al dispositivo. Es evidente que mediante este procedimiento de sincronización, las transferencias de E/S solo son posibles en los momentos previstos por el programador (cuando el programa ejecuta las instrucciones de comprobación del estado del controlador). De esta manera, si un periférico requiere atención urgente, deberá esperar a que el programa llegue a la secuencia de instrucciones prevista para atenderle, lo cual puede llegar a producirse demasiado tarde (por ej., si viene un bloque de información por una línea de comunicaciones, se pueden perder caracteres; o más grave, si se trata de una alarma de sobrepresión en una central nuclear.)



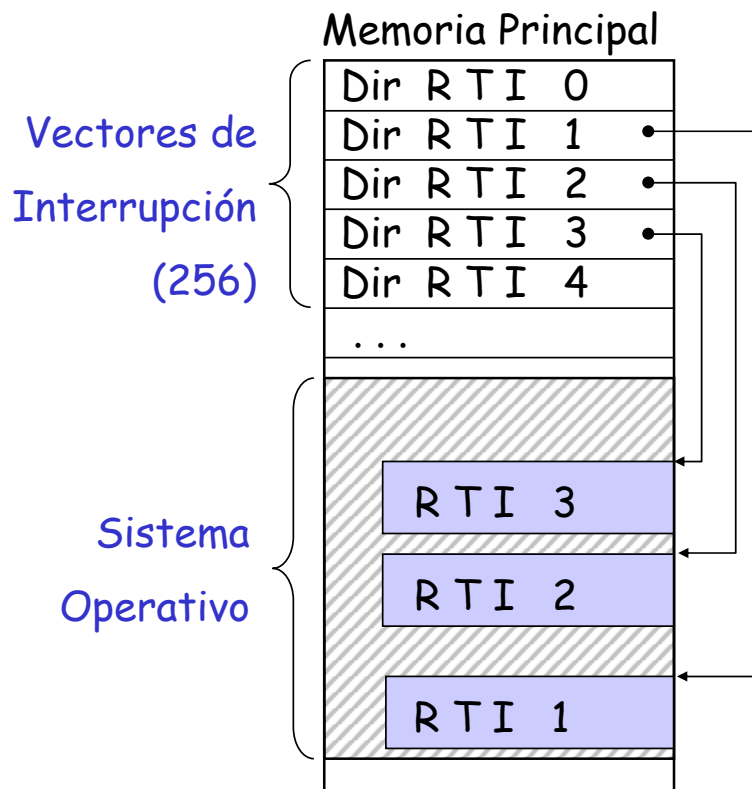
En el esquema de E/S realizado por *polling* vemos que para consultar el estado del dispositivo se dispone de un bucle en el que repetidamente se comprueba el estado. Durante el tiempo que se ejecuta este bucle, el procesador no realiza ninguna labor productiva o útil.

En muchas situaciones, se podrían realizar otras tareas mientras se espera a que el dispositivo esté preparado. Para hacer esto, vendría bien que, de alguna manera, el propio dispositivo avisara al procesador cuando esté preparado, permitiendo así que durante la espera, la CPU realice otro trabajo útil. Este aviso se consigue enviando desde el dispositivo a la CPU una señal hardware denominada "interrupción". Vamos a tratar de describir qué es una interrupción.

Hasta ahora sabemos que las instrucciones se ejecutan en la CPU secuencialmente por su posición en memoria, y la única forma que conocemos de variar este flujo secuencial es mediante instrucciones de bifurcación, llamadas a procedimientos y retornos de subrutinas. Otra forma de alterar esta secuencialidad es mediante interrupciones. Una **interrupción** es un suceso, más o menos esperado pero que no se conoce el momento exacto en que se va a producir. La interrupción se produce bien como consecuencia de un evento externo a la CPU (por los dispositivos de E/S), o bien por la propia CPU como consecuencia de la ejecución o intento de ejecución de una instrucción. Por cualquiera de estos motivos, una interrupción debe entenderse como un suceso que se produce "por sorpresa", pero que hay que tratarlo inmediatamente.

Cuando se produce una interrupción, lo normal es

- 1º) abandonar el flujo secuencial de ejecución actual,
- 2º) dar control a una Rutina de Tratamiento de la Interrupción (RTI) producida,
- 3º) ejecutar la RTI y, por último,
- 4º) reanudar el flujo normal de ejecución en el punto donde se interrumpió.



Puesto que hay diversos motivos que pueden generar una interrupción, debe haber una rutina de tratamiento para cada tipo o motivo de interrupción posible. La cuestión es “cuando se avisa a la CPU de que se ha producido una interrupción ¿cómo sabe la CPU cuál es la dirección de su rutina de tratamiento?”.

Una posibilidad podría ser tener siempre cada rutina de tratamiento en una dirección fija de memoria y, dependiendo de la interrupción producida, la CPU saltaría directamente a esta dirección prefijada. Pero ¿dónde las ponemos? ¿una a continuación de otra? ¿cuánto ocupa cada una?

Teniendo en cuenta que las RTI forman parte del sistema operativo instalado, esta opción forzaría mucho su construcción, pues en cada caso ocupará un tamaño distinto y en distinta dirección.

En su lugar, lo que se suele hacer es saltar a las rutinas de tratamiento mediante indirección. Normalmente, en la parte baja de la memoria se dispone de una **Tabla de Vectores de Interrupción**, denominada así porque cada elemento o entrada de la tabla contiene un Vector de Interrupción o, lo que es lo mismo, la dirección de una rutina de tratamiento de interrupción, de tal manera que el primer elemento de esta tabla contiene la dirección de la rutina de tratamiento de la interrupción 0; la segunda entrada, la dirección de la RTI 1, y así sucesivamente. (En Motorola, esta tabla a la que también se la conoce como **Vector de Interrupciones**, está en la dirección 0; en el Pentium, la dirección es configurable; y en el PowerPC se puede optar entre 1F4 y FFF001F4 como direcciones de comienzo).

El sistema operativo debe encargarse, en su proceso de inicialización, de establecer las direcciones de sus rutinas de tratamiento de interrupciones en los vectores de interrupción correspondientes.

Tipos de Interrupciones

☞ Externas (asíncronas)

☞ Internas (síncronas)

- Anomalías en instrucciones

- Instr. ilegal
- Bus error
- Div. por cero
- Overflow

- Interrupciones Software

INT 7

TRAP 7

Interrupciones
Excepciones
Traps

Ya hemos dicho que una interrupción es un evento que se produce de una forma más o menos inesperada. Vamos a ver aquí los distintos tipos de interrupciones que se pueden producir atendiendo a su fuente u origen.

Teniendo en cuenta que una interrupción la puede producir un dispositivo externo a la CPU o la ejecución de una instrucción en la propia CPU, nos encontramos con los siguientes tipos de interrupciones.

Las **interrupciones externas** las producen los dispositivos de E/S o sensores que son periféricos de la CPU. Estas interrupciones **son asíncronas**, pues nunca se sabe con exactitud cuándo van a producirse.

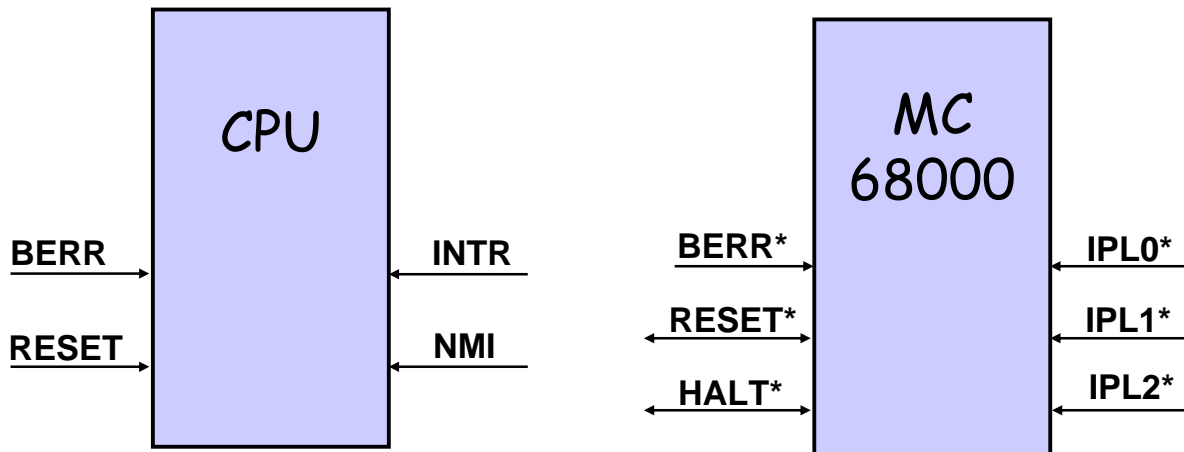
Las **interrupciones internas** se pueden producir por dos motivos:

- En la ejecución de una instrucción se detecta alguna anomalía, como por ejemplo, que el código de operación es desconocido, que se intenta dividir por cero, o que se ha realizado una operación aritmética que ha producido *overflow*. Cada uno de estos motivos genera una interrupción distinta, y el momento exacto en que se produce depende de cada instrucción; en el caso de un código de operación inexistente, la interrupción se produce mientras se decodifica la instrucción, en el caso de un intento de división por cero, la interrupción se produce al comprobar que el divisor es cero; y en el caso del *overflow*, la interrupción se produce al finalizar la operación.
- Interrupciones simuladas o interrupciones software. Son interrupciones generadas por instrucciones del lenguaje máquina incluidas en el código de un programa, cuyo cometido es única y simplemente simular una interrupción. Cuando se ejecuta una de estas instrucciones de interrupción software, el tratamiento que se genera es exactamente igual al que habría si esa misma interrupción la hubiera generado cualquier dispositivo externo. Por ejemplo, la instrucción `INT 7` de Intel hace que se ejecute la rutina de tratamiento de la interrupción 7; una vez ejecutada dicha rutina, se continuará con la instrucción que le sigue en secuencia a `INT 7`.

Las interrupciones software se suelen utilizar para comunicar las aplicaciones de usuario con el sistema operativo (*traps*) y para comprobar el tratamiento de las interrupciones que generan los dispositivos periféricos (o las internas de la CPU).

Las interrupciones internas **son síncronas**, pues la producción de una de estas interrupciones está asociada a una instrucción del propio programa, y no depende en absoluto de circunstancias externas al programa; por esto, si siempre se utilizan los mismos datos, al ejecutar el programa varias veces siempre se producen las mismas interrupciones (si las hay) en los mismos momentos (relativos al comienzo de ejecución del programa).

Algunos fabricantes de procesadores, como Motorola, denominan "excepciones" a todos los tipos de interrupciones, y a las producidas internamente a la CPU las llama *traps*. PowerPC también llama "excepciones" a las interrupciones en general. Intel denomina "interrupciones" a las originadas externamente a la CPU, y "excepciones" a las producidas como consecuencia de la ejecución de alguna instrucción.

¿Cuántas patas hay para comunicarle interrupciones a la CPU?

Ya sabemos que en un sistema hay múltiples tipos de interrupciones externas, las cuales se le comunican a la CPU mediante una señal hardware, o sea, una pata del procesador. Esto sugiere varias preguntas:

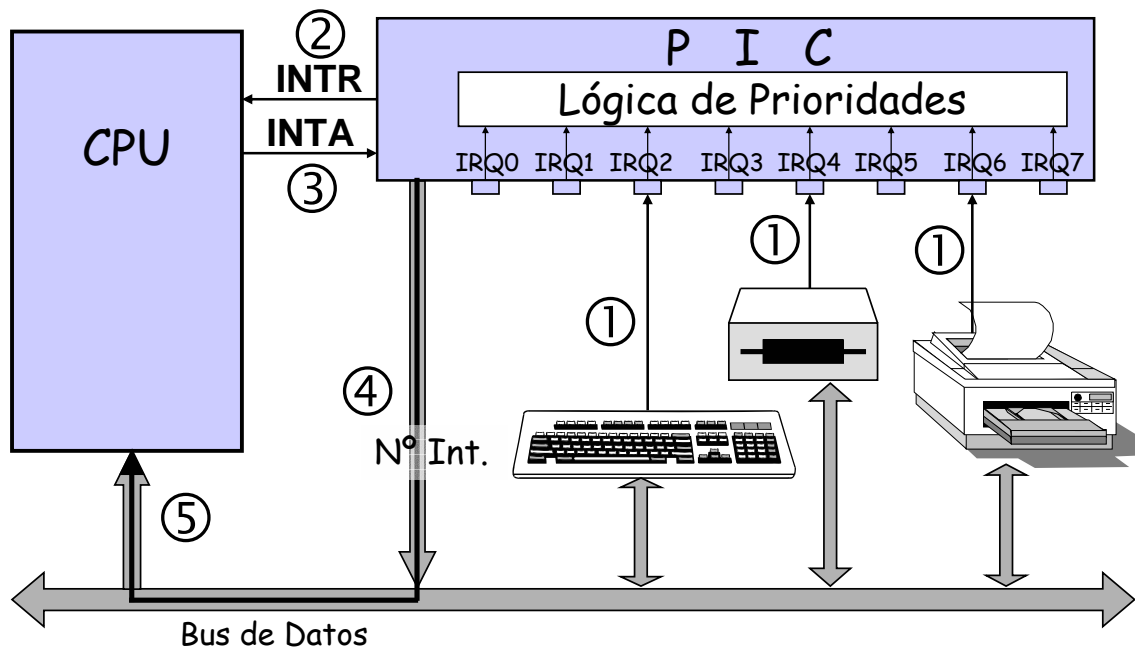
¿Cuántas patas hay para comunicarle interrupciones a la CPU?

El procesador podría disponer de una pata distinta para cada tipo de interrupción, pero dado que, por ejemplo, Intel y Motorola tienen hasta 256 interrupciones, no parece conveniente dedicarle solamente a esto 256 patas en el procesador. En su lugar, lo que hay es una pata (INTR) que avisa de que se ha producido una interrupción, y posteriormente la CPU se encarga de averiguar la interrupción producida. Hay algunas interrupciones especiales, como RESET, Bus Error, o la NMI (*non Maskable Interrupt*), que disponen directamente de una pata en la CPU para cada una de ellas. Ya veremos más adelante el proceso para averiguar el nivel de la interrupción producida cuando se activa la pata INTR.

El Motorola 68000 dispone de tres patas para señalar las interrupciones generales, pero las utiliza para indicar la prioridad de la interrupción producida, no para señalar interrupciones distintas.

En Intel también hay distintas prioridades para las interrupciones, pero la gestión de las prioridades se realiza en un dispositivo externo a la CPU, denominado PIC (*Programmable Interrupt Controller*).

¿Cómo se atienden varias interrupciones simultáneas?



Otra pregunta:

Si se producen simultáneamente varias interrupciones ¿cuál se trata primero?

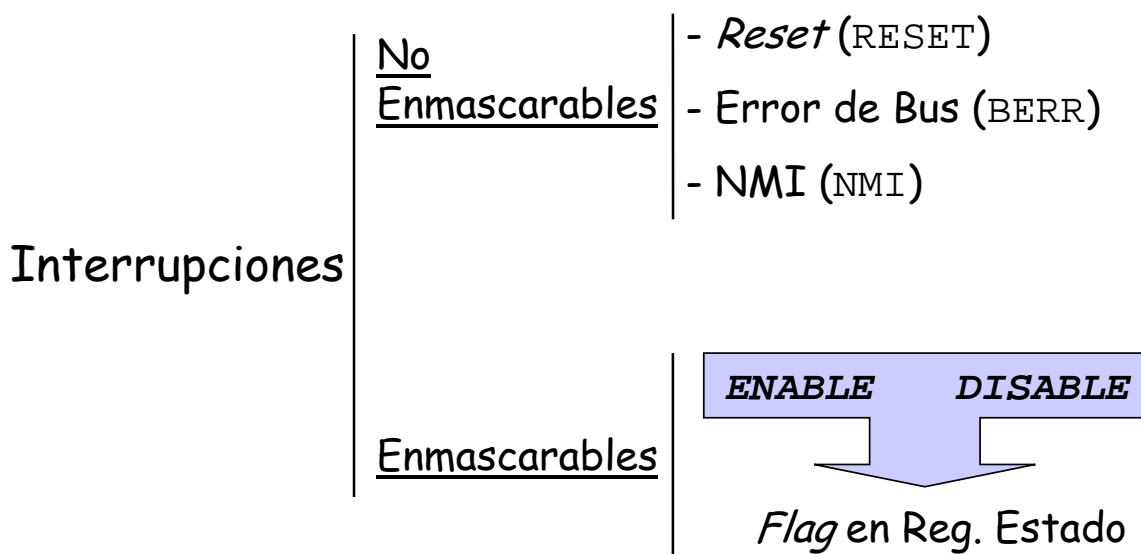
Este problema se puede resolver por dos vertientes, "por programa" o con ayuda del hardware.

Cuando se realiza "por programa", si varios dispositivos están conectados a la pata de interrupciones del procesador, al detectar éste una interrupción debe dar control a una rutina de tratamiento que vaya explorando todos los dispositivos conectados comprobando en sus registros de estado cuál ha sido el dispositivo que generó la interrupción. Esta exploración de tipo *polling* se realiza siguiendo un orden de prioridades establecido en la RTI.

Ya que la exploración de los dispositivos por programa consume mucho tiempo, a ser posible es preferible acudir al hardware, pues este problema se soluciona mediante una pastilla que se conecta entre los dispositivos y la CPU. Esta pastilla, que Intel denomina PIC (*Programmable Interrupt Controller*), por una parte admite 8 entradas de interrupciones (*IRQ*, *Interrupt Request*) que pueden venir de dispositivos de E/S o de otros PIC, y por otra se conecta a la pata o patas de interrupciones generales del procesador. Ya que el PIC es programable, se le establecen unas prioridades, de tal forma que ante la llegada de varias interrupciones simultáneas, se las va comunicando a la CPU por orden de prioridad. Como ya veremos con detalle, cuando la CPU recibe una petición de interrupción, le pregunta al PIC directamente por el número de la interrupción a tratar.

En el 68000, si no se utiliza un PIC, la gestión de las prioridades de distintos dispositivos, debe realizarse mediante un cableado con cierta complejidad.

¿Siempre hay que atender a las interrupciones?



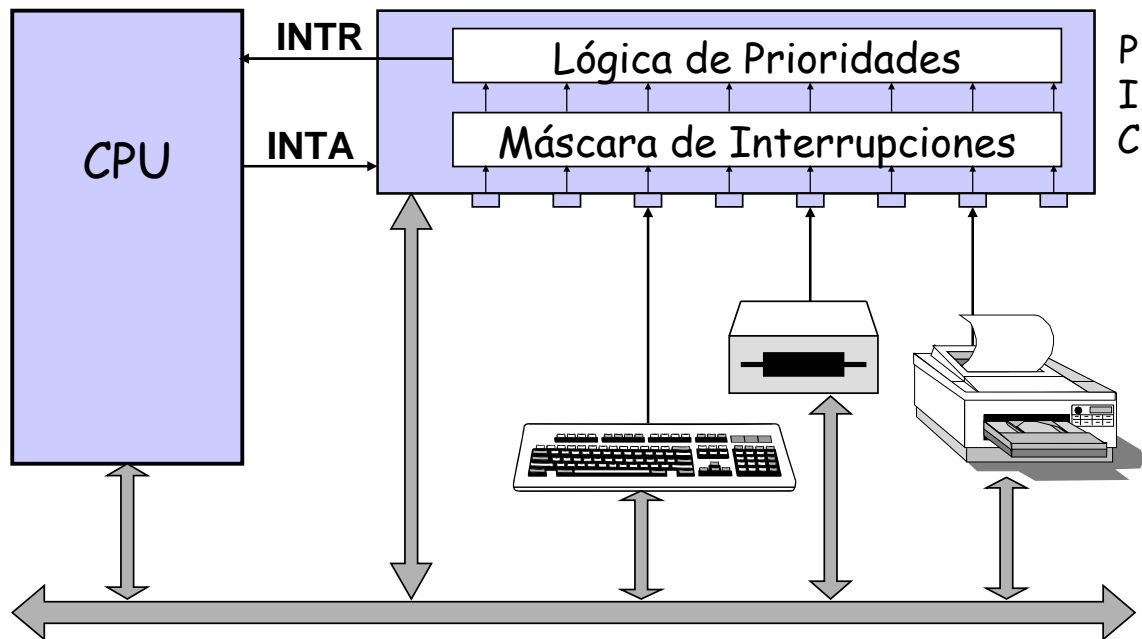
Continuamos con las preguntas.

¿Siempre hay que atender las interrupciones?

Según su importancia, hay dos tipos de interrupciones: enmascarables y no enmascarables. Las interrupciones **no enmascarables** son las que siempre se atienden. Se deben a motivos cuya atención no puede dilatarse en el tiempo. Por ejemplo, un *reset*, un error de bus de direcciones, o una interrupción no enmascarable (NMI), que puede deberse a motivos como una detección de fallo de tensión alimentación o la detección de un error en memoria RAM. Estas interrupciones suelen contar con patas específicas en la CPU (RESET, BERR, NMI), aunque Motorola utiliza la interrupción de nivel 7 como NMI.

Las interrupciones **enmascarables** se pueden atender o no, dependiendo de lo que esté indicado en el registro de estado. Intel cuenta con dos instrucciones, *ENABLE* y *DISABLE*, para permitir o inhibir la aceptación de interrupciones, que activan o desactivan el *flag* de aceptación de interrupciones del registro de estado.

En el Motorola, para inhibir la aceptación de interrupciones debe establecer el nivel de privilegio 7 en su registro de estado (*ORI #0700, SR*), con lo cual solamente aceptaría las interrupciones clasificadas como no enmascarables (las de nivel 7). Para permitir todas las interrupciones se debe ejecutar la instrucción *ANDI #F0FF, SR* la cual pone a cero los bits de prioridad (I_0, I_1, I_2) en el registro de estado.

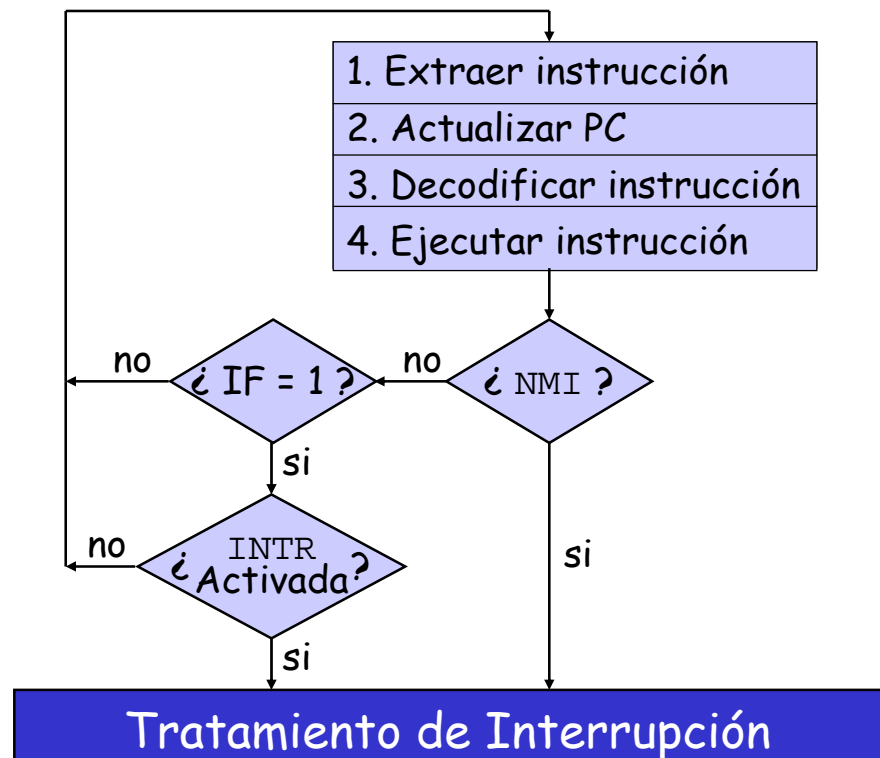
¿Se puede inhibir un nivel concreto de interrupciones?**¿Qué hacemos si en un momento dado no queremos atender un cierto tipo de interrupciones?**

Hasta ahora hemos visto que con las interrupciones enmascarables hay dos posibilidades: atenderlas o no atenderlas. Pero ¿qué pasa si en un momento dado se quieren atender las interrupciones de un periférico y no las de otro?

Esta situación se trata con elementos que ya conocemos. En el caso de Motorola, ya sabemos que cuenta con 7 niveles de privilegio de interrupción, con lo que, aparte de la no enmascarable, puede establecer 6 niveles de privilegio, de tal forma que puede realizar un enmascaramiento selectivo de interrupciones según las prioridades de los dispositivos que pueden interrumpir. Por ejemplo:

```
ORI    #$0300,SR    ;Permite las interrupciones
ANDI   #$F3FF,SR    ;de nivel > 3
```

Intel, por su parte, lo soluciona mediante el PIC, en el que se pueden establecer máscaras de interrupciones que indican selectivamente cuáles de las 8 interrupciones que puede tomar como entrada pueden interrumpir al procesador. Como ya hemos comentado, en el PIC, además de la máscara de interrupciones, también se programan las prioridades con las que se deben atender múltiples interrupciones simultáneas.

Aceptación

Vamos a describir aquí el proceso genérico de atención y tratamiento de interrupciones. Más adelante comentaremos el tratamiento específico de Motorola.

El procesador no acepta interrupciones mientras está ejecutando una instrucción (excepto `RESET` y `BUS ERROR`), y únicamente cuando ha finalizado su ejecución, y antes de alimentar la siguiente instrucción, comprueba si hay una interrupción NMI (no enmascarable). Si la hay, se pasa al proceso de tratamiento correspondiente, si no se ha producido una NMI, se comprueba en el registro de estado si tiene las interrupciones permitidas o inhibidas. Si están inhibidas, simplemente continúa alimentando y ejecutando instrucciones. Si están permitidas, comprueba si está activada la pata de interrupciones generales `INTR` (*Interrupt Request*), y si es así, comienza el proceso de atención o tratamiento a la interrupción, que lo tratamos con detalle en la página siguiente.

Tratamiento

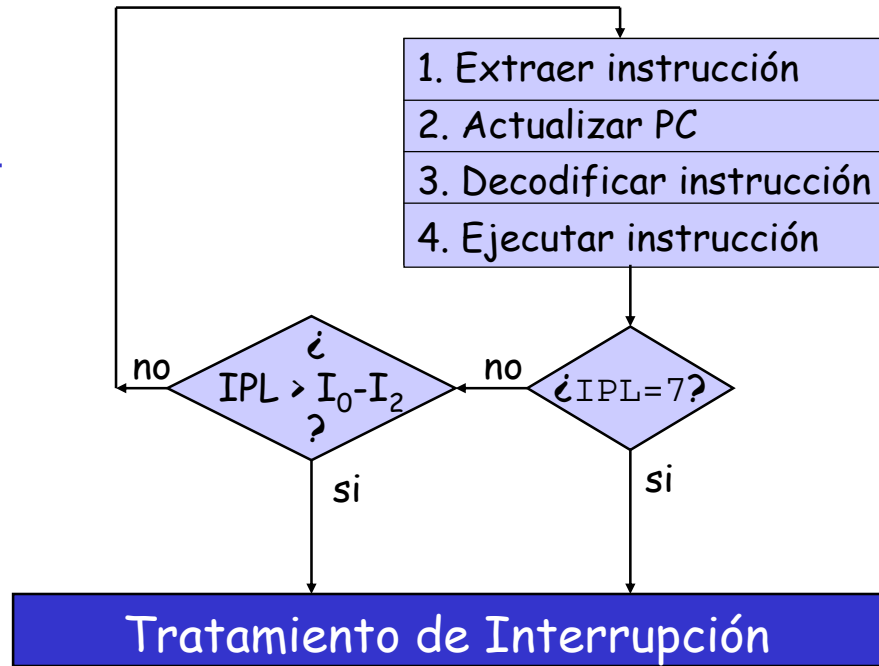
1. **SR** y **PC** → Pila
2. Inhibir interrupciones (**IF** = 0)
3. Activar **INTA**
4. El PIC desactiva **INTR**
5. El PIC pone el n° de vector de la int. en bus de datos
6. La CPU lee del bus de datos el n° del vector de int.
7. La CPU calcula la dir. del vector de interrupción
8. Vector de int. → **PC**
9. Toma control la RTI correspondiente
10. Al finalizar la RTI, se ejecuta una instr. **IRET**
 - **POP PC**
 - **POP SR** ⇒ Interrupciones permitidas
11. Continúa el programa interrumpido

Veamos con detalle el proceso de tratamiento o atención a la interrupción:

1. Se meten en la pila el Registro de Estado y el Contador de Programa.
2. Se inhiben las interrupciones.
3. Se contesta con la señal **INTA** (*Interrupt Acknowledge*).
4. El dispositivo (o el PIC) desactiva la señal de interrupción.
5. El PIC pone en el bus de datos el n° del vector de la interrupción que se quiere tratar.
6. La CPU lee del bus de datos el n° del vector *V* de la interrupción a tratar.
7. El procesador debe calcular la dirección del vector de interrupción que contiene la dirección de la rutina de tratamiento correspondiente, la cual se encuentra en la Tabla de Vectores de Interrupción. Si esta tabla está en la dirección 0 de memoria, y cada vector de interrupción ocupa 32 bits (4 bytes), simplemente debe multiplicar el número de interrupción *V* por 4, y el resultado es la dirección del vector.
8. La CPU pone el contenido del vector de interrupción (esto es, la dirección de la RTI correspondiente) en el Contador de Programa.
9. Toma control la RTI correspondiente al nivel de la interrupción atendida.
10. Al finalizar la rutina de tratamiento, su última instrucción es del tipo "Retorno de Interrupción" (**IRET**, **RTE**, ...). Esta instrucción saca de la pila los valores del Contador de Programa y del Registro de Estado (que se metieron al aceptar la interrupción la CPU) y los pone en estos registros, con lo que las interrupciones vuelven a estar permitidas.
11. Continúa la ejecución del programa que se estaba ejecutando cuando se produjo la interrupción.

El tratamiento de las interrupciones **RESET** o **BUS ERROR** es particular para cada una de ellas. El tratamiento de las interrupciones **NMI** es similar al descrito para las enmascarables, con la diferencia de que se atienden aún estando la CPU con las interrupciones inhibidas, y de que, por su gravedad, no suelen devolver el control al programa interrumpido, sino que finaliza la ejecución de los programas en el ordenador.

Aceptación en el MC68000



El tratamiento de las interrupciones “**vectorizadas**” en los procesadores de la familia 68000 es similar al descrito en la transparencia anterior, con las siguientes diferencias.

El 68000 no tiene un *flag* de inhibición de interrupciones, sino un nivel de aceptación de interrupciones, compuesto por los *flags* I_0-I_2 del registro de estado. Ya vimos que el 68000 tiene 3 patas de interrupción IPL_0-IPL_2 . Cuando el nivel de la interrupción es superior al nivel de aceptación de interrupciones, se atiende. El nivel 7 equivale a tener las interrupciones inhibidas, pues solamente se aceptan las no enmascarables (las de nivel 7).

En cualquier caso, cuando se atiende una interrupción, se procede según el tratamiento que se describe en la página siguiente:

Tratamiento en el MC68000

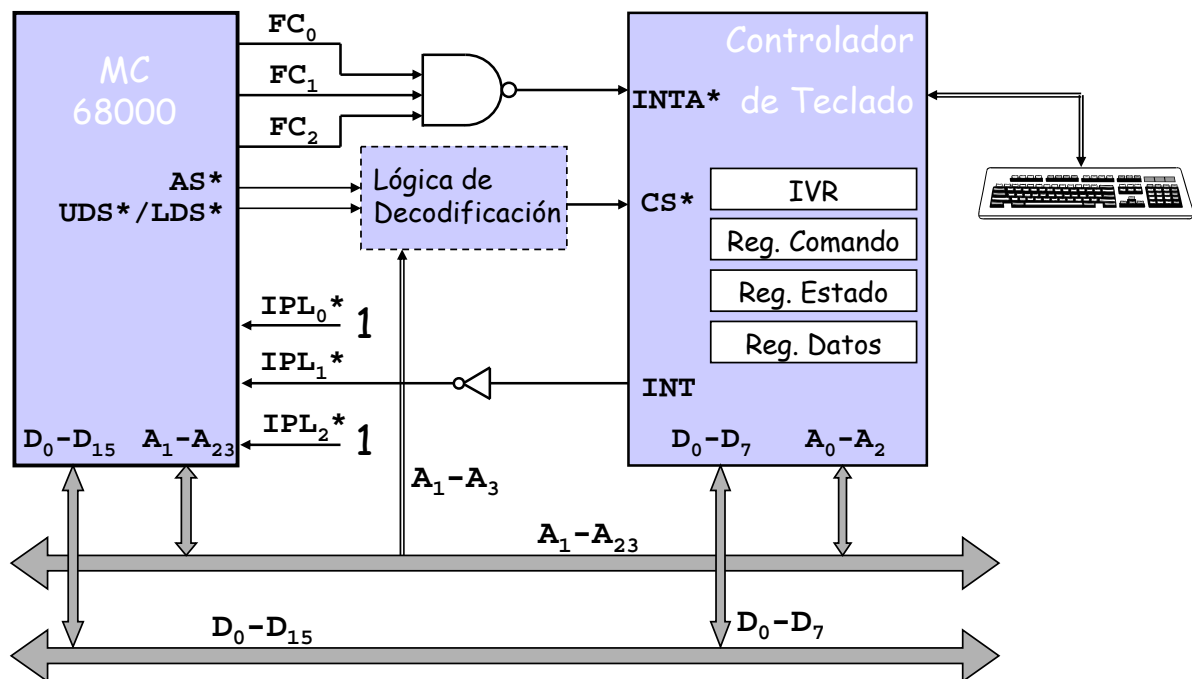
1. SR → Registro temporal
2. Inhibir interrupciones ($IPL_0 - IPL_2 \rightarrow I_0 - I_2$)
Paso a Modo Supervisor
3. SR temporal y PC → Pila (SSP)
4. Atender interrupción ($111 \rightarrow FC_0 - FC_2$)
 $IPL_0 - IPL_2 \rightarrow A_1 - A_3$
5. El dispositivo de E/S desactiva $IPL_0 - IPL_2$
6. El dispositivo de E/S pone el n° de vector de la int. en bus de datos ($D_0 - D_7$)
7. La CPU lee del bus de datos el n° del vector de int.
8. La CPU calcula la dir. del vector de interrupción
9. Vector de int. → PC
10. Toma control la RTI correspondiente
11. Al finalizar la RTI, se ejecuta una instr. RTE
 - POP PC
 - POP SR ⇒ Interrupciones permitidas
12. Continúa el programa interrumpido

Veamos el tratamiento que recibe una interrupción aceptada en el Motorola 68000:

1. Se guarda el Registro de Estado en un registro temporal.
2. Se establece el nivel de la interrupción aceptada como nuevo nivel de aceptación de interrupciones, y se pone la CPU en modo Supervisor.
3. Se meten en la pila el registro temporal y el Contador de Programa.
4. Se indica que se atiende la interrupción mediante las patas de estado FC_0 , FC_1 y $FC_2 = 111$. Además se indica el nivel de la interrupción aceptada mediante las patas $A_1 - A_3$.
5. El dispositivo (o el PIC) desactiva la señal de interrupción y 6) pone en el bus de datos ($D_0 - D_7$) el número del vector de la interrupción que se quiere tratar.
7. La CPU lee del bus de datos el número del vector V de la interrupción a tratar.
8. El procesador calcula la dirección del vector de interrupción que contiene la dirección de la rutina de tratamiento correspondiente, simplemente debe multiplicar el número de vector V por 4, y el resultado es la dirección del vector.
9. La CPU pone el contenido del vector de interrupción (esto es, la dirección de la RTI correspondiente) en el Contador de Programa.
10. Toma control la RTI correspondiente al nivel de la interrupción atendida.
11. Al finalizar la rutina de tratamiento, su última instrucción es RTE (*Return from Exception*). Esta instrucción saca de la pila los valores del Contador de Programa y del Registro de Estado (que se metieron al aceptar la interrupción la CPU) y los pone en estos registros.
12. Continúa la ejecución del programa que se estaba ejecutando cuando se produjo la interrupción.

Una interrupción es “**autovectorizada**” si en el paso 6, el dispositivo activa la pata VPA (*Valid Peripheral Address*). En este caso, la prioridad de la interrupción indicada por las patas $IPL_0 - IPL_2$ (1 a 7) indica uno de los vectores de interrupción reservados, 25 a 31 (decimal). Así, a las interrupciones de prioridad 1 les corresponde el vector 25, a las de prioridad 2, el vector 26, etc.

Este tratamiento de las interrupciones se realiza solamente con periféricos de la familia 6800, predecesora de la 68000.



Aquí tenemos el esquema simplificado de un sistema de lectura de teclado desde un procesador Motorola 68000 que utiliza la técnica de las interrupciones.

En este caso, se utiliza el sistema de interrupciones vectorizadas. Para ello se utiliza la interrupción de nivel 2 (prioridad 2), para lo cual se activa la pata IPL_1^* . Esta pata se activa desde la señal INT del controlador del teclado.

Para reconocer la interrupción, la CPU pone a 111 las señales de control FC_0-FC_2 , con lo que se activa la señal $INTA$ del controlador de teclado. Para que el controlador entienda que el reconocimiento de la interrupción va dirigido a él, debe tener activada su señal CS , lo cual se consigue con una lógica de decodificación apropiada. (Aquí no se detalla la lógica de decodificación de direcciones utilizada para activar la señal CS del controlador del teclado, aunque, como cabe esperar, intervendrán las señales AS , UDS/LDS y algunas de direcciones).

Una vez reconocida la interrupción, el controlador de teclado debe indicarle a la CPU el número del vector de la interrupción que se debe utilizar para tratar dicha interrupción. Esto lo hace el controlador poniendo el número de vector (que contiene en su registro IVR , *Interrupt Vector Register*) en los 8 hilos de menor peso del bus de datos (D_0-D_7), de donde los lee la CPU.

Una vez que la CPU conoce el número del vector de interrupción, le cede el control a la rutina de tratamiento correspondiente, cuya dirección se encuentra en el vector indicado.

En la siguiente página podemos ver la programación asociada a este sistema de E/S del teclado.

En los chips de algunos dispositivos, la activación de su pata $INTA$ es suficiente para que el dispositivo entienda que ese reconocimiento de interrupción es para él, sin necesidad de activar también su pata CS (*Chip Select*).

```

VECTOR          EQU 64
DIR_VEC_INT     EQU VECTOR*4
INTS_SI        EQU $40
INTS_NO        EQU $0
NUEVO_SR       EQU $0100
RC             EQU $0D
REG_CONTROL_TECLADO EQU $FFFFFF1
REG_DATOS_TECLADO EQU $FFFFFF3
REG_VECTOR_TECLADO EQU $FFFFFF5
;Iniciación
...
MOVE.L #LEER, DIR_VEC_INT
MOVE.L #LINEA, PTR_L
MOVE.B #VECTOR, REG_VECTOR_TECLADO
MOVE.B #INTS_SI, REG_CONTROL_TECLADO
MOVE.L #PRINCIPAL, -(A7)
MOVE.W #NUEVO_SR, -(A7)
RTE
PRINCIPAL ;Programa principal
...
Leer ;Rutina de tratamiento de interrupción
...
LINEA      DS.B 128 ;Reserva 128 bytes de mem.
PTR_L     DS.L 1 ;Reserva 1 puntero en mem.

```

Veamos un ejemplo de cómo sería la rutina de tratamiento de interrupción, con su correspondiente inicialización, descrita en ensamblador del 68000.

Supongamos que durante la ejecución de un Programa Principal, se quieren leer las teclas pulsadas en el teclado e ir las metiendo en un buffer (LINEA), y cuando se pulse la tecla de Retorno de Carro (CR), se llama a una rutina (TRATAR_LINEA) para que trate el contenido del buffer.

El controlador del teclado tiene dos registros, REG_DATOS_TECLADO y REG_CONTROL_TECLADO. En el primero se puede leer un carácter cuando nos avisa de que se ha pulsado una tecla. El segundo se utiliza para indicarle al controlador si puede generar interrupciones o no.

Supondremos que el vector que entrega el controlador de teclado corresponde al primer vector libre para el usuario, es decir, el 64. Esto quiere decir que le corresponde el vector de la dirección 64x4, esto es, 256 ó FF en hexadecimal.

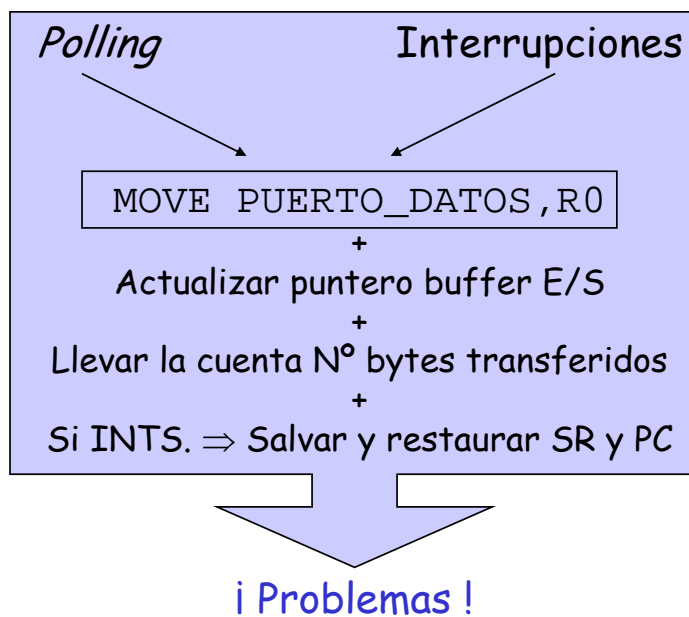
Antes de que el Programa Principal (un programa de usuario) tome control, el hardware debe estar debidamente programado (y debe hacerse en modo supervisor), para lo cual, formando parte del código que se ejecuta al arrancar la máquina, nos encontramos con el fragmento de programa con el comentario "Iniciación". Al finalizar el proceso de inicialización se le debe ceder el control al usuario (en modo usuario) en el Programa Principal.

En este fragmento de la inicialización se comienza por establecer en el vector de interrupción 64 la dirección de la RTI correspondiente (LEER). A continuación inicializa el puntero utilizado para ir metiendo caracteres en el buffer (PTR_L), y permite la producción de interrupciones en el teclado poniendo la máscara INTS_SI en el registro de control. La inicialización termina estableciendo el registro de estado y el contador de programa de la CPU. El SR se pone a \$0100, que indica el modo "usuario" y un nivel de prioridad 1. El contador de programa se establece con la dirección del programa principal. Obsérvese que los registros PC y SR adquieren realmente estos valores con la instrucción RTE.


Veamos en detalle, en la página siguiente, cómo sería la rutina de tratamiento de interrupciones.

;Declaración de constantes			
...			
;Inicialización			
...			
PRINCIPAL ;Programa principal			
...			
;Rutina de tratamiento de interrupción			
LEER	MOVEA.L	A0,-(A7)	
	MOVEA.L	PTR_L,A0	
	MOVE.B	REG_DATOS_TECLADO,(A0)+	
	MOVEA.L	A0,PTR_L	
	CMPI.B	#RC,-1(A0)	
	BNE	LEIDO	
	MOVE.B	#INTS_NO,REG_CONTROL_TECLADO	
	BSR	TRATAR_LINEA	
	MOVE.B	#INTS_SI,REG_CONTROL_TECLADO	
LEIDO	MOVEA.L	(A7)+,A0	
	RTE		
LINEA	DS.B	128	;Reserva 128 bytes de mem.
PTR_L	DS.L	1	;Reserva 1 puntero en mem.

La rutina de tratamiento de interrupción (LEER) comienza guardando el contenido del registro A0 en la pila, pues va a ser utilizado. A continuación lee el carácter del teclado y lo lleva al buffer de lectura, incrementando el puntero PTR_L que indica la primera posición libre del buffer. Después comprueba si el carácter leído es el Retorno de Carro. Si no lo es, salta a LEIDO, donde se restaura el registro A0 y finaliza la rutina de tratamiento de la interrupción. Si el carácter es un Retorno de Carro, inhibe la generación de interrupciones en el teclado y llama a la rutina TRATAR_LINEA. Al finalizar el tratamiento, se vuelven a permitir las interrupciones e, igualmente, se restaura el registro A0.




 Velocidad Máxima
 Limitada


 La CPU no puede
 dedicarse a otra tarea

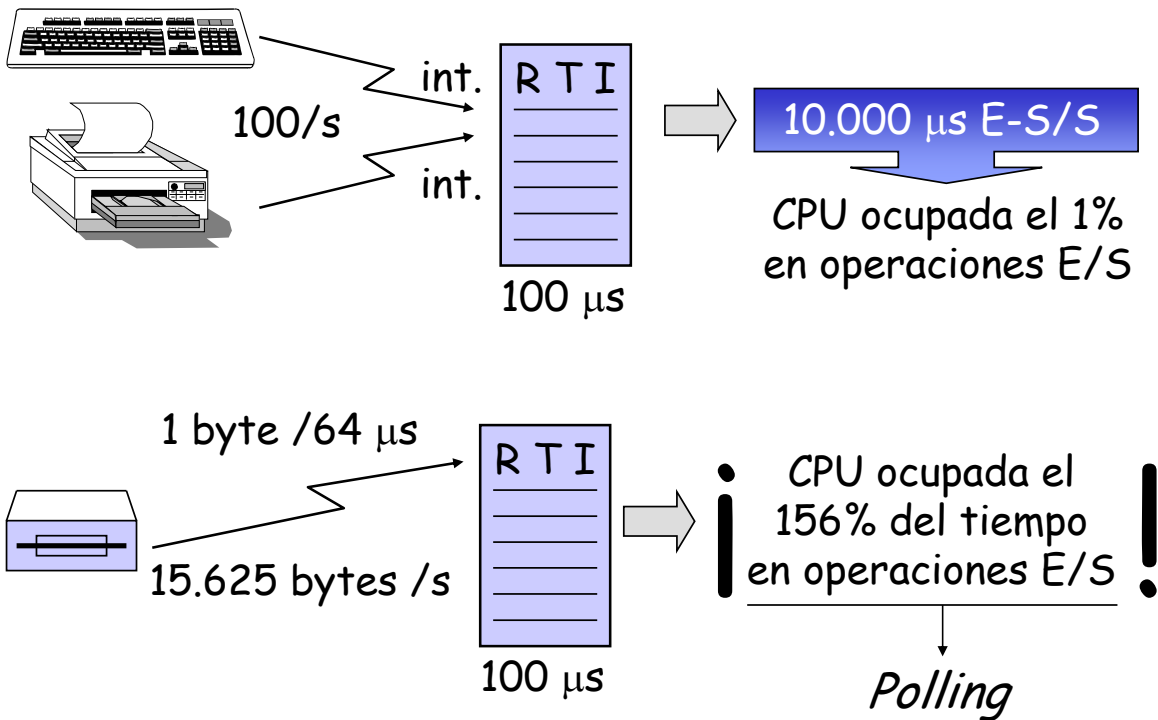
En los apartados anteriores hemos visto dos sistemas para realizar la E/S: por *polling* y por interrupciones, y en ambos casos la transferencia del dato se acababa realizando mediante una instrucción del tipo `MOVE PUERTO_DATOS, R0`

Esta instrucción se ejecutaba después de que la CPU comprobara que el dispositivo estaba “preparado”, para lo cual tenía que estar haciendo *polling* en el registro de estado del dispositivo, o esperar a que éste enviase una interrupción. Además de esto, también se necesitan instrucciones para ir actualizando los punteros que recorren el buffer de entrada o salida de datos, y llevar la cuenta del número de bytes que se envían o reciben. Cuando se utilizan interrupciones, hay una sobrecarga adicional, pues hay que salvar y restaurar, al menos, el contador de programa y el registro de estado.

Con estas premisas, nos encontramos con que estos dos métodos de E/S tienen estos dos serios inconvenientes:

1. La velocidad de transferencia está limitada por la velocidad a la que la CPU pueda realizar las operaciones citadas arriba (comprobar, mover, incrementar puntero, incrementar contador, salvar estado, restaurar estado, ...).
2. La CPU es la encargada de realizar el proceso de transferencia de datos, por lo que no puede dedicarse a otra tarea mientras haya octetos para enviar o recibir.

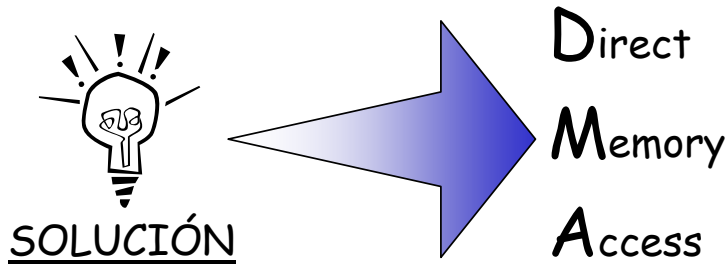
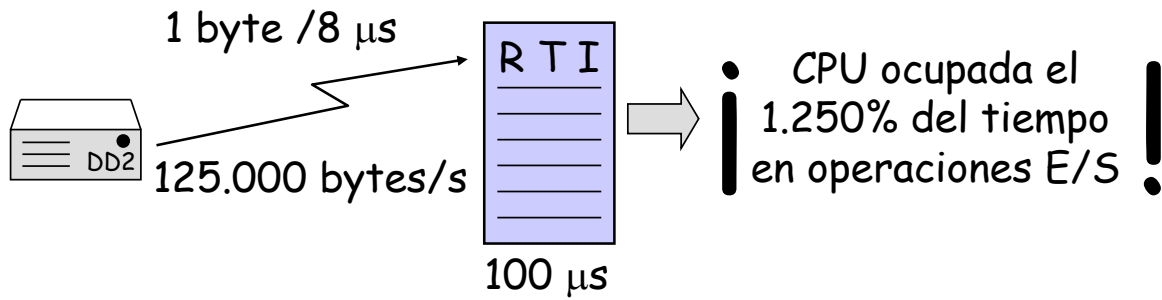
Veamos en la siguiente transparencia unos ejemplos que nos muestran cómo se materializan estos problemas cuando se utilizan dispositivos de E/S que manejan grandes volúmenes de datos y a gran velocidad, lo cual da pie a que sea necesario encontrar un sistema de E/S alternativo al *polling* y las interrupciones.



Es razonable que dispositivos relativamente lentos, como terminales e impresoras, interrumpan un programa cada vez que hay que transferir un byte. Por ejemplo, supongamos que se requieren 100 instrucciones, o 100 microsegundos, para atender cada interrupción de estos dispositivos lentos, incluyendo el salvado y recuperación de los registros, servicio de la interrupción y la gestión del buffer de E/S. Con una tasa de 100 transferencias (interrupciones) por segundo solamente se utilizaría el 1% del tiempo total de la CPU para ejecutar instrucciones, lo cual deja la gran parte del tiempo para que la CPU pueda hacer otras cosas.

Ahora consideremos un dispositivo de almacenamiento masivo, como un disco magnético o una cinta. La información está almacenada en bloques de 128 a 2048 bytes o más, dependiendo del formato del dispositivo concreto, y la unidad básica de transferencia es el bloque. Para que un programa transfiera un bloque de datos entre el dispositivo y la memoria, primero debe arrancar la operación mecánica (como mover la cinta) para hacer que el bloque buscado pase bajo la cabeza de lectura/escritura. El tiempo de espera para que esta operación se complete se denomina *latencia de acceso*. El tiempo medio de las latencias de acceso para un disco están entre 8 y 250 milisegundos; una cinta magnética puede llegar a tener una latencia de acceso de minutos. Por esto, para evitar grandes tiempos de espera activa, parece razonable utilizar una interrupción para señalar que se ha conseguido llegar hasta el bloque seleccionado.

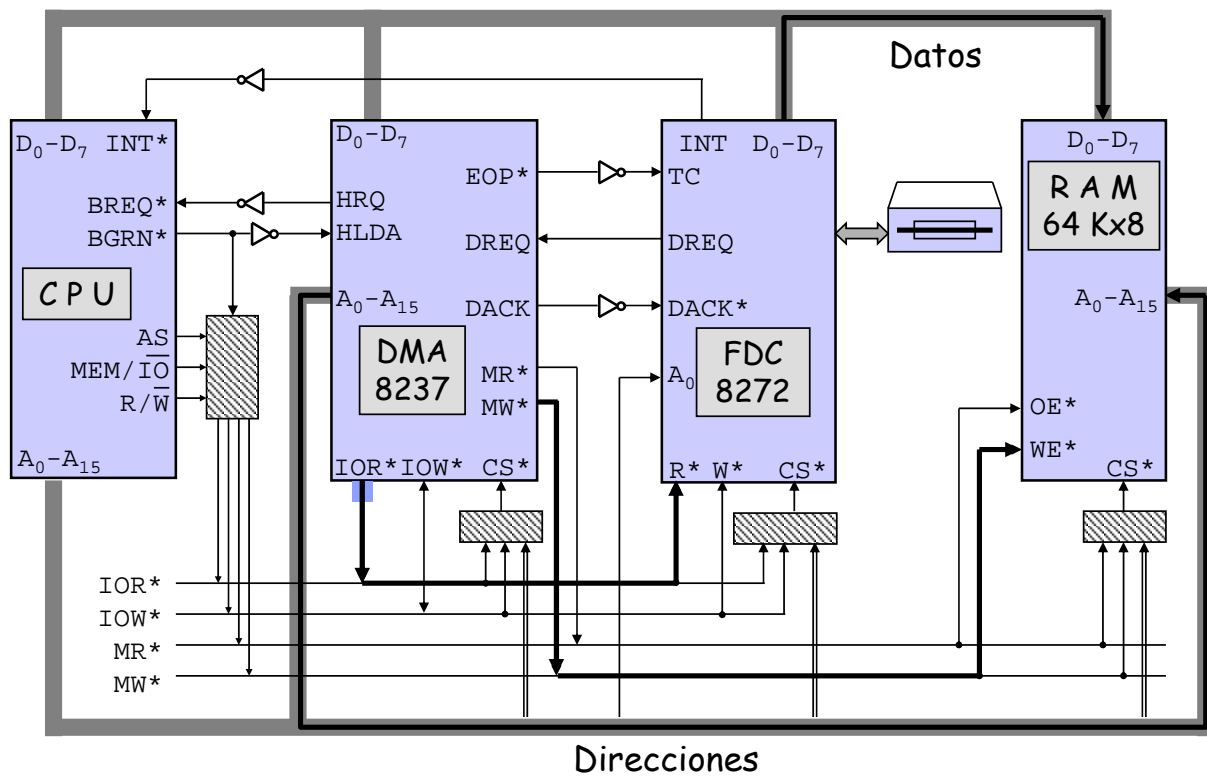
Una vez que se tiene el bloque bajo la cabeza de lectura/escritura, las interrupciones pierden su utilidad. Por ejemplo, en un disquete de baja densidad los bits de datos pueden pasar bajo la cabeza de lectura a una velocidad de un byte cada 64 microsegundos. No es fácil que una rutina de tratamiento de interrupción sea capaz de realizar el tratamiento para capturar el byte y devolver el control al programa interrumpido antes de que aparezca el siguiente byte. Seguramente haría falta un bucle de espera activa (tipo *polling*) para poder realizar la transferencia de los bytes del bloque.



Con un disquete de alta densidad la solución mediante *polling* se vuelve más difícil, pues aparece un byte cada 32 o cada 16 microsegundos. Y con un disco duro, la transferencia por *polling* resulta imposible en muchas CPUs, pues el disco entrega un byte cada 8 microsegundos como mucho. Llegado este punto, se hace necesario un método de entrada/salida distinto del ofrecido por el sistema del *polling* o las interrupciones.

Cuando hay grandes volúmenes de datos de E/S y el dispositivo los puede entregar a gran velocidad, se requiere una solución alternativa que nos resuelva estos problemas. La solución va a venir de la mano de un nuevo dispositivo denominado **Controlador de DMA (Direct Memory Access)**.

Este controlador se encarga de la transferencia directa de datos, sin intervención de la CPU, bien entre dispositivos, bien entre los dispositivos y la memoria, o bien entre dos zonas de memoria. Esto quiere decir que tiene que realizar las funciones que ejecutaría la CPU para llevar a cabo la transferencia, así, para cada byte o palabra transferida, el controlador de DMA debe suministrar las señales de dirección y todas las señales de control del bus, y ya que normalmente transfiere bloques de datos, también debe encargarse de ir incrementando las direcciones de memoria donde se transfieren los datos y de llevar la cuenta del número de datos transmitidos.



Aunque el controlador de DMA (o simplemente DMA) puede transmitir datos sin intervención de la CPU, previamente debe ser programado por ésta. Para arrancar la transferencia de un bloque de datos, la CPU le debe suministrar, al menos, la siguiente información básica:

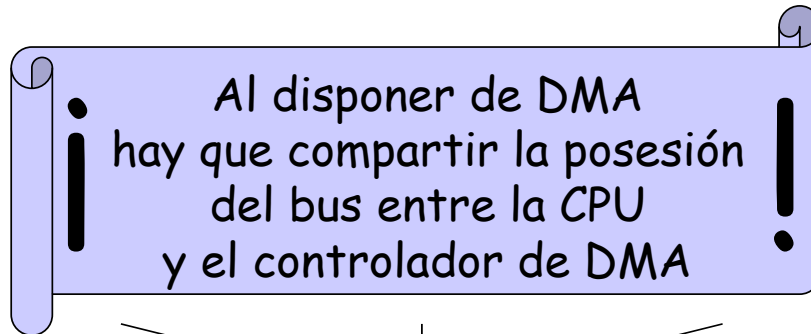
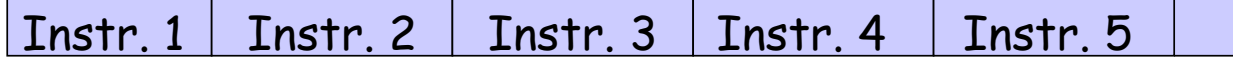
- Indicar si la operación es de lectura o escritura en memoria.
- La dirección de comienzo del buffer en memoria.
- El número de bytes a transferir.

Una vez que se le ha dado esta información al controlador de DMA, la CPU puede continuar con otro trabajo mientras, paralelamente, el DMA se encarga de la transferencia del bloque de datos byte a byte, o palabra a palabra entre el dispositivo y la memoria. Cuando la transferencia está completa, el DMA genera una interrupción para avisar a la CPU. De esta manera, la CPU solamente está implicada en la transferencia al comienzo y al final de la transferencia del bloque de datos.

En la E/S por *polling* había que estar comprobando el valor de un registro de estado; en la E/S por interrupciones el dato se leía direccionando un registro de datos del controlador del dispositivo. Cuando se trabaja con ayuda de un DMA, la comunicación y sincronización entre el DMA y el controlador del dispositivo no se realiza mediante los puertos o registros que vimos en los sistemas de E/S mediante *polling* o interrupciones, sino directamente mediante patas de control y, por supuesto, los buses de direcciones y datos. Con DMA, los registros del controlador del dispositivo solamente se utilizan en la programación del controlador para indicar una acción, como que se desea leer o escribir un bloque de datos, y para indicar el modo de la transferencia; a partir de ahí, el resto de la sincronización es mediante señales de control.

En el esquema de arriba, para leer un sector del disquete, la CPU escribirá en los registros de control del controlador de disquete FDC 8272, y programará el controlador de DMA para efectuar una transferencia del tamaño de un sector desde el FDC a la memoria. A partir de ahí, la CPU se dedicará a otra labor. Cuando el FDC tenga un dato listo para transmitir activará la pata *DREQ* (*Data Request*), con lo que el DMA solicitará la posesión del bus. Cuando el DMA tenga el control del bus, activará las señales de lectura *IOR* y de escritura *MW*, y le indicará a la memoria la dirección de escritura del dato procedente del FDC. Por último, activará la señal *DACK* (*Data Acknowledge*) y el FDC pondrá el dato en el bus de datos. Transferido el dato a la memoria, el DMA desactiva las señales de lectura y escritura, y la señal *DACK*. El controlador de disquete responde desactivando *DREQ*. Cuando el FDC vuelve a tener otro dato disponible se repite esta secuencia de pasos.

Así, hasta que el contador del DMA llega a cero (la transferencia del bloque ha terminado); entonces mediante su señal *EOP* (*End Of Process*) activa la pata *TC* (*Terminal Count*) del controlador de disquete, con lo que el FDC activa la pata *INT* para indicarle a la CPU que la lectura del sector ha terminado.

Ejecución sin DMA

↓

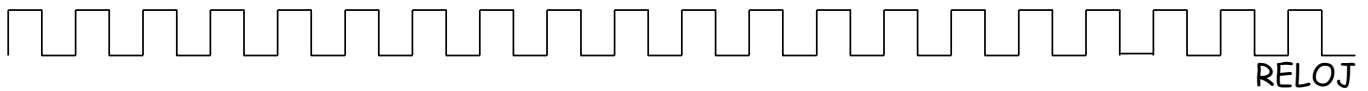
Política de Reparto

Obsérvese que cuando se realiza la transferencia de N palabras mediante un DMA, la única interferencia es el tiempo que hay que pedirle prestado el bus a la CPU durante N ciclos para realizar la transferencia de las N palabras. Este tiempo es muy inferior al que se requeriría en un sistema de E/S por interrupciones, en cuyo caso, por cada palabra que llega se tiene que ejecutar la rutina de tratamiento de la interrupción correspondiente, desde luego con un tiempo muy superior al de un ciclo de memoria.

Como ya hemos visto, el controlador de DMA, una vez programado, es capaz de realizar la transferencia de datos de forma autónoma, pero esto no resulta totalmente "gratis".

Si con ayuda de un DMA se realiza una operación de E/S con un dispositivo de alta velocidad, como un disco, se requieren muchos ciclos de bus para realizar la transferencia. Durante este tiempo la CPU tendrá que esperar (un DMA siempre tiene mayor prioridad que la CPU, ya que, normalmente, los dispositivos de E/S no toleran retardos y se puede perder la información si ésta no se recoge a tiempo).

Nos encontramos entonces con que **la posesión del bus debe compartirse entre la CPU y el controlador de DMA**. Veamos a continuación diversas maneras de compartir el bus entre la CPU y un controlador de DMA.

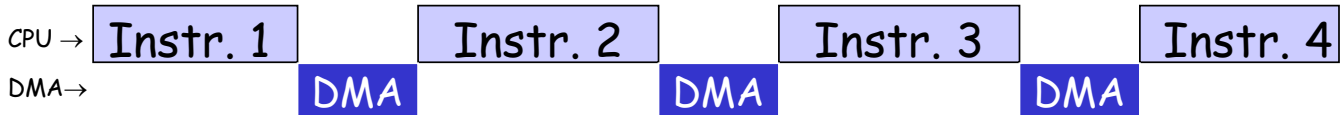


RELOJ

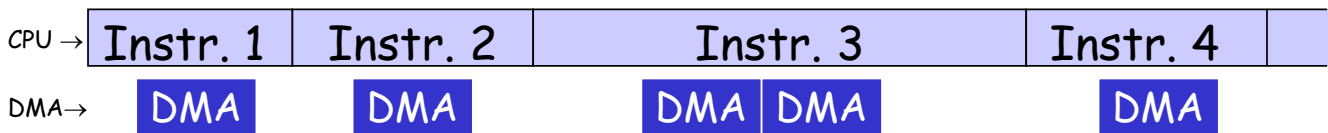
Transferencia de Bloque



Robo de Ciclo



Bus Transparente



Modo Bloque (Burst)

Cuando el DMA toma posesión del bus, se realiza la transferencia de un bloque de un gran número de palabras de datos, y la CPU queda en estado de HALT o STOP, puesto que no puede utilizar los buses. Cuando finaliza la transferencia, el controlador de DMA envía una interrupción a la CPU, con lo que ésta sale del estado de parada, atiende la interrupción y continúa la ejecución del programa que se interrumpió al ceder el bus al DMA.

Este modo es el que se utiliza con dispositivos de memoria secundaria, como los discos, donde las transferencias de datos no se pueden detener o ralentizar porque se produciría una pérdida de información. Si bien es el que proporciona la transferencia de datos más rápida, lo es a costa de tener la CPU parada (sin ejecutar instrucciones) mientras se realiza la transferencia.

Robo de Ciclo

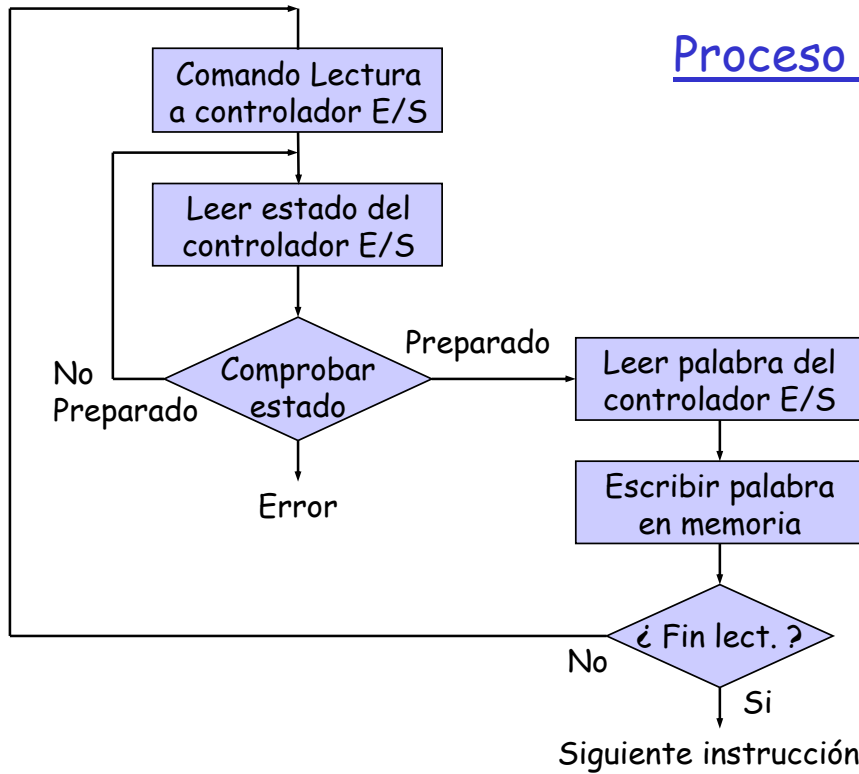
Una alternativa al *modo bloque* es el *robo de ciclo*, en el que cuando se le cede el control del bus al DMA, se le permite que lo utilice para transferir una única palabra de datos, después de lo cual debe devolver el control del bus a la CPU. Con este sistema, la transferencia de bloques de datos tiene que realizarse mediante una serie de ciclos de bus de DMA intercalados con ciclos de bus de la CPU. Este modo reduce la velocidad de transferencia respecto al *modo bloque*, pero también reduce la interferencia que se le causa a la CPU.

De este modo, las dos actividades, la del procesador y la del controlador, se realizan en paralelo (realmente, en cuasi-paralelo), aunque es evidente que ambas resultan ralentizadas al tener que repartirse el tiempo de posesión del bus.

Bus Transparente

La interferencia comentada en las políticas anteriores puede eliminarse completamente si se diseña una interfaz de DMA que solamente *robe* ciclos de bus cuando la CPU no necesita el bus, ya que ésta no requiere la posesión del bus el 100% del tiempo de ejecución de una instrucción, pues el bus puede quedar libre cuando la CPU está decodificando una instrucción, cuando está calculando la dirección de los operandos o del resultado o cuando está utilizando la Unidad Aritmético-Lógica.

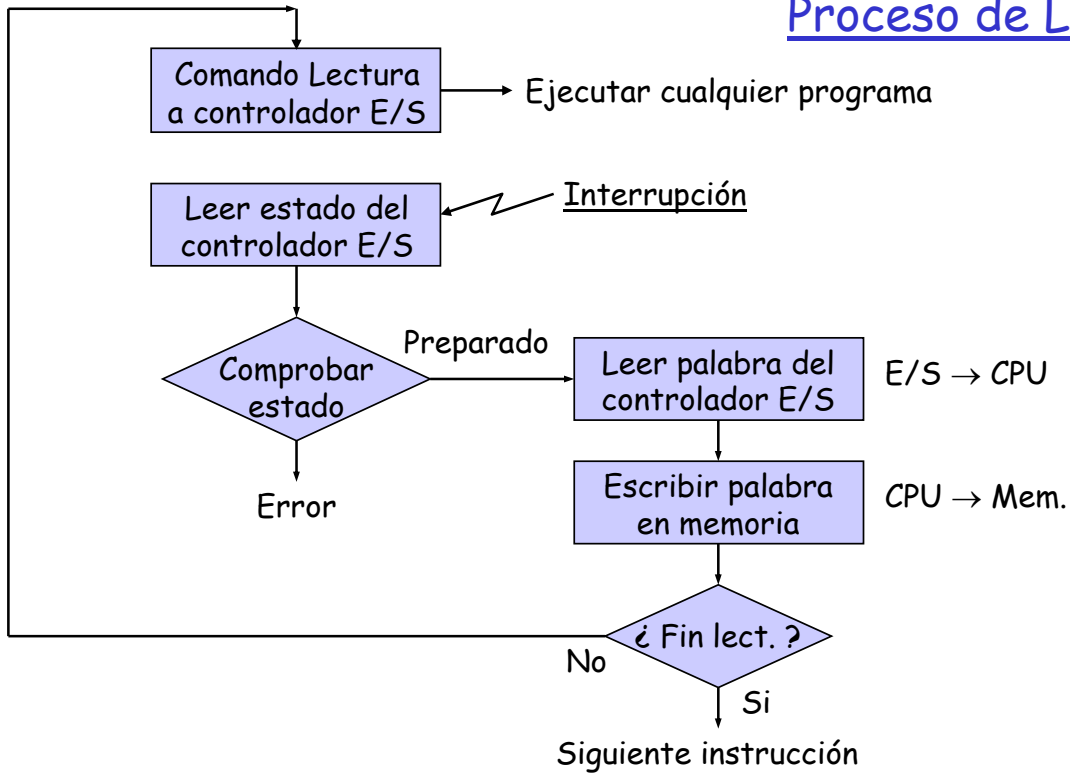
En los sistemas con memoria caché actuales, las operaciones de DMA pueden realizarse completamente en paralelo con la actividad normal de la CPU, es decir, la CPU puede estar accediendo a código y datos de la memoria caché mientras el DMA realiza la transferencia de datos entre el dispositivo de E/S y la memoria principal. Para ello, se dispone de un bus dedicado para comunicar la CPU con la caché.

Proceso de Lectura

En las siguientes transparencias simplemente se muestran, a modo de resumen, los esquemas de los tres métodos de E/S que hemos visto.

Obsérvese cómo en los métodos por sondeo y por interrupciones, el bucle principal se repite mientras no se haya realizado completamente el movimiento del bloque de datos.

Proceso de Lectura



Proceso de Lectura