Tema 8: Organización de la Entrada/salida

Objetivos:

- Identificar las funciones básicas que debe disponer una unidad de E/S con independencia del periférico conectado.
- Analizar los mecanismos de sincronización entre la unidad de E/S y la CPU, con especial énfasis en el sistema de interrupciones.
- Estudiar el sistema de acceso directo a memoria (DMA) utilizado cuando la velocidad y el volumen de datos es elevado.
- Introducir la estructura, funcionamiento y programación de los procesadores de E/S o canales.

Contenido:

1. **Funciones implicadas en las operaciones de entrada/salida**
2. **Estructura del sistema de E/S: módulos de e/s y controladores de dispositivos**
3. **Mecanismos básicos de e/s: sincronización**
4. **E/S controlada por programa**
5. **E/S por interrupción: gestión de interrupciones**
6. **E/S por acceso directo a memoria (*DMA*): motivación**
7. **Procesadores de *E/S*: tipos y estructura.**

1. Funciones implicadas en las operaciones de entrada/salida

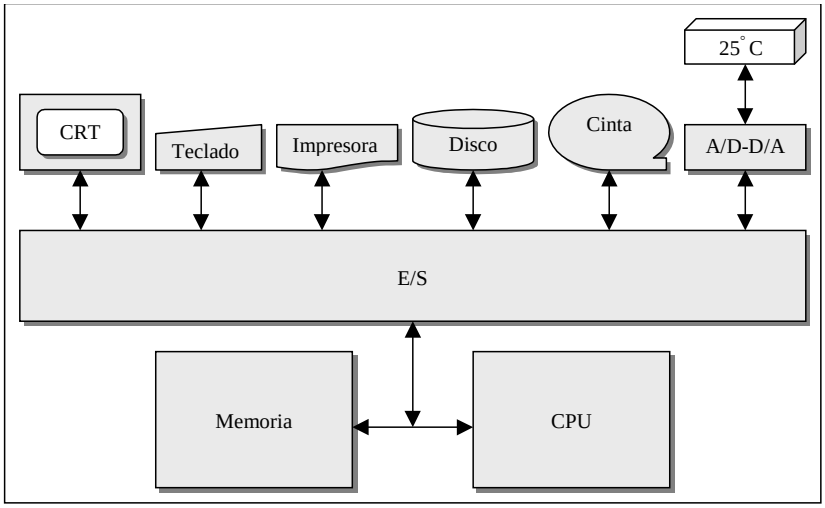
Para que un computador pueda ejecutar un programa debe ser ubicado previamente en la memoria, junto con los datos sobre los que opera, y para ello debe existir una unidad funcional de entrada de información capaz de escribir en la memoria desde el exterior. Análogamente, para conocer los resultados de la ejecución de los programas, los usuarios deberán poder leer el contenido de la memoria a través de otra unidad de salida de datos. La unidad de Entrada/Salida (E/S) soporta estas funciones, realizando las comunicaciones del computador (memoria) con el mundo exterior (periféricos). Los dispositivos periféricos que se pueden conectar a un computador se suelen clasificar en tres grandes grupos:

a) Dispositivos de presentación de datos. Son dispositivos con los que interactúan los usuarios, portando datos entre éstos y la máquina, por ejemplo, ratón, teclado, pantalla, impresora, etc.

b) Dispositivos de almacenamiento de datos. Son dispositivos que forman parte de la jerarquía de memoria del computador. Interactúan de forma autónoma con la máquina, aunque también sirven para el intercambio de datos con el usuario, por ejemplo, los discos magnéticos.

c) Dispositivos de comunicación con otros procesadores. Permiten la comunicación con procesadores remotos a través de redes, por ejemplo, las redes de área local o global.

d) Dispositivos de adquisición de datos. Permiten la comunicación con sensores y actuadores que operan de forma autónoma en el entorno del computador. Se utilizan en sistemas de control automático de procesos por computador y suelen incorporar conversores de señales A/D y D/A.



Los dispositivos de transporte y presentación de datos representan una carga muy baja de trabajo para el procesador comparados con los dispositivos de almacenamiento. La siguiente tabla muestra las velocidades de transferencia típicas para diferentes dispositivos:

Dispositivos	Velocidad
Sensores	1 Bps – 1 KBps
Teclado	10 Bps
Línea de comunicaciones	30 Bps – 20 MBps
Pantalla (CRT)	2 KBps
Impresora de línea	1 – 5 KBps
Cinta (cartridge)	0.5 – 2 MBps
Disco	4.5 MBps
Cinta	3-6 MBps

Aunque la velocidad de transferencia de los dispositivos de presentación de datos ha sido tradicionalmente lenta comparada con la de los dispositivos de almacenamiento, en los últimos tiempos la situación está cambiando. Cada vez más, los computadores se utilizan para manejar documentos multimedia que constan de gráficos, vídeos y voz. La siguiente tabla presenta algunos parámetros de transferencia para los dispositivos modernos de E/S multimedia:

Medio	Velocidad	Retardo máximo
Gráficos	1 MBps	1 - 5 segundos
Vídeo	100 MBps	20 milisegundos
Voz	64 KBps	50 - 300 milisegundos

- Los gráficos requieren una gran capacidad de procesamiento de datos, hasta el punto que se han diseñado procesadores de propósito especial para manejar de forma eficiente las representaciones gráficas (GPU: Graphic Processor Unit).
- El problema del vídeo es simplemente la animación de los problemas gráficos, ya que debe crearse una nueva imagen cada 1/30 de segundo (33 milisegundos).
- El procesamiento de la voz es también elevado porque exige la creación o el reconocimiento de varios fonemas en tiempo real. De hecho es el medio que más capacidad de procesamiento requiere debido a que presenta el mayor grado de intolerancia por retrasos en el usuario.

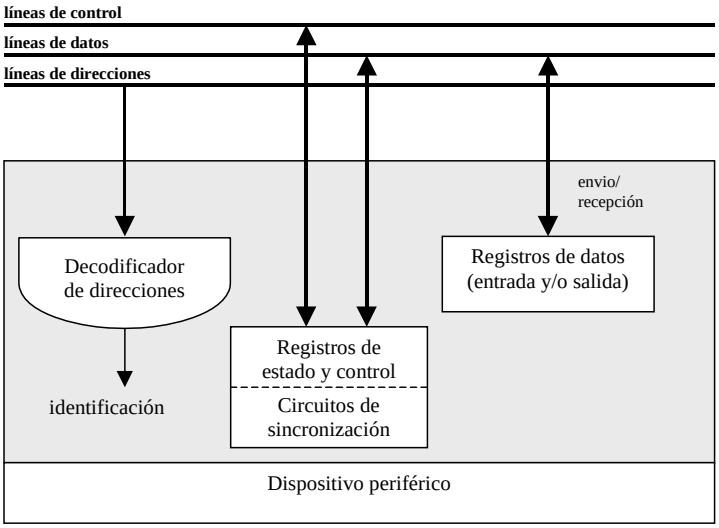
Los dispositivos periféricos que pueden conectarse a un computador para realizar entrada y salida de información presentan, pues, las siguientes características:

- Tienen formas de funcionamiento muy diferentes entre sí, debido a las diferentes funciones que realizan y a los principios físicos en los que se basan.
- La velocidad de transferencia de datos es también diferente entre sí y diferente de la presentada por la CPU y la memoria.
- Suelen utilizar datos con formatos y longitudes de palabra diferentes

No obstante estas diferencias, existen una serie de funciones básicas comunes a todo dispositivo de E/S:

- Identificación única del dispositivo por parte de la CPU
- Capacidad de envío y recepción de datos
- Sincronización de la transmisión, exigida por la diferencia de velocidad de los dispositivos de E/S con la CPU

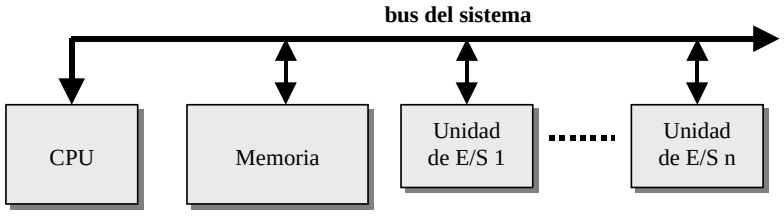
La identificación del dispositivo se realiza con un decodificador de direcciones. El envío y la recepción de datos tiene lugar a través de registros de entrada y salida de datos. Los circuitos de sincronización se manipulan por medio de registros de estado y control. El siguiente esquema representa gráficamente estas funciones:



Las tres funciones básicas se pueden realizar a través del bus del sistema que conecta la memoria y la CPU, o bien se puede utilizar un bus específico para las operaciones de E/S. Estas alternativas se traducen en dos formas de organización de los espacios de direcciones:

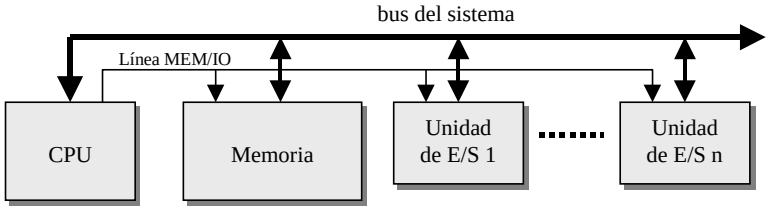
- Espacios de direcciones unificados

Las unidades de E/S se ubican en el espacio único de direcciones como si fuesen elementos de memoria. A cada unidad de E/S se le asigna un conjunto de direcciones (suficiente para diferenciar todos sus registros internos). La interacción entre CPU y unidad de E/S se realiza a través de instrucciones de referencia a memoria. El bus del sistema es único.

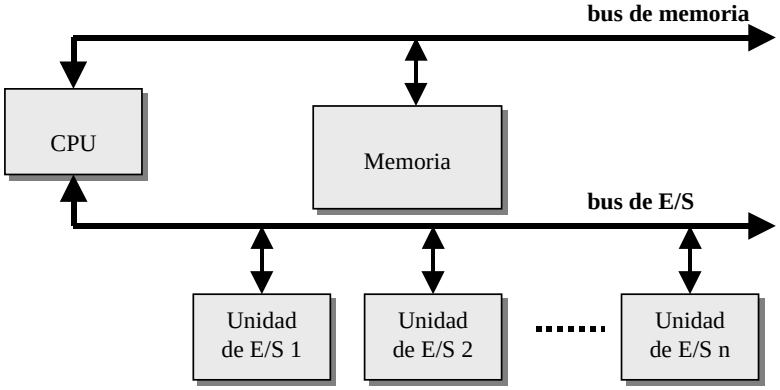


- Espacios de direcciones independientes (Memoria y E/S)

Las unidades de E/S se ubican en un espacio de direcciones diferente al de memoria. La interacción entre CPU y unidad de E/S se realiza a través de instrucciones específicas de E/S. La separación de espacios de direcciones puede soportarse con un bus único de uso compartido entre Memoria y E/S en función del estado de una línea de control MEM/IO:



Pero el desdoblamiento de espacios de direcciones puede responder a la existencia de dos buses independientes, uno para memoria (bus del sistema) y otro para E/S:



Funcionalmente son equivalentes, pero desde el punto de vista de la codificación de programas difieren en el uso de las instrucciones. En el caso de E/S asignada en memoria se utilizan instrucciones de referencia a memoria, mientras que para E/S aislada existe un grupo particular de instrucciones para realizar esta función. Los dos ejemplos siguientes muestran ambas alternativas:

E/S programada asignada en memoria

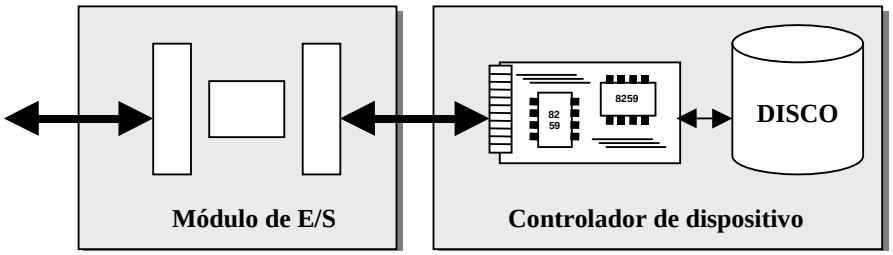
```
200     LOAD        #1          / AC  <--  #1
        STORE       517         / M[517] <-- <AC>
202     LOAD        517         / AC  <-- M[517]
        BPL         202         / Bifurca  si  N=0
        LOAD        516         / AC  <-- M[516]
```

E/S programada aislada

```
200     START I/O   5
201     TEST I/O    5
        BNR         201     / Bifurca  si  dispositivo no disponible
        IN          5
```

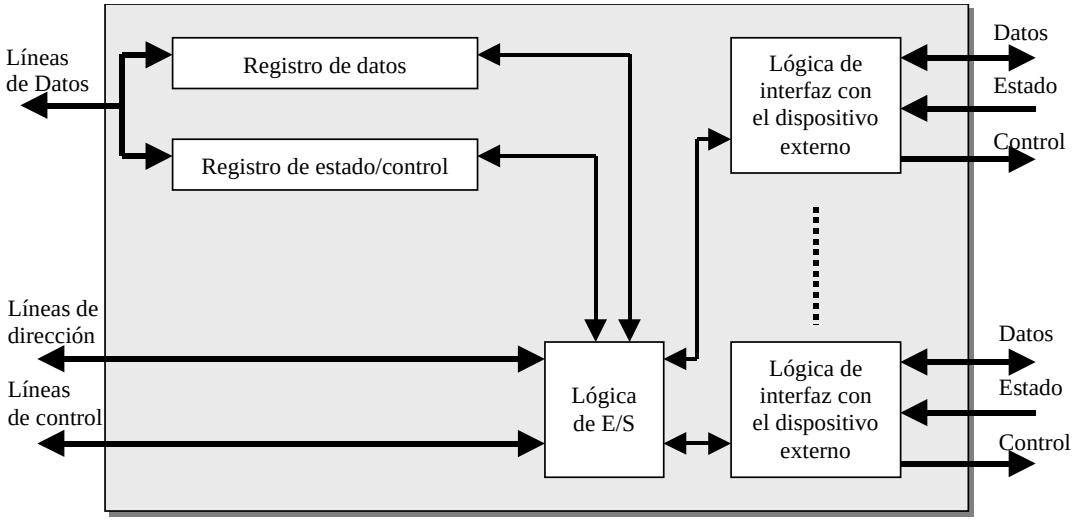
2. Estructura del sistema de E/S: módulos de e/s y controladores de dispositivos

Las diferencias existentes entre los dispositivos periféricos han hecho que la unidad de E/S de un computador se organice en torno a dos tipos de elementos, unos que soportan las características comunes a todos los dispositivos (módulos de E/S) y otros específicos para cada periférico que son los controladores de dispositivo:

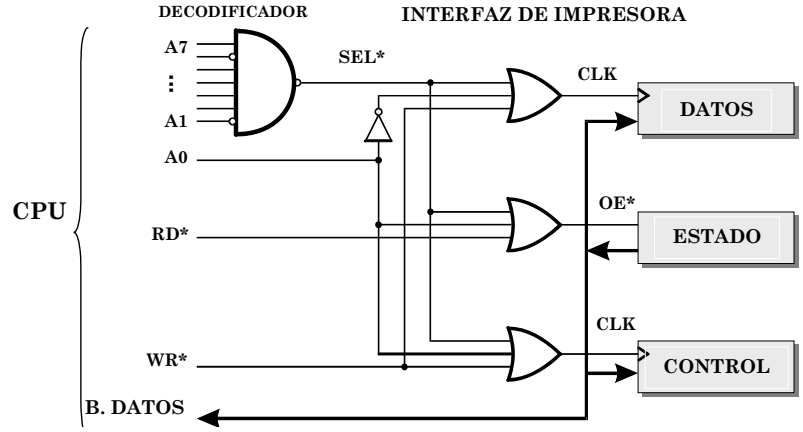


2.1. Módulos de E/S

Un módulo de E/S permite que el procesador gestione una amplia gama de dispositivos periféricos de una forma similar, ocultando los detalles concretos de temporización, formatos de datos y principios físicos de funcionamiento. El módulo de E/S se conecta con el procesador a través de un conjunto de líneas de *datos*, *dirección* y *control* (un bus). Los datos que se transfieren se almacenan temporalmente en un *registro de datos*. El estado del módulo se refleja en los bits de un *registro de estado*. El *registro de control* permite configurar y programar diferentes funciones en el módulo. Estos dos registros (estado y control) pueden unificarse en uno sólo en módulos simples. Un módulo de E/S dispone de la lógica específica para su conexión con uno o más dispositivos periféricos. En la siguiente figura se muestra la estructura general de un módulo de E/S:

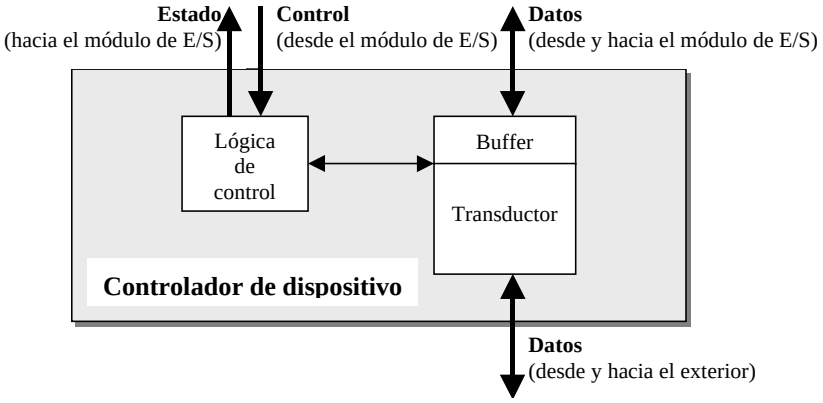


Un módulo sencillo de E/S capaz de realizar la interfaz de un computador con una impresora podría tener la estructura mostrada en la siguiente figura. El decodificador hace que el módulo se identifique (SEL*) para A7 = 1, A6 = 0, A5 = 1, A4 = 1, A3 = 1,A2 = 1, A1 = 0, A0 = --, es decir, que ocupe las direcciones de memoria: 10111100 y 10111101, la primera corresponderá a los registros de estado y control (A0 = 0), y la segunda al registro de datos (A0 = 1). Los registros de estado y control pueden tener una única dirección asociada porque el primero sólo se lee (RD*) y el segundo sólo se escribe (WR*) desde el bus.



2.2. Controlador de dispositivo (periférico)

La estructura del controlador de un dispositivo tendrá que adaptarse en cada caso a las peculiaridades específicas del periférico. Unos tendrán que actuar sobre elementos electromecánicos (impresoras de línea), otros sobre elementos ópticos (CD-ROM), o magnéticos (discos), etc. Sin entrar en las singularidades de cada uno, podemos decir que los controladores de dispositivos periféricos presentan una estructura general como la representada en la siguiente figura:



La conexión con el módulo de E/S se realiza a través de señales de *control*, *estado* y *datos*. Es la parte del controlador que omologa su comportamiento singular al esquema general de gestión de la E/S. Las señales de control determinan la función que debe realizar el dispositivo. La *lógica de control* asociada al dispositivo controla su operación en respuesta a las indicaciones del módulo de E/S. El *transductor* convierte las señales eléctricas asociadas a los datos a otra forma de energía. Además, suele existir un *buffer* asociado al transductor para almacenar temporalmente el dato que se transfiere entre el módulo de E/S y el dispositivo. En el tema 9 estudiaremos una serie de dispositivos periféricos particulares y sus controladores asociados.

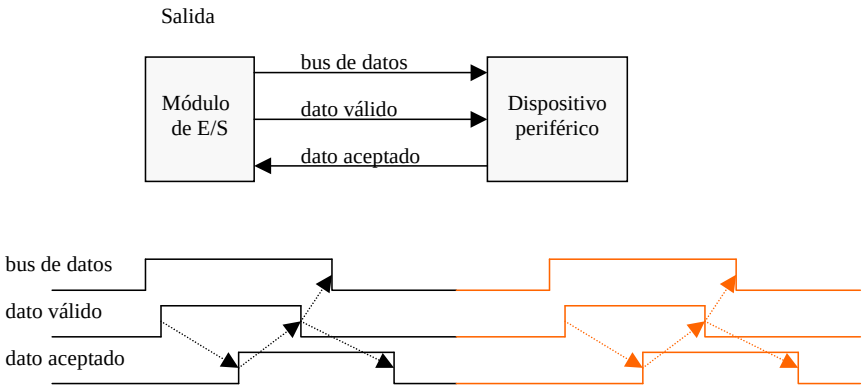
2.3. Comunicación entre el módulo de E/S y el controlador de dispositivo periférico

Analizaremos en este apartado el dialogo de señales que de forma general tiene lugar entre el módulo de E/S y el controlador de dispositivo en una transmisión de datos.

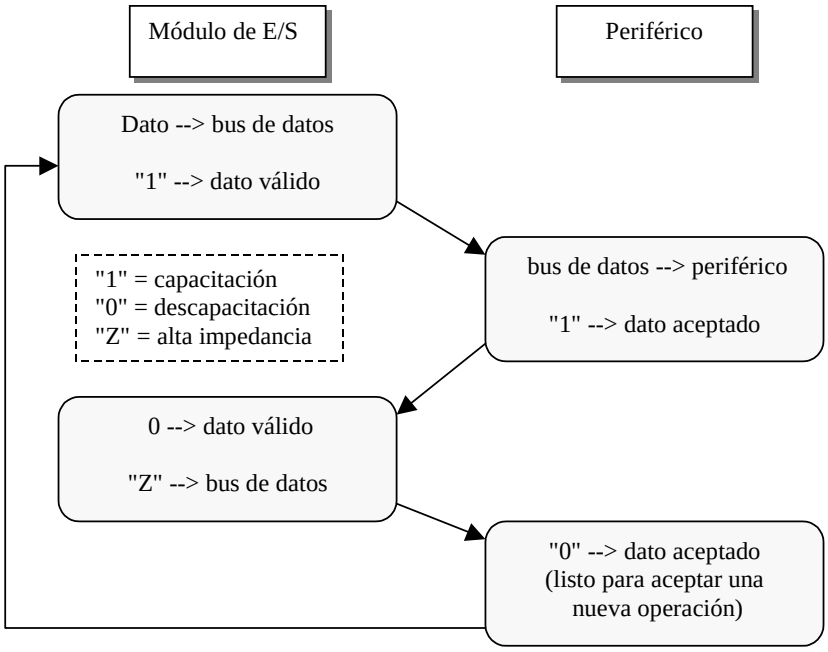
- **Salida**

En el caso de un dispositivo de salida (o de e/s en operación de salida) las líneas básicas que intervienen son las del *bus de datos* y dos de control: *dato válido* y *dato aceptado*. La primera indica al dispositivo la validez del dato presente en el bus de datos, y es activada por el módulo de E/S cuando ha estabilizado el bus de datos con el valor del dato a transmitir. La segunda la activa

el dispositivo periférico en respuesta a la lectura y procesamiento del dato, y como respuesta a la disponibilidad para aceptar un nuevo dato. En la siguiente figura hemos representado en forma gráfica la evolución temporal de las señales en una operación de salida

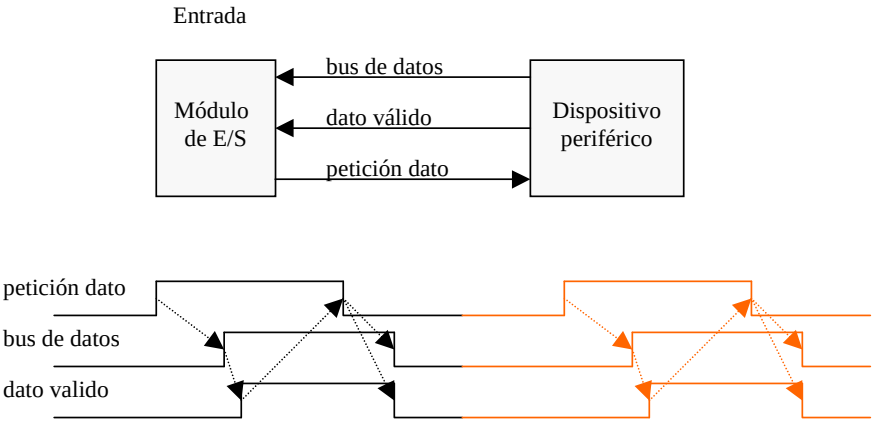


La anterior secuencia de acciones que realizan el módulo de E/S y el dispositivo periférico en una operación de salida podemos también verlas en el siguiente diagrama:

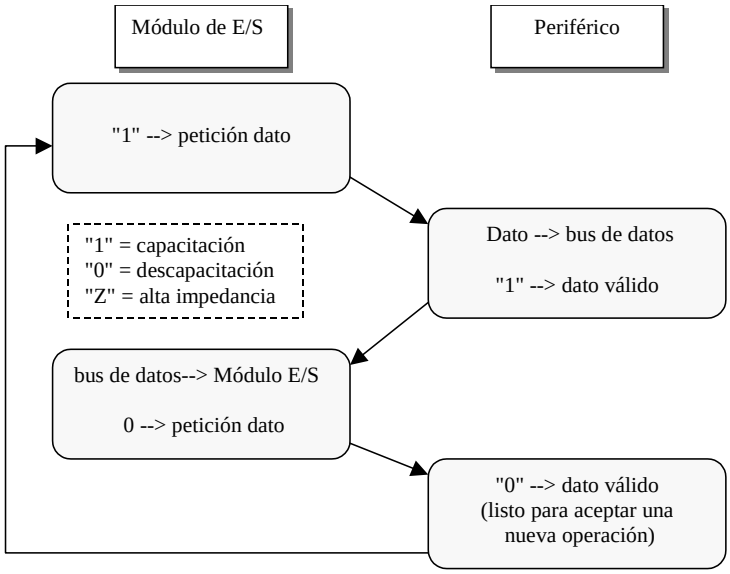


- **Entrada**

En el caso de un dispositivo de entrada (o de e/s en operación de entrada) las líneas básicas que intervienen son las del *bus de datos* y dos de control: *petición de dato* y *dato aceptado*. La primera solicita al dispositivo un dato de entrada, y es activada por el módulo de E/S. La segunda la activa el dispositivo periférico cuando ha generado el dato y su valor es estable en el bus de datos. Con la activación de esta señal el módulo de E/S conoce la validez del dato en el bus y puede proceder a su carga en el registro de datos. En la siguiente figura hemos representado en forma gráfica la evolución temporal de las señales en una operación de entrada

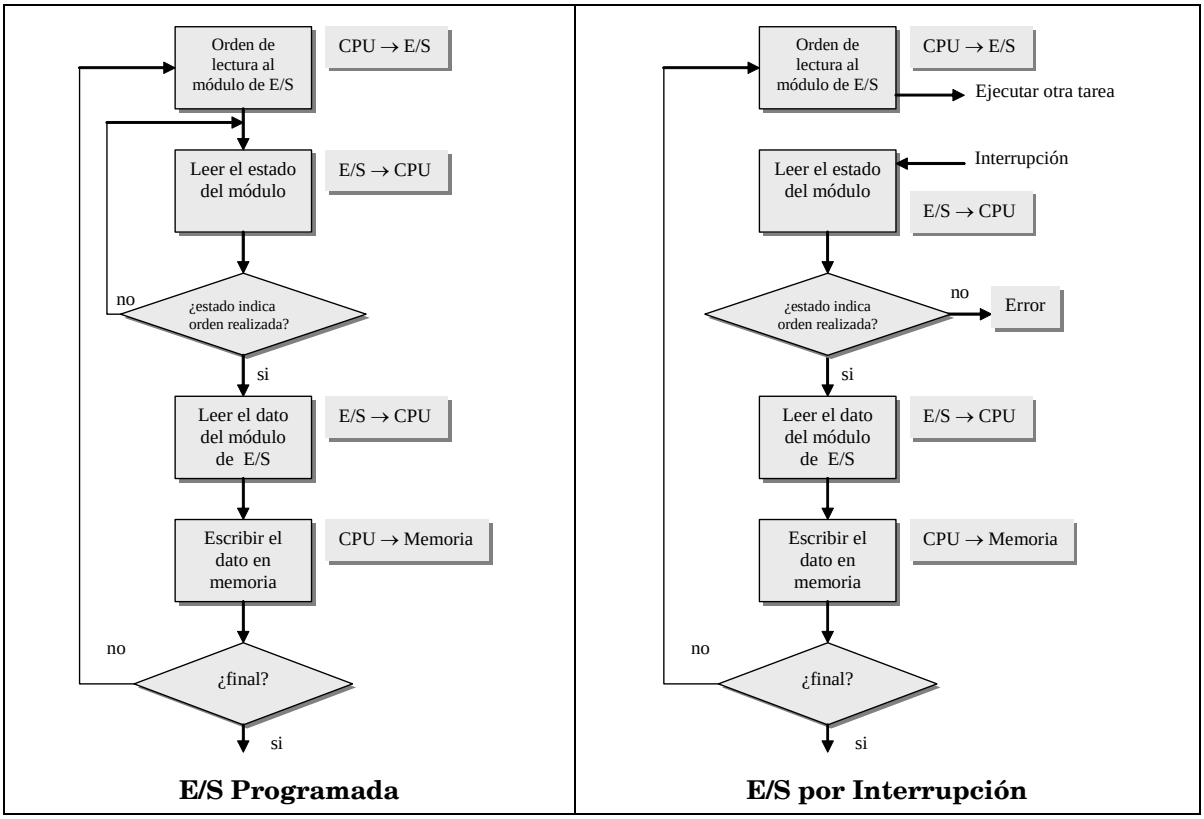


La anterior secuencia de acciones que realizan el módulo de E/S y el dispositivo periférico en una operación de entrada podemos también verlas en el siguiente diagrama:



3. Mecanismos básicos de e/s: sincronización

Las diferencias de velocidad entre la CPU y los periféricos de E/S, y lo que es más importante, la no previsibilidad del tiempo de respuesta de estos últimos, hace necesario un mecanismo de sincronismo que permita coordinar adecuadamente las transferencias de datos entre ambas unidades. Existen dos mecanismos básicos para sincronizar las operaciones de E/S con las de la CPU: sincronización por programa y sincronización por interrupción. El comportamiento de cada uno de estos mecanismos se resume en los dos siguientes organigramas:

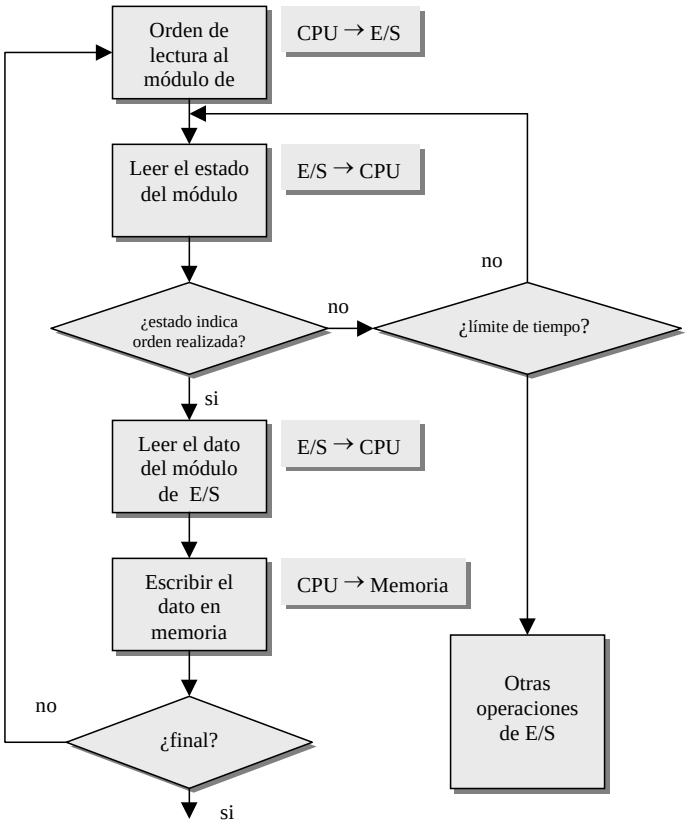


4. E/S controlada por programa

La sincronización por programa (E/S programada) es la más sencilla de implementar en un computador, sin embargo, presenta algunos inconvenientes:

- Pérdida de tiempo: el computador no realiza trabajo útil en el bucle de espera
- Impide la realización de tareas periódicas, como la exigida por el refresco de una pantalla
- Dificultades para atender varios periféricos

Los dos últimos inconvenientes podrían paliarse en parte limitando el tiempo de espera, como se muestra en el siguiente organigrama:



4.1. Ejemplos de E/S controlada por programa

Vamos a determinar en este apartado el porcentaje de tiempo de ocupación de la CPU por tres dispositivos de E/S. Para ello vamos a suponer que los conectamos a un procesador mediante E/S programada: un ratón, un disco flexible y un disco duro. Supondremos que el número de ciclos que requiere la operación completa de E/S sobre el dispositivo es de 400, y que el procesador trabaja a 500 MHz. Se supone que se puede ejecutar la consulta del estado del dispositivo con la frecuencia necesaria para que no se pierda ningún dato, y que los dispositivos están potencialmente siempre ocupados.

1. El ratón debe ser leído 30 veces por segundo para asegurar que no se pierde ningún movimiento
2. El disco flexible transfiere datos al procesador en unidades de 16 bits, a una velocidad de 50KB/seg. No debe perderse ningún dato.
3. El disco duro transfiere datos en bloques de 4 palabras, y puede transferir a una velocidad de 4MB/seg. No debe perderse ningún dato.

Ratón:

Ciclos de reloj por segundo para la lectura = 30 X 400 = 12.000 ciclos por segundo

Porcentaje de ciclos de procesador consumidos en la lectura = $12 \times 10^3 / 500 \times 10^3$ = **0,002** %

El resultado muestra que el porcentaje de tiempo de CPU que utiliza el ratón es muy bajo.

Disco flexible

Frecuencia a la que debe realizarse la lectura será: 50KB/seg / 2 byte/acceso por encuesta =

25 K acceso por lectura/ seg.

Ciclos por segundo para la lectura = 25K x 400

Porcentaje de ciclos de procesador consumidos en la lectura = 25 x 1000 x 400/500 x 10^6 = **2 %**

Este tiempo ya resulta importante.

Disco duro

Hay que realizar lectura del dispositivo a una frecuencia que coincida con el ritmo al que llegarán los bloques de 4 palabras, que es 250 K veces por segundo (4 MB por segundo / 16 bytes por transferencia). Por tanto:

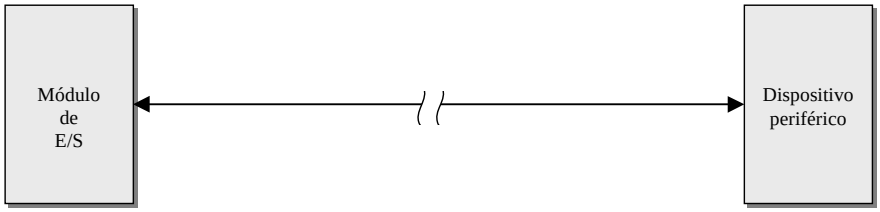
Ciclos consumidos para lectura = 250 K x 400

Porcentaje de ciclos de procesador consumidos en la lectura = 100 x 10^6 / 500 x 10^6 = **20 %**

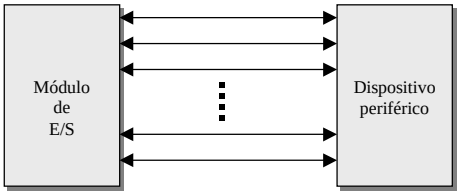
Resultado que indica que una quinta parte del tiempo del procesador se consumirá en la lectura del dispositivo.

4.2. E/S serie y paralelo

La conexión entre el módulo de E/S y el dispositivo periférico se puede realizar en forma serie o paralela. La E/S serie utiliza una única línea de transmisión, y se emplea cuando módulo de E/S y dispositivo periférico están a una distancia media o larga y el costo del medio de transmisión resulta importante. La transmisión tiene lugar haciendo que la línea adquiera sucesivamente a lo largo del tiempo el estado de cada uno de los bits constitutivos del mensaje. El tiempo asignado a cada bit determina la velocidad de transmisión en bits/segundo o baudios.



En cambio, la E/S paralela se utiliza para conectar módulos de E/S que se encuentran relativamente cerca del dispositivo periférico. Utiliza un conjunto de líneas por las que se transmiten simultáneamente (en paralelo) los bits del mensaje



Existen dos métodos para sincronizar las transmisiones en la E/S serie:

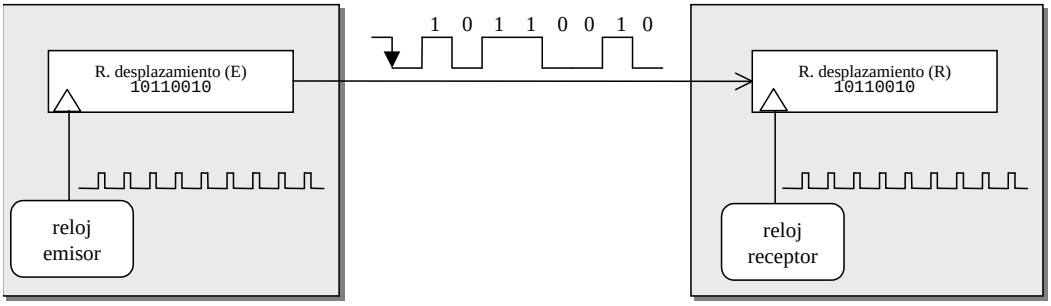
Asíncrona, que no utiliza reloj (de ahí su nombre) y que se resincroniza periódicamente con el dispositivo receptor al inicio de cada byte transmitido

Síncrona, que transmite simultáneamente la señal de datos y el reloj de sincronización

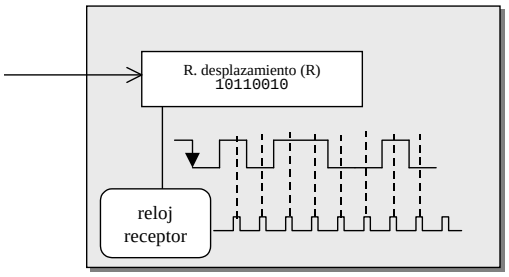
4.2.1. E/S serie asíncrona

En este tipo de E/S no existe un reloj común entre el emisor (módulo de E/S) y el receptor (dispositivo), aunque ambos utilizan señales locales de reloj para generar (emisor) y muestrear (receptor) la información, señales con valores de frecuencia nominalmente iguales. El estado de reposo de la línea de tranmisión serie es en alta. El emisor comienza la transmisión con un bit de *start* de valor 0 cuyo flanco negativo detecta el receptor como inicio de una transmisión. A partir de ese momento el emisor transmite en forma serie los bits de datos de su registro de salida (que es un registro de desplazamiento) a una frecuencia marcada por su reloj local (reloj emisor). El receptor

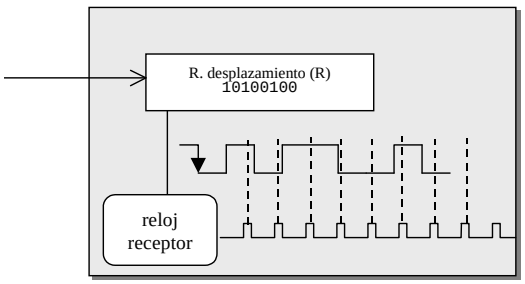
muestrea la línea con una frecuencia nominalmente igual a la del emisor sobre su registro de entrada (que es otro registro de desplazamiento). Como ambos relojes son físicamente diferentes, no se evita que con el tiempo se vaya desplazando uno respecto al otro, con el risgo que si los bits transmitidos son muchos el receptor los muestree incorrectamente. Sin embargo, como emisor y receptor se resincronizan con el bit de start del siguiente caracter transmitido, y sólo se tranmite un caracter cada vez, estos deslizamientos no producen error.



En la siguiente figura se muestra que el reloj local del receptor muestrea la línea de transmisión en instantes de tiempo correctos, es decir, dentro del intervalo de tiempo correspondiente a un bit (intervalo bit)

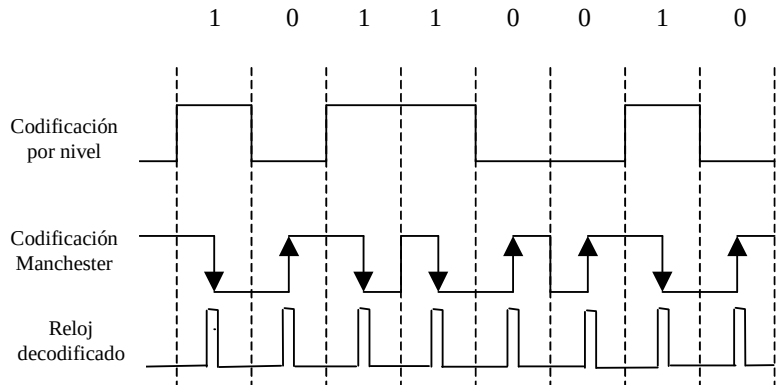


En cambio, si el reloj local del receptor tiene una frecuencia ligeramente inferior (mayor período), como se muestra en la siguiente figura, a partir del cuarto intervalo bit la señal será muestreada incorrectamente en el receptor.



4.2.2. E/S serie síncrona

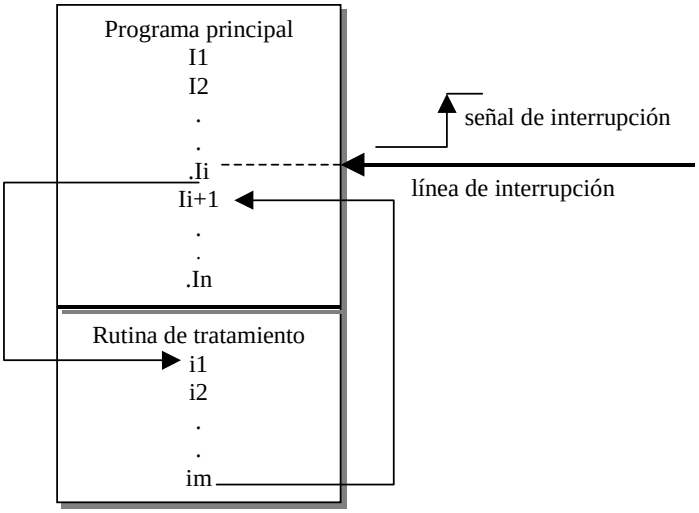
La E/S serie síncrona utiliza una señal de transmisión que codifica conjuntamente la señal de información y el reloj. Por ejemplo, el código de *Manchester,* que se muestra en la siguiente figura, utiliza una codificación por transiciones, de tal manera que el 0 se codifica como una transición positiva y el 1 como una transición negativa. De esta forma en un intervalo bit existe al menos un cambio en la señal, con lo que se puede extraer la señal del reloj.



5. E/S por interrupción: gestión de interrupciones

En la E/S programada el procesador tiene que esperar un tiempo considerable a que el módulo de E/S esté preparado para realizar la operación. El procesador espera comprobando repetidamente el estado del módulo de E/S, degradándose significativamente el rendimiento de la *CPU*. Para evitar este inconveniente se introdujo el sistema de interrupciones en los procesadores.

Básicamente una interrupción viene determinada por la ocurrencia de una señal externa que provoca la bifurcación a una dirección especifica de memoria, interrumpiendo momentáneamente la ejecución del programa. A partir de esa dirección se encuentra la *rutina de tratamiento* que se encarga de realizar la operación de E/S propiamente dicha, devolviendo después el control al punto interrumpido del programa.

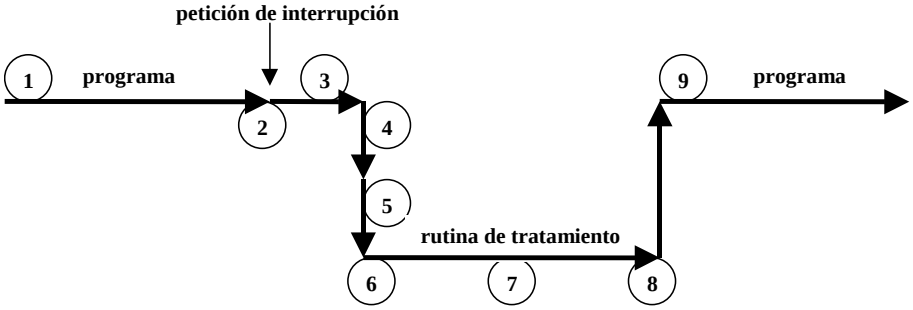


Podemos, pues, ver una interrupción como un salto a subrutina (*rutina de tratamiento*) ocasionado por una señal externa, y no por una instrucción del programa. De esta forma se pueden eliminar los tiempos muertos de consulta de la E/S programada

La implementación de un sistema de interrupciones implica introducir una fase de consulta de las líneas de interrupción al final de la ejecución de cada instrucción. En un procesador sin sistema de interrupciones, se podría conseguir un efecto similar introduciendo una instrucción de consulta y la correspondiente de salto sobre el valor de la consulta, detrás de cada instrucción natural del programa. De esta forma se garantizaría la respuesta al dispositivo de E/S en el momento que pasa a estado disponible, al tiempo que la *CPU* ejecuta instrucciones útiles del

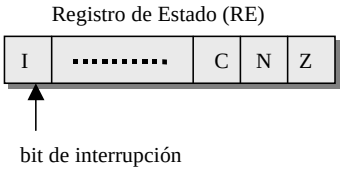
programa. El precio a pagar sería el recargo introducido por la ejecución de las parejas de instrucciones de consulta y salto introducidas detrás de cada instrucción útil del programa. Sería algo así como una *E/S* programada en la que en lugar de preguntar en el mismo punto del programa por el estado del periférico, se replica la consulta por todo el programa, intercalándose con la ejecución de las instrucciones útiles. Pues bien, un sistema de interrupciones podemos verlo como la integración en hardware del supuesto software anterior, es decir, la integración de la consulta y posible salto dentro de la ejecución de cada instrucción del repertorio.

El mecanismo de interrupción de un procesador tiene que implementar todas las medidas que hagan que se pueda bifurcar a la rutina de tratamiento y recuperar el estado del programa interrumpido cuando la rutina finaliza su ejecución. En el siguiente esquema se detallan las fases que tienen lugar en la atención a una interrupción producida por algún dispositivo periférico:

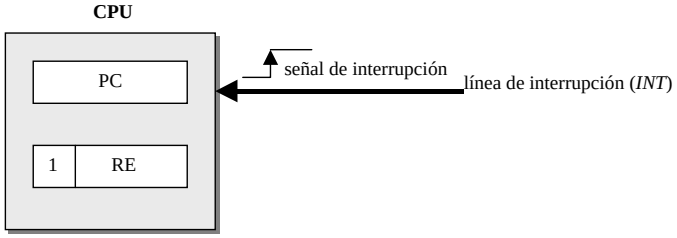


(1) El programa en ejecución (*CPU*) activa el sistema de interrupciones utilizando instrucciones que operan (ponen a 1 y a 0) sobre el bit de capacitación de las interrupciones *I* del registro de estado (*RE*):

STI I <--1; CLI I <-- 0

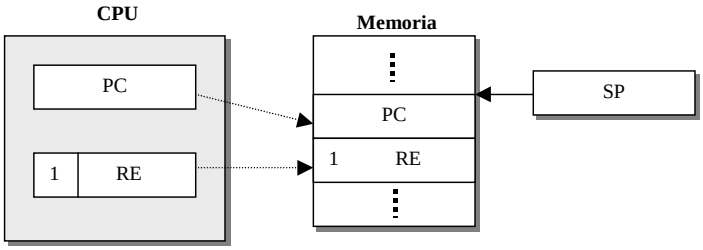


(2) Se produce la petición de interrupción por parte de algún dispositivo periférico en un instante de tiempo impredecible para la *CPU*.

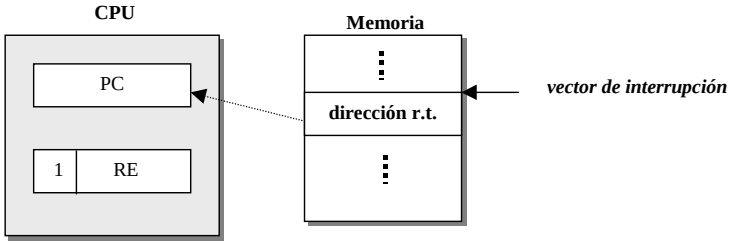


(3) La *CPU* finaliza la ejecución de la instrucción en curso.

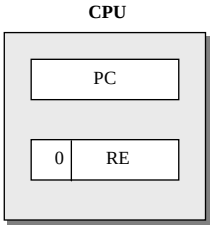
(4) La *CPU* salva automáticamente el estado en la pila, es decir, el contador de programa (*PC*) y el registro de estado (*RE*):



(5) La *CPU* obtiene la dirección de la rutina de tratamiento a partir del *vector de interrupción* (*VI*), que usualmente se ubica en memoria.



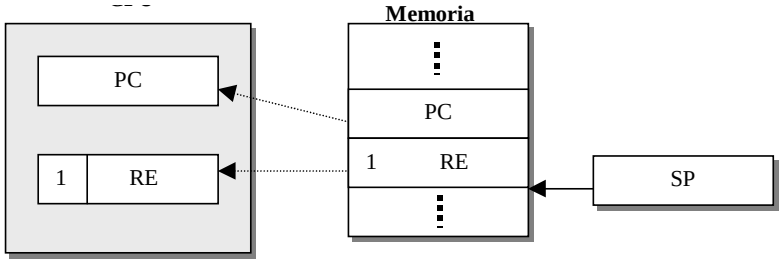
(6) La *CPU* descapacita las interrupciones (*I = 0*) para que durante la ejecución de la primera instrucción de la rutina de tratamiento no se vuelva a detectar la misma interrupción y provoque un bucle infinito.



(7) La *CPU* ejecuta la rutina de tratamiento de la interrupción que realiza lo siguiente:

- Salva en la pila los registros a utilizar
- Realiza la operación de Entrada/Salida
- Restaura desde la pila los registros utilizados

(8) Finaliza la rutina de tratamiento con la ejecución de la instrucción de retorno de interrupción (*RTI*), que restaura automáticamente el estado de la *CPU* desde la pila y vuelve al programa interrumpido:



(9) La *CPU* continúa la ejecución del programa interrumpido, quedando las interrupciones capacitadas automáticamente al recuperarse el valor *I = 1* del *RE*.

5.1. Tipos de sistemas de interrupciones

Clasificaremos las interrupciones atendiendo a dos criterios independientes: la fuente que produce la interrupción, y el modo de obtener la dirección de la rutina de tratamiento o vector de interrupción.

a) <u>Atendiendo a la fuente que produce la interrupción</u>:

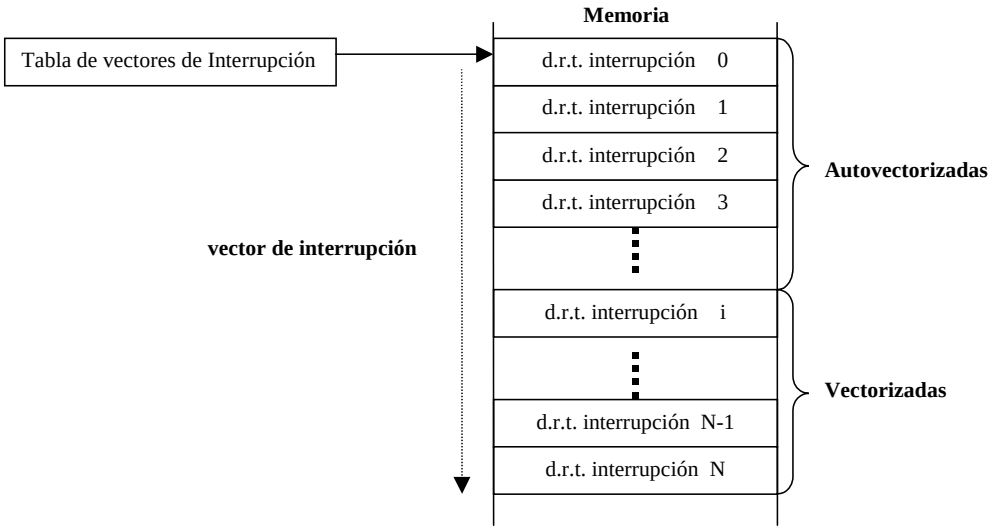
- **Interrupciones hardware**
  - <u>Internas</u>: producidas por la *CPU*
    - división por cero
    - desbordamiento
    - instrucción ilegal
    - dirección ilegal
    - logaritmo de cero
    - raíz cuadrada de negativos
    - etc.
  - <u>Externas</u>: producidas por los dispositivos de *E/S*
    - vectorizadas
    - no vectorizadas
- **Interrupciones software**: producidas por la ejecución de instrucciones de la *CPU*.

b) <u>Atendiendo al modo de obtener el vector de interrupción</u>:

<u>Interrupciones autovectorizadas</u>: el vector de interrupción es fijo, una posición de memoria asociada a la línea de interrupción.

<u>Interrupciones vectorizadas</u>: el vector de interrupción o parte de él lo suministra el propio periférico cuando se le reconoce la interrupción.

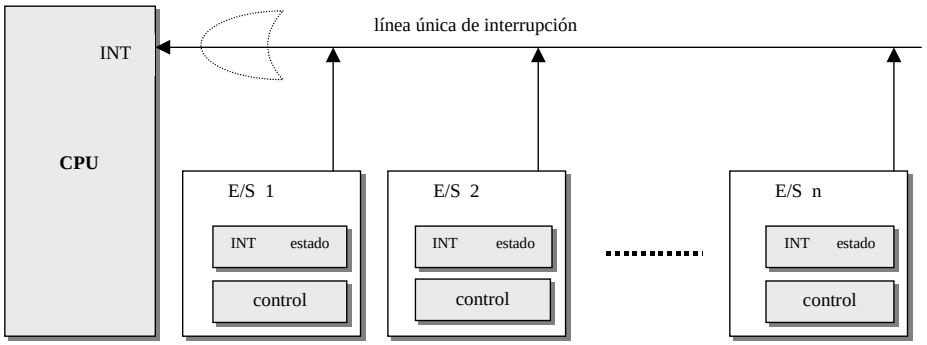
El método usual de obtener la dirección de la rutina de tratamiento de la interrupción en ambos casos es a través de la tabla de vectores de interrupción, tabla que contiene las direcciones de las rutinas de tratamiento de cada interrupción.



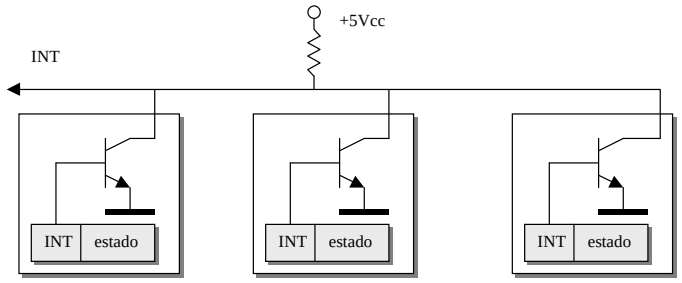
5.1.1. Interrupciones autovectorizadas

Disponen de una línea única de petición interrupción a la que se pueden conectar más de un dispositivo periférico, efectuándose en el punto de conexión la función lógica *OR*. El vector de interrupción para esta línea ocupa una posición fija de la tabla de vectores de interrupción. Cuando uno o más de los dispositivo periféricos conectados a la línea genera una señal de interrupción, la *CPU* debe identificar por software el dispositivo causante de la interrupción, y en caso de ser

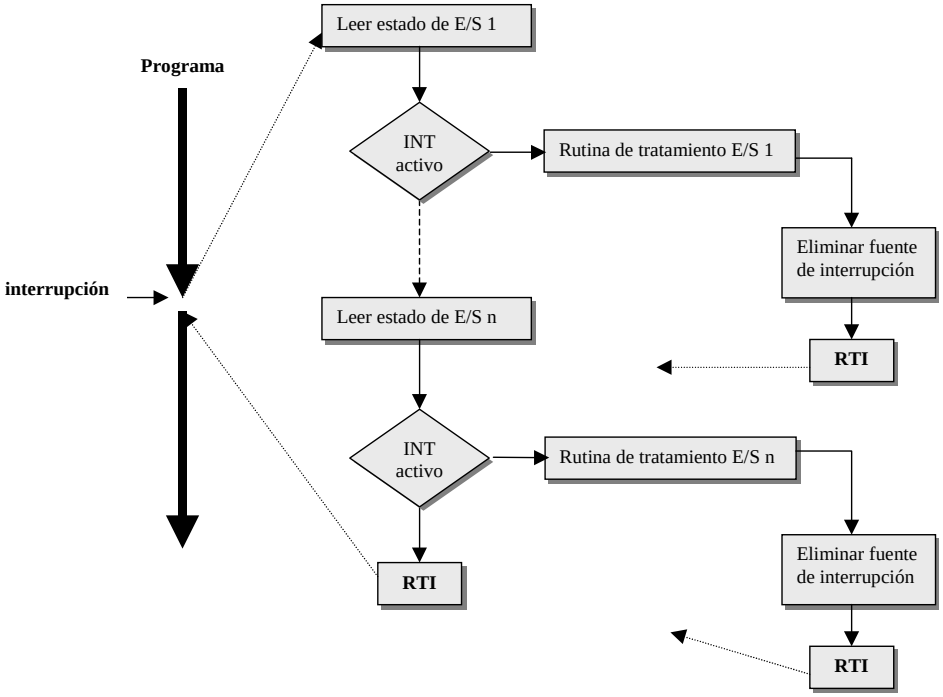
varios, establecer el orden de prioridad entre ellos. La identificación se realiza consultando los registros de estado locales de cada módulo de *E/S*.



La *OR-cableada* en la línea de interrupción *INT* se realiza normalmente con tecnología de transistores con colector abierto. Por ello normalmente la señal activa de interrupción es en baja.



La rutina general de tratamiento asociada a la línea de interrupción comienza su ejecución identificando por encuesta el periférico causante de la interrupción, para bifurcar después a la rutina de tratamiento específica de dicho periférico, en donde se realiza la operación de E/S propiamente dicha:

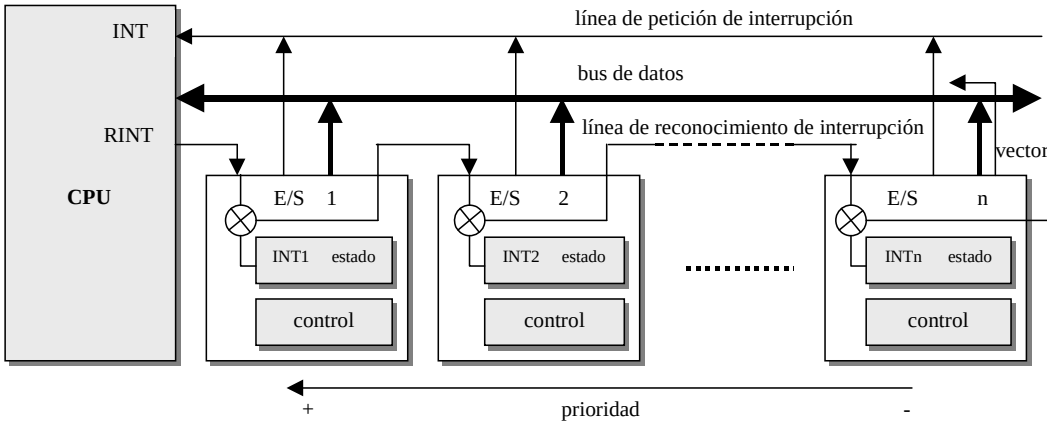


5.1.2. Interrupciones vectorizadas:

- Disponen de dos líneas de interrupción, una de petición y otra de reconocimiento.

El vector de interrupción es generado por el dispositivo que produce la interrupción.

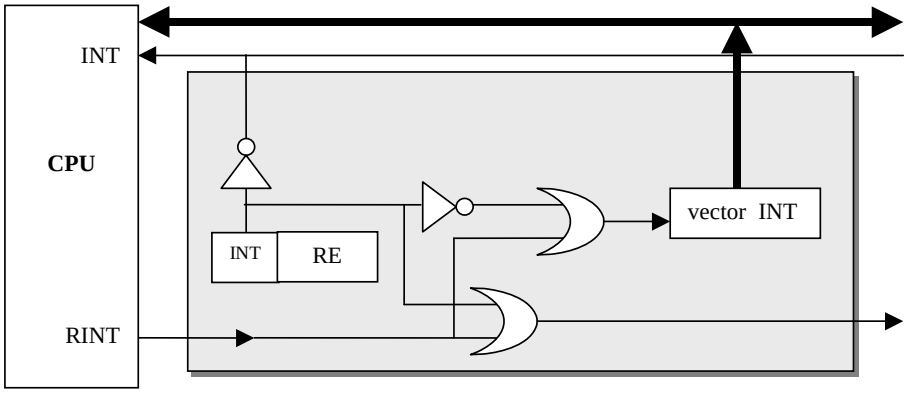
- Utiliza un mecanismo de *daisy chaining* en la transmisión de la señal de reconocimiento.



El proceso de interrupción se desarrolla en los siguientes pasos:

1. El dispositivo de *E/S* genera la petición de interrupción activando *INT*.
2. La *CPU* la reconoce activando *RINT*.
3. Los dispositivos de *E/S* que reciben *RINT* y no han interrumpido, la transmiten al siguiente elemento de la cadena.
4. El dispositivo de *E/S* que recibe *RINT* y ha realizado la petición coloca el vector de interrupción *n* en el bus de datos.
5. A partir del vector *n* la *CPU* bifurca a la rutina de tratamiento correspondiente al dispositivo.

La lógica interna del paso de la señal de reconocimiento por encadenamiento (*daisy chain*) para cada módulo se representa en la siguiente figura:



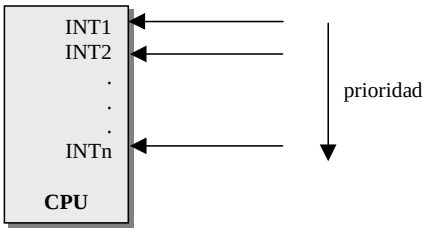
5.2. Prioridades

Cuando dos o más dispositivos de *E/S* generan una petición de interrupción, el sistema de prioridades determina la interrupción que se atiende en primer lugar.

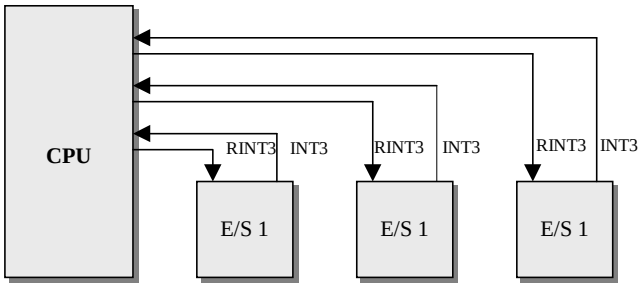
Interrupciones no vectorizadas: la prioridad la establece el software de encuesta con el orden de recorrido de todos los dispositivos de E/S salida conectados a una línea. Es decir, se establece por software.

Interrupciones vectorizadas: la prioridad la determina el orden de proximidad a la *CPU* establecido por la línea de retorno de interrupción Es decir, se establece por hardware.

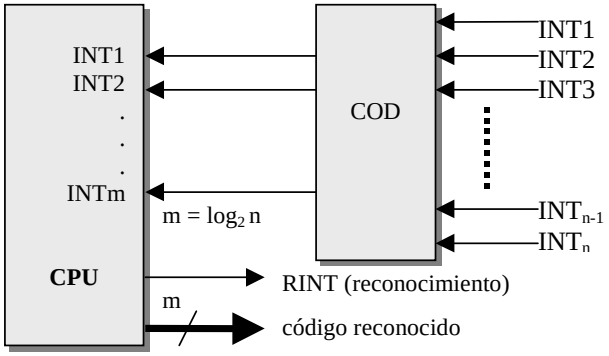
Cuando existen más de una línea de interrupción, la prioridad viene establecida por hardware en el diseño del procesador:



En ocasiones, cada línea de interrupción tiene su propia señal de reconocimiento:



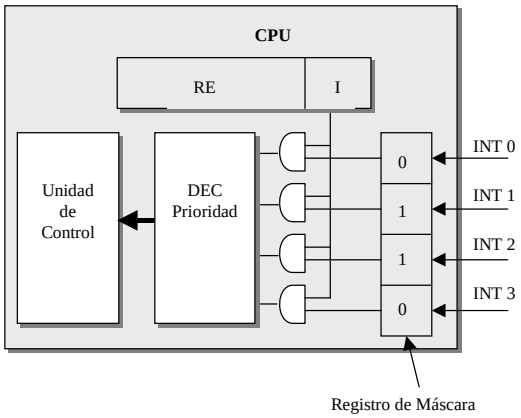
Aunque suele ser frecuente, para minimizar el número de pines del chip, que las diferentes líneas entren codificadas a la *CPU*, siendo necesario el uso de un codificador externo:



En estos casos suele existir una única señal de reconocimiento *RINT*, generándose por las líneas menos significativas del bus de direcciones el código de la interrupción reconocida, que será la más prioritaria entre las que realizaron la petición.

5.3. Enmascaramiento de interrupciones

El sistema de interrupciones de un procesador dispone en general de la posibilidad de ser inhibido, es decir, impedir que las interrupciones sean atendidas por la *CPU*. Esta posibilidad hay que utilizarla en determinadas ocasiones en las que por ningún concepto se puede interrumpir el programa en ejecución. Además de la inhibición total del sistema, existe la posibilidad de enmascarar individualmente algunas de las líneas de interrupción utilizando un registro de máscara:

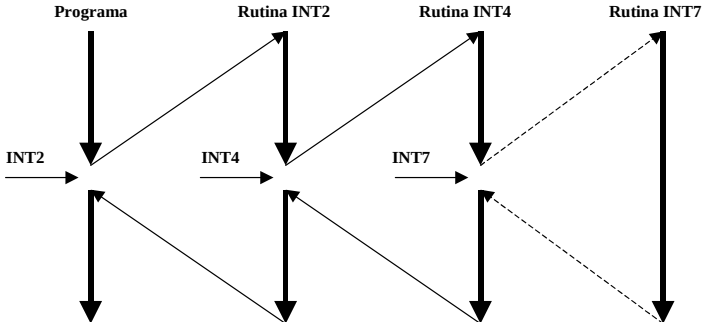


La descapacitación de algunas interrupciones también se puede realizar a nivel local del propio módulo de E/S, actuando sobre su registro de control.

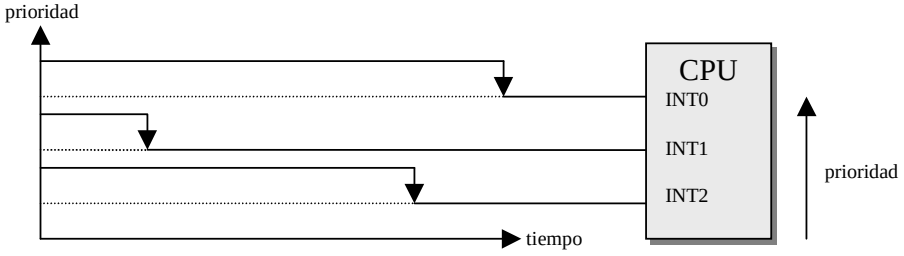
A nivel software también se pueden inhibir las interrupciones producidas sobre una misma línea por diferentes periféricos, en la fase de identificación de la rutina de tratamiento correspondiente.

5.4. Anidamiento de interrupciones

Un tema importante en un sistema de interrupciones es el de la capacidad de anidar interrupciones, es decir, la posibilidad de interrumpir la rutina de tratamiento de una interrupción por la generación de otra interrupción. Tal posibilidad viene determinada por el sistema de prioridades: "*un dispositivo sólo puede interrumpir a otro cuando su nivel de prioridad es mayor que el que se está atendiendo*".



Para describir con más detalle la forma en que opera la combinación de las prioridades y el anidamiento en un sistema de interrupciones, consideremos una *CPU* con tres líneas de interrupción: *INT0, INT1, INT2*, siendo la primera la más prioritari, y la última la menos prioritaria. Supongamos que llegan tres peticiones de interrupción en el orden temporal del siguiente esquema:

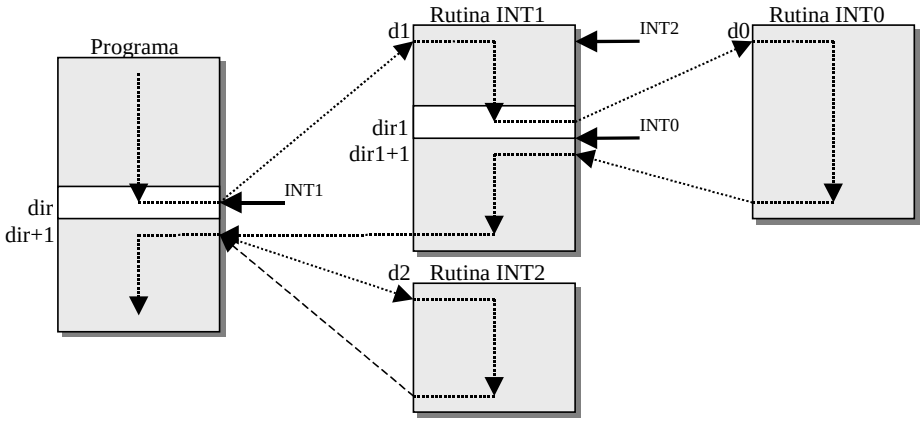


La *CPU* dispone de tres bits en su registro de estado para permitir la inhibición de forma independiente cada línea:

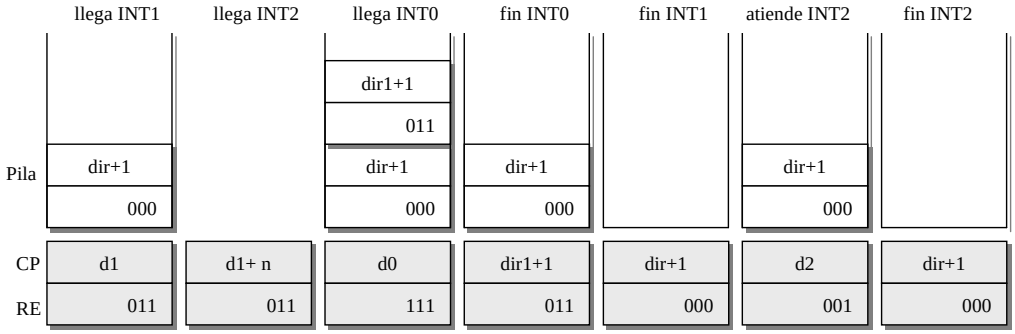
Si INTi = 0 nivel capacitado

Si INTi = 1 nivel descapacitado

Las interrupciones con menor prioridad no podrán interrumpir la rutina de tratamiento de una de mayor prioridad, por lo que la secuencia de peticiones y servicios será la que aparece en el siguiente esquema:



La evolución consiguiente de la pila y el estado de la *CPU* (*CP* y *RE*) será el siguiente:



5.5. Ejemplos de sistemas de interrupciones

Estudiaremos en este apartado de forma general los sistemas de interrupciones de dos micrprocesadores, uno de *8* bits, el *MC 6809*, y otro de *16*, el *MC 68000*. Con el primero estudiaremos la forma de multiplicar una línea de interrupción autovectorizada en varias, también autovectorizadas, utilizando un dispositivo hardware. Con el segundo veremos un sistema de interrupciones más complejo con interrupciones autovectorizadas y vectorizadas.

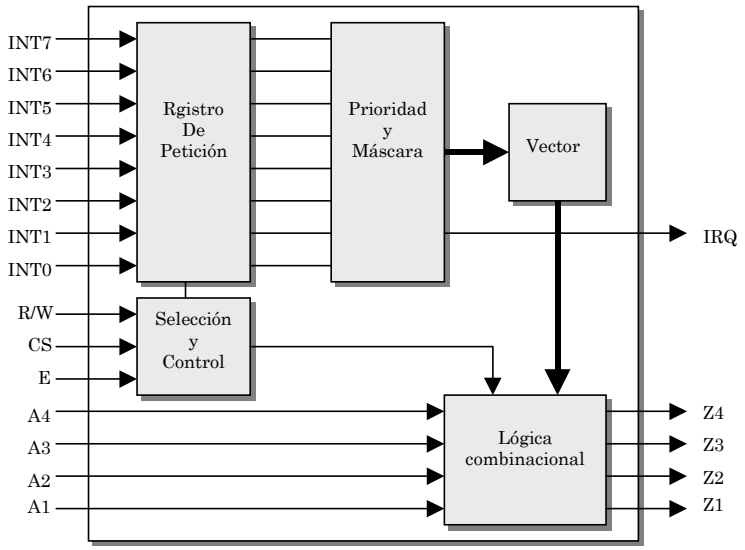
5.5.1. Sistema de interrupciones del microprocesador MC 6809

El *MC 6809* dispone de las siguientes líneas de interrupción:

Línea	Prioridad	Dirección Rutina	Bit de máscara en RE	Comportamiento
IRQ	Baja	FFF8 – FFF9	I	Salva todos los registros
FIRQ	|	FFF6 – FFF7	F	Salva CP y RE
NMI	↓	FFFC – FFFD	No enmascarable	Salva todos los registro
RESET	Alta	FFFE - FFFF	No enmascarable	No salva registros

Veremos como se puede transformar la línea *IRQ* en 8 líneas de interrupción diferentes. Para ello se aprovecha el que cuando se produce una interrupción por esta línea y es atendida por la *CPU*, ésta accede a su vector de interrupción que se encuentra en las direcciones *FFF8* y *FFF9*, lo que significa que en el bus de direcciones aparecerán estas direcciones. Sólo en respuesta a una interrupción por IRQ se generan estas direcciones en el bus, por lo que podemos identificarla por hardware y descomponerla en 8 pares diferentes, una por cada línea nueva de interrupción.

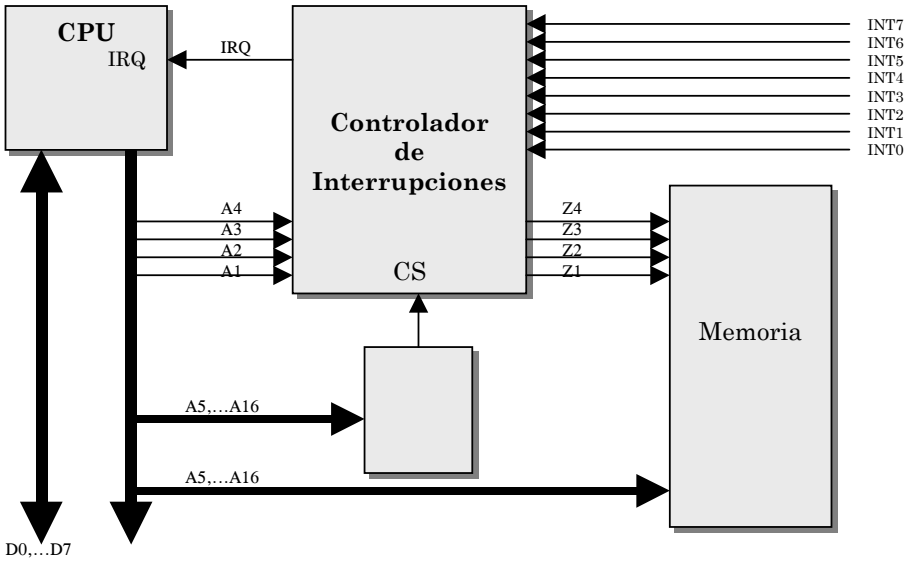
El esquema completo de un dispositivo que realiza esta conversión lo hemos representado en la siguiente figura. El dispositivo se selecciona cuando se activa la señal *CS*, por lo que esta señal deberá generarse cuando en el bus aparezcan las direcciones *FFF8* y *FFF9*. Cuando eso ocurra, el dispositivo transformará las 4 líneas menos significativas del bus de direcciones (*A4,A3,A2,A1*) en otras 4 (*Z4,Z3,Z2.Z1*) de manera que en las *Zi* se genere uno de los 8 pares de las nuevas direcciones, el correspondiente a la línea *INTi* activada de mayor prioridad.



La tabla de transformación de direcciones sería, pués, la siguiente:

Dirección entrada	Línea	Dirección salida
A16....A4 A3 A2 A1	1	A16....Z4 Z3 Z2 Z1
FFF8 – FFF9	INT7	FFE6 – FFE7
	INT6	FFE8 – FFE9
	INT5	FFEA – FFEB
	INT4	FFEC – FFED
	INT3	FFEE – FFEF
	INT2	FFF0 – FFF1
	INT1	FFF2 – FFF3
	INT0	FFF4 – FFF5

La disposición del controlador de interrupciones respecto a la *CPU* y la memoria se muestra en el siguiente esquema:

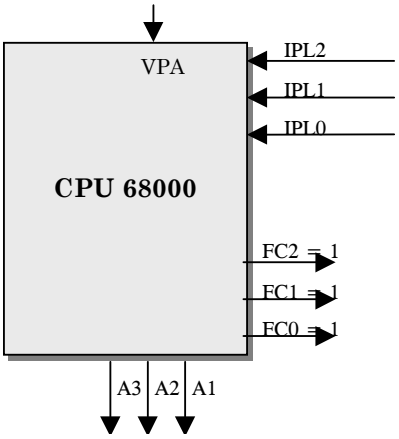


5.5.2. Sistema de interrupciones del microprocesador MC 68000

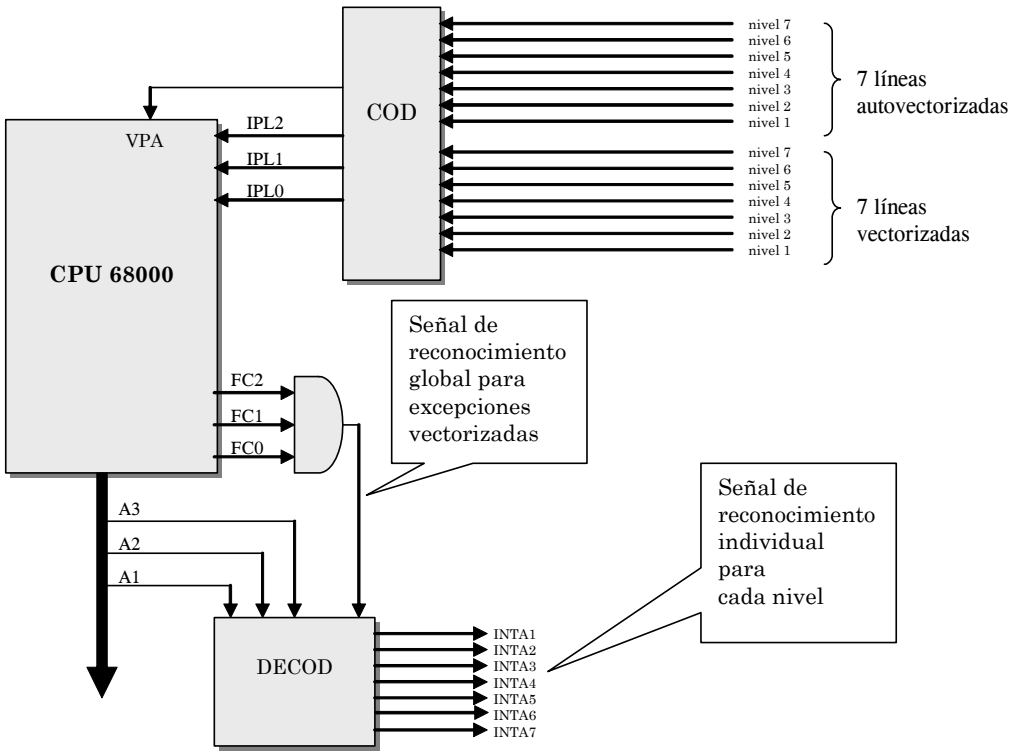
En el *MC 68000* las interrupciones se generalizan al concepto de *excepción*, que incluye las interrupciones hardware de *E/S* propiamente dichas, las producidas por eventos internos en la *CPU*, y las producidas por software.

5.5.2.1. Líneas de interrupción

Las excepciones de E/S llegan a la *CPU* en forma codificada a través de tres líneas externas: *IPL2*, *IPL1*, *IPL0*, que codifican la ausencia de excepción (*111*), y *7* niveles con prioridad (*000* el más prioritario y *110* el menos prioritario).



Una cuarta línea, *VPA*, determina si los siete niveles de las líneas *IPLi* codifican una excepción autovectorizada (*VPA = 0*) o vectorizada (*VPA = 1*). En el primer caso el vector de excepción es fijo para cada nivel, y en el segundo lo determina el periférico y lo transmite por las *8* líneas menos significativas del bus de datos cuando le llega la señal de reconocimiento procedente de la *CPU*, que corresponde al valor *111* de las señales de estado *FC2, FC1, FC0*. El número de la línea vectorizada reconocida lo genera la *CPU* a través de las líneas del bus de direcciones *A3, A2, A1*. Por ello, es necesario complementar con circuitería lógica externa el sistema, tal como aparece en el siguiente esquema:



5.5.2.2. Enmascaramiento

Los bits I2,I1,I0 del registro de estado codifican los niveles inhibidos por la CPU, de acuerdo a la siguiente tabla:

I2 I1 I0	Niveles inhibidos
0 0 0	Ninguno
0 0 1	Nivel 1
0 1 0	Niveles 1 al 2
0 1 1	Niveles 1 al 3
1 0 0	Niveles 1 al 4
1 0 1	Niveles 1 al 5
1 1 0	Niveles 1 al 6
1 1 1	Niveles 1 al 6
El nivel 7 no es enmascarable	

T		S		I_2	I_1	I_0		**X**	**N**	**Z**	V	**C**

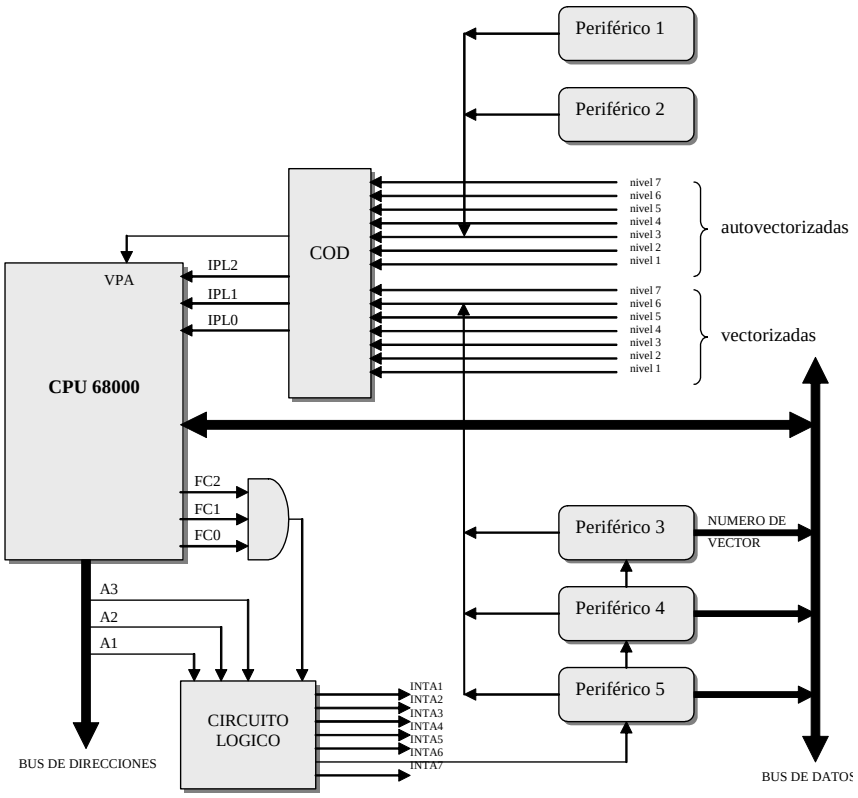
5.5.2.3. Vectores de excepción

Los vectores de excepción son posiciones de memoria donde la CPU obtiene la dirección de la rutina de tratamiento de la excepción. En el MC 68000 existen 255 vectores que se agrupan en la tabla de vectores de excepción ubicada en la zona más baja de memoria (desde la dirección 0 a la 1020 en decimal). Cada vector ocupa 2 palabras (4 bytes) excepto el vector de RESET que ocupa 4 palabras, 2 para el CP y las otras 2 para el SP, y se identifica con un número de 8 bits (en decimal del 0 al 255, excepto el 1, que corresponde a la segunda mitad del 0, que es el de RESET). La dirección real se obtendrá multiplicando por 4 el número de vector. El número de vector lo genera la CPU (excepciones autovectorizadas) o el periférico (excepciones vectorizadas).

Tabla de vectores de excepción	
Nº de vector	Asignación
0	RESET (CP inicial)
1	RESET (SP inicial)
2	error del bus
3	dirección no existente
4	instrucción ilegal
5	división por cero
6	instrucción CHK (entero fuera de rango)
.	
25	autovector nivel 1
26	autovector nivel 2
27	autovector nivel 3
28	autovector nivel 4
29	autovector nivel 5
30	autovector nivel 6
31	autovector nivel 7
.	
.	
64	
.	
255	192 vectores de excepción de usuario

5.5.2.4. Ejemplo de conexión de periféricos

Para finalizar veamos el esquema completo de conexión de periféricos al sistema de excepción del MC 68000. Se han conectado tres periféricos de forma encadenada (*daisy chain*) a la línea vectorizada de nivel 3.

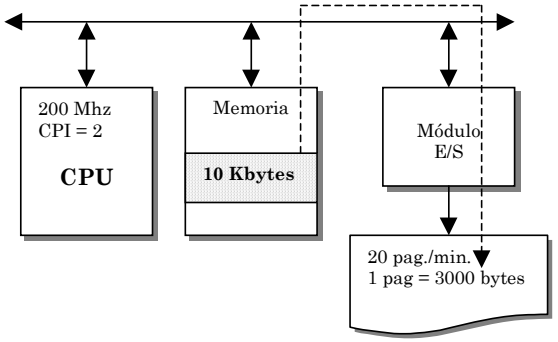


6. E/S por acceso directo a memoria (DMA): motivación

La E/S por interrupciones ocupa menos tiempo de CPU que la E/S controlada por programa. Sin embargo, con cualquiera de las dos alternativas las transferencias de datos deben pasar a través de la *CPU*. Esto significa que la velocidad de las transferencias está limitada por la velocidad a la que la *CPU* atiende el dispositivo periférico, ya que tiene que gestionarlas ejecutando una serie de instrucciones. En la E/S controlada por programa la *CPU* está dedicada exclusivamente a la *E/S*, transfiriendo los datos a relativa velocidad, pero al precio de dedicarse a ello a tiempo completo. Las interrupciones liberan en buena medida a la *CPU* de la gestión de las transferencias, a costa de disminuir su velocidad, pues la rutina de servicio de las interrupciones contiene por lo general instrucciones ajenas a la propia transferencia. Ambos procedimientos manifiestan, pues, ciertas limitaciones que afectan a la actividad de la *CPU* y la velocidad de transferencia.

Para poner de manifiesto de forma cuantitativa las limitaciones de la *E/S* programada y por interrupción, consideremos los dos siguientes supuestos:

Supuesto 1: conexión de un periférico lento, una impresora láser. La *CPU* opera a *200 Mhz*. y su *CPI* vale *2*. La impresora opera a *20* páginas/minuto, con *3.000* caracteres (*bytes*) por página. Se trata de imprimir un bloque de *10 Kbytes* ubicado en la memoria.



E/S programada

La impresora imprime 20 * 3.000 = 60.000 caractéres/minuto = 1.000 caracteres/segundo = 1 Kbyte/s. Luego los 10 Kbytes los imprimirá en 10 segundos.

La CPU está ocupada durante 10 segundos

E/S por interrupción

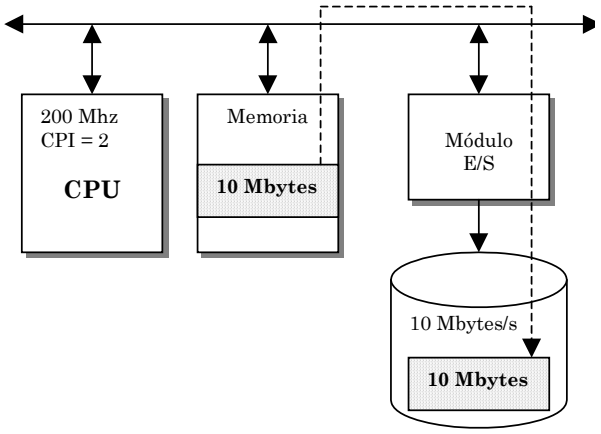
Para este caso habrá que conocer el número de instrucciones que se ejecutan dentro de la rutina de tratamiento de la interrupción. Supondremos que son 10 las instrucciones (3 para salvar contexto, 2 para comprobar el estado y bifurcar, 1 para la transferencia, 3 para restaurar contexto, y 1 para retorno de interrupción, RTI). La impresora genera una interrupción cada vez que está preparada para recibir un nuevo caracter.

Para transferir los 10 Kbytes se producirán 10.000 interrupciones, lo que significa ejecutar 100.000 instrucciones. Una instrucción tarda en ejecutarse 2 * 5 ns = 10 ns, ya que el tiempo de ciclo Tc = 1/Frecuencia = $1/200*10^6$ seg. = $1/200*10^6 * 10^{-9}$ ns = 5 ns.

Luego la CPU se ocupa durante $10^5 * 10$ ns = 10^6 ns = $10^6 * 10^{-9}$ s. = 0,001 s. = 1 milisegundo.

La E/S por interrupción ha reducido en 10.000 veces el tiempo que la CPU está ocupada en atender la impresora. Sin embargo, la velocidad de la operación de E/S no ha cambiado, como era de esperar al estar dominada por la velocidad del periférico.

Supuesto 2: conexión de un periférico rápido, un disco magnético. La CPU opera igual que en caso anterior, es decir, a 200 Mhz y su CPI vale 2. El disco opera a una velocidad de transferencia de 10 Mbytes/s. Se trata en este caso de transmitir un bloque de 10 Mbytes de la memoria al disco.



a) E/S programada

A la velocidad de *10 Mbytes/s* el disco tarda *1* segundo en recibir los *10 Mbytes*, y como por *E/S* programada la *CPU* envia un byte cada vez que el disco está preparado para recibirlo, la *CPU* está ocupada en la tranferencia durante *1 segundo*.

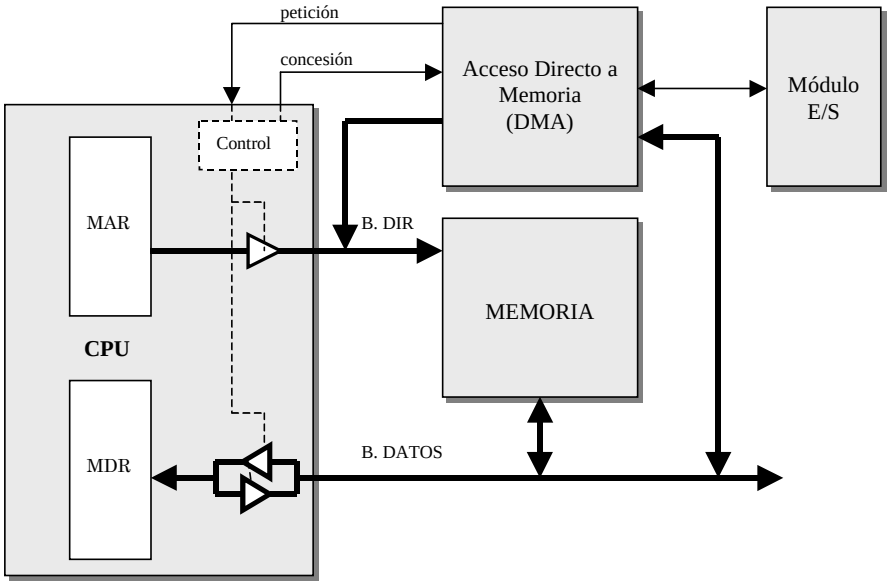
b) E/S por interrupción

Seguimos suponiendo que la rutina de tratamiento ejecuta *10* instrucciones. Como ahora se transmiten 10^7 *bytes* se producirán otras tantas interrupciones, lo que significa que la CPU ejecuta en toda la operación $10^7 * 10 = 10^8$ instrucciones. Por tanto el tiempo total de ocupación de la *CPU* en la transferencia de E/S será 10^8 instrucciones * *10* ns/instrucción = 10^9 *ns = 1 segundo*.

Luego en el supuesto 2 las interrupciones no ahorran tiempo de CPU en la operación de E/S frente a la *E/S* programada. La razón está en que se ha llegado al límite de velocidad que es capáz de soportar la línea de interrupción del supuesto (*1 byte* cada *100* ns). Por encima de los *10 Mbytes/s* la *E/S* por interrupción perdería datos puesto que se presentaría un nuevo byte antes de haber finalizado la rutina de tratamiento del byte anterior. La E/S programada todavía admitiría mayor velocidad puesto que no tendría que ejecutar las instrucciones de guarda y restauración del contexto ni la instrucción *RTI*. El bucle de la *E/S* programada ejecutaría sólo *3* instrucciones (*2* para comprobar el estado y bifurcar, *1* para la transferencia), es decir tardaría *30 ns* en lugar de los *100 ns* que tarda la rutina de tratamiento de la interrupción. Por tanto podría llegar hasta el límite de *30 ns/byte*, es decir, *33 Mbytes/segundo*

Por tanto, cuando se tienen que transferir grandes cantidades de datos y a una elevada velocidad, es necesario disponer de una técnica que realice de forma más directa las transferencias entre la memoria y el dispositivo periférico, limitando al máximo la intervención de la *CPU*. Esta técnica se conoce con el nombre de *acceso directo a memoria* (*DMA: Direct Memory Access*).

Básicamente se trata de un módulo con capacidad para leer/escribir directamente en la memoria los datos procedentes/enviados de/a los dispositivos periféricos. Para ello solicita la correspondiente petición a la *CPU*. Antes de que la *CPU* conceda acceso a memoria al *DMA*, pone en estado de alta impedancia su conexión a los buses del sistema (datos, direcciones y *R/W*), lo que es equivalente a desconectarse de la memoria durante el tiempo que es gestionada por el *DMA*. Cuando finaliza la operación de *E/S* el *DMA* genera una interrupción y la *CPU* vuelve a tomar control de la memoria. De esta forma la velocidad de transferencia sólo estarán limitadas por el ancho de banda de la memoria.



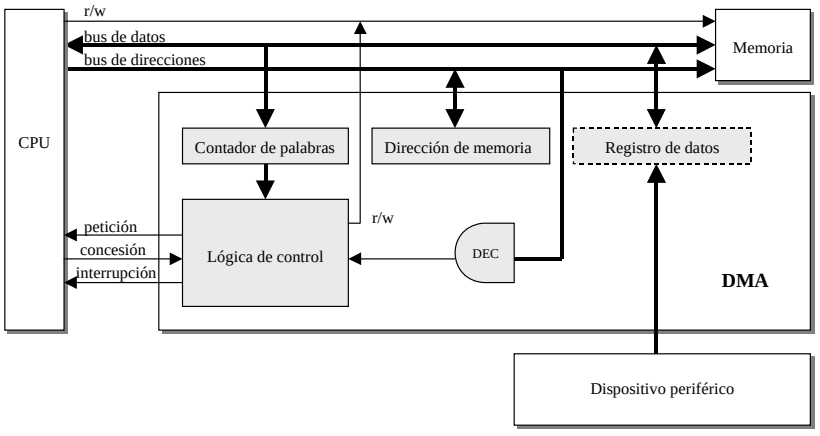
6.1. Estructura y funcionamiento de un controlador de DMA

Para gestionar las transferencias de información, un controlador de *DMA* dispone de *3* registros: *datos, dirección y contador de palabras.*

El *registro de dirección* almacena la dirección de la siguiente palabra a transmitir, incrementándose de forma automática después de cada transferencia. El *contador de palabras* almacena el número de palabras que quedan por transmitir y también se decrementa automáticamente después de cada transferencia. La *lógica de control* comprueba el contenido del contador de palabras y finaliza la operación cuando alcanza el valor *0*. Un *decodificador* se encargará de identificar la dirección de memoria asignada al *DMA*. La *CPU* debe enviar, pues, la siguiente información al *DMA* cuando quiera realizar una operación de *E/S*:

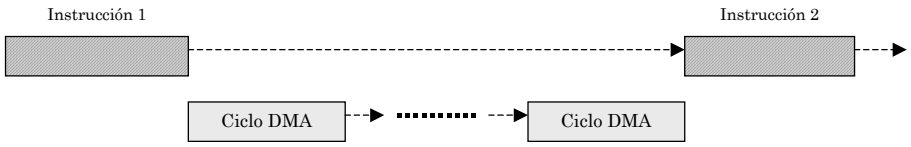
- El sentido de la operación de E/S: lectura o escritura
- La dirección del periférico
- La posición de memoria donde comienza el bloque a leer o escribir
- El número de palabras que componen el bloque

Transmitida esta información, la *CPU* pasa a realizar otra tarea, delegando totalmente la operación de *E/S* al *DMA*. El DMA transfiere directamente, palabra a palabra, el bloque de datos entre el periférico y la memoria, sin pasar por la *CPU*. Cuando la transferencia finaliza el *DMA* envía una señal de interrupción a la *CPU*.

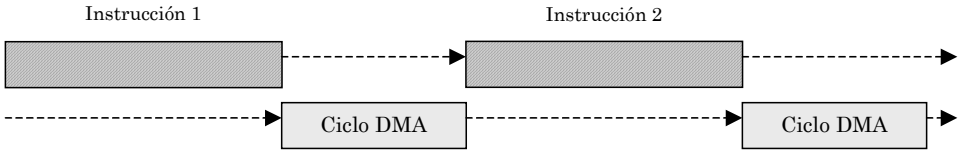


Para poder transferir a/desde la memoria, el DMA necesita controlar el bus durante un tiempo suficiente para completar la transferencia. Sin embargo, este tiempo no tiene que ser continuo, puede fraccionarse en pequeños intervalos que se alternan con la CPU. Existen diferentes alternativas en la forma de controlar el bus. Cada alternativa supone un compromiso diferente entre velocidad de transferencia y actividad de la CPU. El empleo de una alternativa en concreto dependerá de las prestaciones que se deseen y de las características del procesador que se utilice. Son las siguientes:

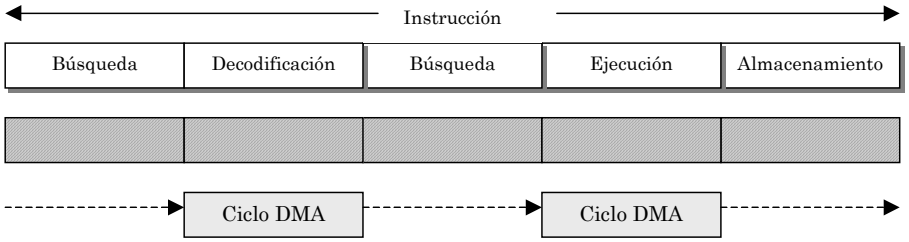
Ráfagas: el *DMA* toma control del bus y no lo libera hasta haber transmitido un bloque de datos completo. Con este método se consigue la mayor velocidad de transferencia pero puede dejar sin actividad a la *CPU* durante períodos grandes de tiempo.



Robo de ciclos*:* el DMA toma control del bus y lo retiene durante un solo ciclo. Transmite una palabra y libera el bus. Es la forma más usual de transferencia y en ella se dice que el DMA roba ciclos a la CPU.



Transparente: el *DMA* accede al bus sólo en los ciclos en los que la *CPU* no lo utiliza. Y esto ocurre en diferentes fases de ejecución de las instrucciones. De esta forma la ejecución del programa no se ve afectado en su velocidad de ejecución.



En los sistemas con memoria cache también se puede aprovechar el acceso de la *CPU* a la cache para que el *DMA* pueda acceder simultáneamente a la memoria principal. En resumen, los pasos que se siguen en la transferencia mediante *DMA* son:

1. La *CPU* ejecuta las instrucciones de *E/S* que cargan los registros de dirección y contador de palabras del controlador de *DMA*. El registro de dirección debe contener la dirección base de la zona de memoria principal que se va a utilizar en la transferencia de datos. El registro contador de palabra almacena el número de palabras que se transfieren desde/hacia la memoria.
2. Cuando el controlador de *DMA* está preparado para transmitir datos, activa la línea de *petición*. La *CPU* espera a un punto de concesión del *DMA*, renuncia al control de los buses de datos y direcciones y activa la línea de reconocimiento de *DMA*.
3. El *DMA* transfiere directamente los datos a/desde memoria principal por alguno de los métodos que se acaban de ver. Después de transferir una palabra, el registro de

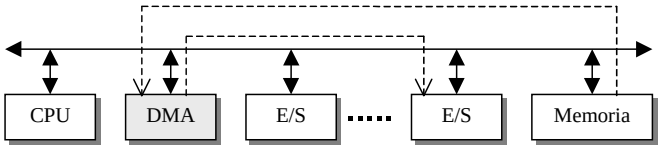
dirección y el registro contador de palabras del controlador se incrementa y decrementa respectivamente.

4. Si el contenido del registro contador de palabra no es *0*, pero el periférico no está preparado para enviar o recibir los siguientes datos, el *DMA* devuelve el control a la *CPU* liberando el bus del sistema y desactivando la línea de *petición*. La *CPU* responde desactivando la línea de reconocimiento de *DMA* y continuando con su operación normal.
5. Si el contenido del contador de palabras es 0, el controlador de *DMA* renuncia al control del bus del sistema y envía una señal de interrupción a la *CPU*.

El DMA, se puede configurar de diferentes formas:

6.1.1. Bus único (*DMA* independiente)

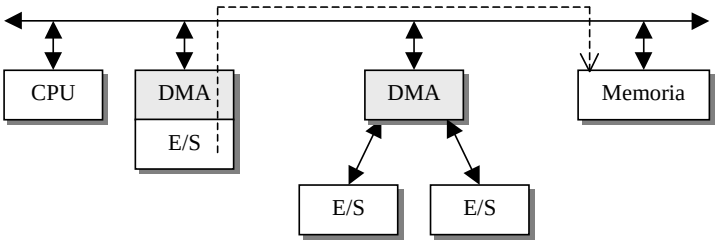
Todos los módulos comparten el bus del sistema. El *DMA*, que actúa en sustitución de la *CPU*, intercambia datos entre la memoria y el periférico utilizando un procedimiento análogo al de *E/S* controlada por programa, es decir, hace de intermediario entre ambas unidades.



Esta configuración, aunque puede ser muy económica, es claramente poco eficaz, ya que cada transferencia de una palabra consume 2 ciclos del bus.

6.1.2. Integración de funciones *DMA-E/S*

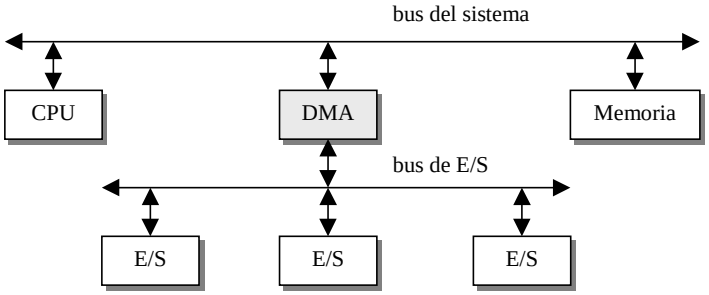
Reduce a 1 el número de ciclos de utilización del bus. Esto significa que hay un camino entre el controlador de *DMA* y uno o más controladores de E/S que no incluyen al bus del sistema.



La lógica del *DMA* puede ser una parte de un controlador de *E/S* o puede ser un módulo independiente que controla a uno o más controladores de *E/S*.

6.1.3. Bus de *E/S* conectado al *DMA*

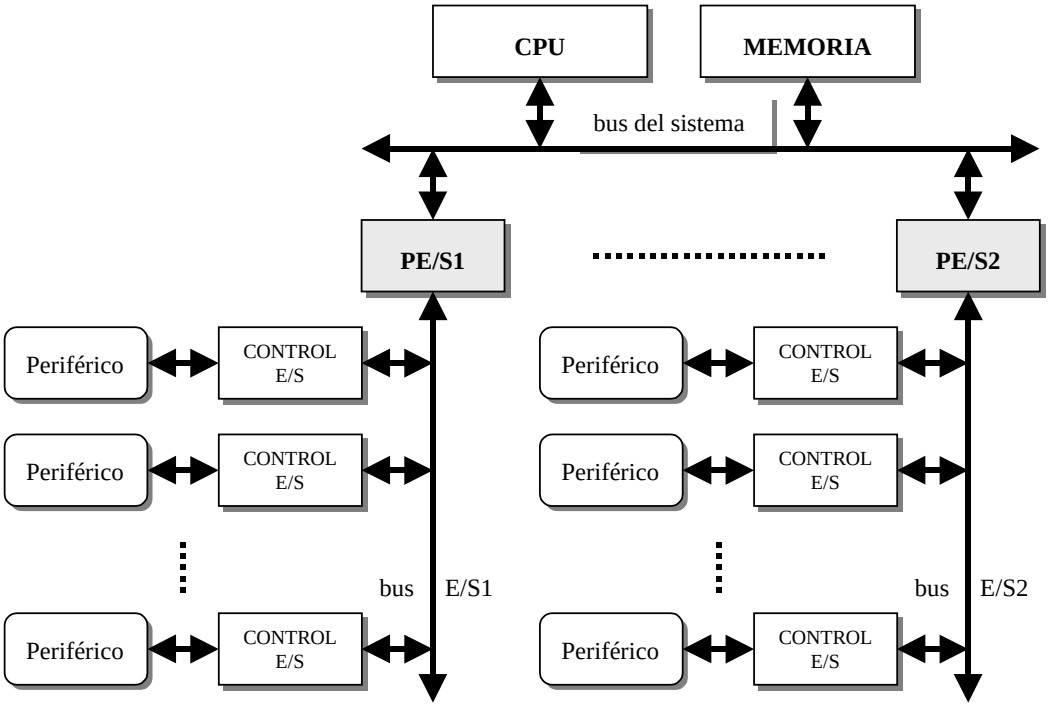
El concepto anterior se puede generalizar si se utiliza un bus de *E/S* para conectar los controladores de E/S al *DMA*. Esta alternativa reduce a una el número de interfaces de E/S en el *DMA*, y proporciona una configuración fácilmente ampliable.



7. Procesadores de *E/S*: tipos y estructura.

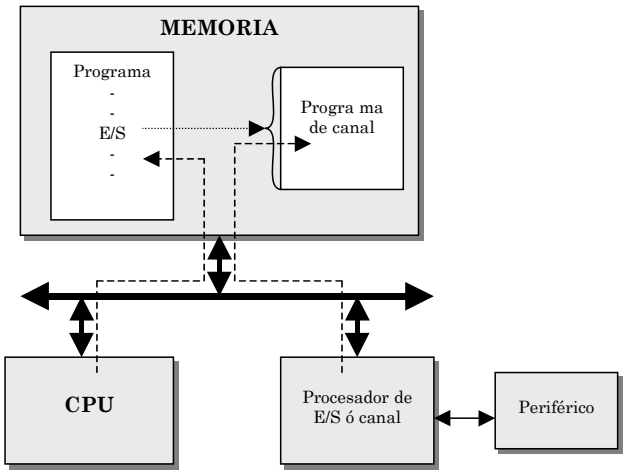
Como hemos visto en los 2 temas anteriores y en los primeros apartados de este tema, la E/S de los computadores ha experimentado una continua evolución. Comenzó con la E/S controlada por programa. Le siguió la introducción de las interrupciones para evitar que la *CPU* malgastase su tiempo esperando la realización de las operaciones de *E/S*, aumentando el rendimiento global del sistema. Posteriormente se introduce en el controlador de *E/S* la capacidad para acceder directamente a memoria a través del *DMA*. De esta forma se pueden transferir bloques de datos a/desde memoria sin intervención de la *CPU*, excepto al comienzo y al final de la transferencia.

Pero este proceso de evolución de la E/S no termina en el *DMA*. En el paso siguiente se potencia la capacidad del controlador de E/S hasta convertirlo en un procesador con un conjunto de instrucciones especializadas en operaciones de E/S. La *CPU* traduce las operaciones de E/S en órdenes dirigidas al procesador de *E/S* (*PE/S*) para que ejecute un programa de *E/S* residente en memoria. El *PE/S* ejecuta ese programa sin la intervención de la *CPU* y sólo se interrumpe cuando se ha ejecutado la secuencia completa. A los procesadores de *E/S* se les suele denominar también con el nombre de canales de E/S. No obstante los dos términos se emplean de forma indistinta.

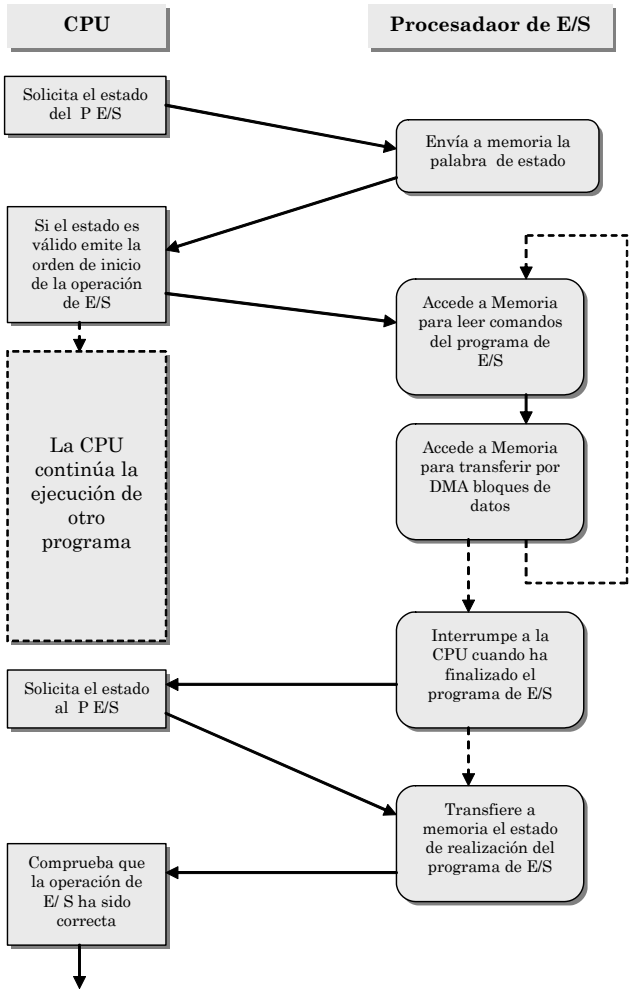


Un PE/S representa, pues, una extensión del concepto *DMA*. Un *PE/S* tiene la capacidad de ejecutar instrucciones de *E/S*, lo que le da un control completo sobre dicha operación. En los computadores que incluyen *PE/S*, la CPU no ejecuta las instrucciones de E/S, éstas se almacenan en memoria principal para ser ejecutadas por un *PE/S*. Así, la *CPU* inicia una transferencia de E/S al dar una orden al *PE/S* para que ejecute un programa en memoria. El programa especifica entre otras cosas las siguientes:

- El periférico (o periféricos) que interviene en la operación de *E/S*.
- Las zonas de memoria utilizadas en la transferencia
- Las acciones a realizar si se producen ciertas condiciones de error durante la transferencia.



Las comunicaciones entre la *CPU* y el procesador de *E/S* a lo largo de la ejecución de una operación de *E/S* podemos resumirlas en el siguiente esquema:



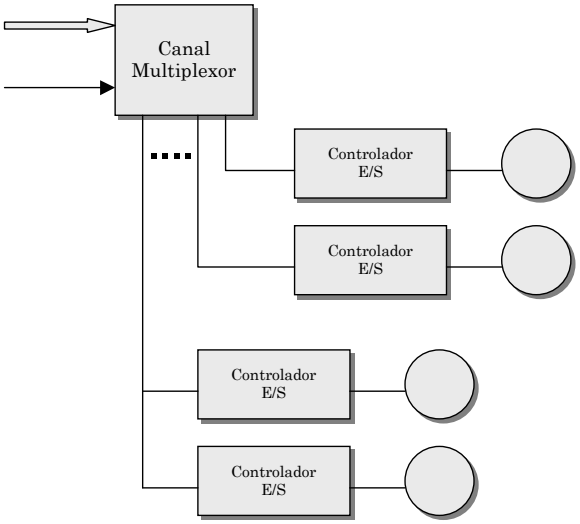
Existen tres tipos de *PE/S* o canales:

- Canales multiplexores

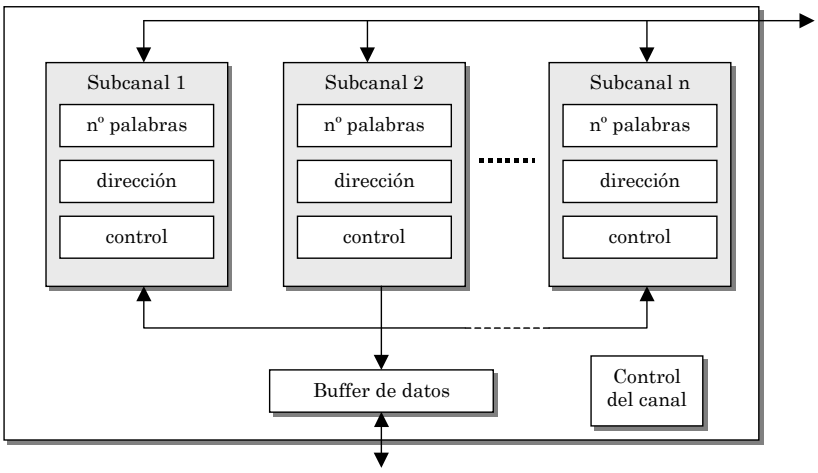
- Canales selectores
- Canales multiplexores por bloques

7.1. Canal multiplexor

Un canal multiplexor se utiliza para conectar dispositivos de velocidad baja y media. Son capaces de transferir datos a velocidades mayores que las de los dispositivos individuales conectados al canal. Esto les permite operar en forma multiplexada sobre varios dispositivos de E/S.

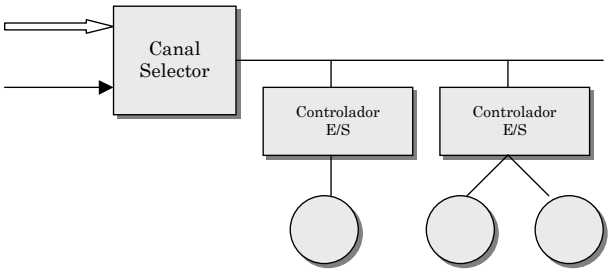


En la siguiente figura aparece la organización interna de un canal multiplexor. Los parámetros correspondientes a la operación de cada dispositivo se ubican en posiciones fijas de la memoria principal. Cuando el canal direcciona un dispositivo particular, busca previamente los parámetros en memoria y los lleva a los registros de uno de los subcanales. Cuando el dispositivo se desconecta, los valores actualizados se llevan de nuevo a memoria. El número de subcanales disponibles en el sistema determina el número máximo de operaciones de E/S que pueden mantenerse simultáneamente.



7.2. Canal selector

Un canal selector controla múltiples dispositivos de alta velocidad. En cualquier instante de tiempo está dedicado a la transferencia de datos con uno sólo de estos dispositivos. El PE/S selecciona un dispositivo y efectúa la transferencia de datos completa.



7.3. Canal multiplexado por bloques

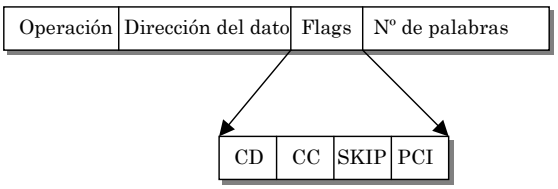
Este tipo de canal combina las características de los canales multiplexores con las de los selectores. Permite la multiplexación de varias operaciones de E/S de alta velocidad, bloque por bloque. Las ventajas obtenidas a través del empleo de canales multiplexados por bloques, en comparación con las de los canales selectores, surgen de la naturaleza de la operación de dispositivos de E/S tales como discos y cintas magnéticas. Una operación común con una unidad de disco puede requerir la siguiente secuencia de comandos:

- Localización de pista (mover la cabeza de lectura/escritura a una pista dada).
- Búsqueda de un sector específico.
- Inicio de la lectura de los datos contenidos en el sector.

Debido al considerable retraso mecánico involucrado en las primeras dos operaciones, no es adecuado mantener ocupado el canal durante esta secuencia, lo que sería el caso con un canal selector. Un canal multiplexor por bloques puede enviar a la unidad de disco la orden de *localización de pista* y después desconectarse para dar servicio a otros dispositivos de E/S. Cuando se completa la operación de *localización de pista*, el canal se vuelve a conectar, envía la orden de *búsqueda del sector* y se desconecta de nuevo. Al terminar *la búsqueda de sector*, el canal inicia la *transferencia del sector* y permanece conectado hasta que finaliza la operación. El uso de un canal multiplexor por bloques da como resultado una mejora importante en la productividad, en comparación con los canales selectores.

7.4. Programas de operación de los procesadores de E/S.

En un *DMA* simple los parámetros relacionados con las transferencias de datos se cargan en los registros correspondientes utilizando instrucciones de E/S. En máquinas grandes que utilizan canales de E/S, esto se sustituye por el concepto de programa de canal. El canal, como se ha dicho antes, es un pequeño procesador que tiene acceso a la memoria principal. Es capaz de ejecutar un conjunto limitado de instrucciones, conocidas como *comandos de canal.* Estos comandos especifican aquellos parámetros que el canal necesita para controlar los dispositivos de E/S y realizar operaciones de transferencia de datos. Un programa de canal es una secuencia de estos comandos almacenada en la memoria principal. La *CPU* inicia una operación de E/S con la emisión de una orden especial al canal que hace que el canal empiece a buscar en memoria y ejecutar los comandos que constituyen el programa de canal. El programa de canal consta de una serie de *palabras de comando de canal (CCW: channel command words)* cuyo formato simplificado aparece en la siguiente figura:



Operación Este campo corresponde al código de operación de las instrucciones máquina normales. Especifica la operación que debe realizar la *CCW*. La unidad de control decodifica este campo y envía las señales adecuadas de control al dispositivo. Existen varias operaciones, las más importantes son las siguientes:

> *Lectura:* el canal transfiere a memoria principal un bloque de palabras de tamaño especificado en el campo *nº de palabras*, en orden ascendente de direcciones, empezando en la dirección especificada en el campo *dirección del dato*.
>
> *Escritura:* el canal transfiere datos de memoria principal al dispositivo. Las palabras se transfieren en el mismo orden que en la operación de *lectura*.
>
> *Control:* se utiliza esta orden para enviar instrucciones específicas al dispositivo de E/S, como rebobinar una cinta magnética, etc.
>
> *Bifurcación:* cumple en el programa de canal la misma función que una instrucción de salto en un programa normal. El canal ejecuta las *CCW* en secuencia, salvo cuando aparece una *CCW* de este tipo, que utiliza el campo *dirección del dato* como la dirección de la siguiente *CCW* a ejecutar.

Flags Los bits de este campo indican lo siguiente:

CC (Encadenamiento de comando): cuando este bit está a 1, indica al canal que la siguiente *CCW* especifica una nueva operación de *E/S* que debe realizarse con el mismo dispositivo. El canal ejecuta primero la operación de *E/S* especificada en la *CCW* que tiene a 1 el *flag CC*. Después ejecuta la siguiente sobre el mismo dispositivo. Desde el punto de vista del dispositivo, las dos operaciones resultantes de la primera y segunda *CCW* aparecen como dos comandos de *E/S* separados.

CD (Encadenamiento de datos): una *CCW* con el bit *CD* a *1*, indica al canal que la siguiente CCW contiene una nueva *dirección del dato* y un nuevo *nº de palabras*. Éstos deben utilizarse para transferir un segundo bloque de datos hacia o desde el dispositivo de *E/S*, mediante la orden especificada por la *CCW* actual. Cuando el canal termina la transferencia de datos especificada en la *CCW* con el bit *CD* a *1*, no corta la conexión con el dispositivo de *E/S*, continúa la transferencia utilizando la *dirección* y el *nº de palabras* de la siguiente *CCW*. Con el encadenamiento de datos, los datos transferidos por la primera y segunda *CCW* aparecen como un bloque para el dispositivo de *E/S*. Esto facilita las transferencias entre un dispositivo y posiciones no contiguas de la memoria principal.

Para poner de manifiesto la diferencia entre encadenamiento de comandos y de datos veamos lo que ocurre con una transferencia de dos bloques de datos de la memoria principal a una cinta magnética. Si se utiliza encadenamiento de comandos, los dos bloques se escribirán en la cinta separados por un separador de registros que la unidad de cinta inserta de forma automática al final de una operación de *E/S* especificada por una *CCW* en donde *CD = 0*. En cambio, el encadenamiento de datos provoca que los dos bloques se unan en un solo registro en la cinta. Una CCW con los *flags CD* y *CC* a *0* significa el final del programa de canal.

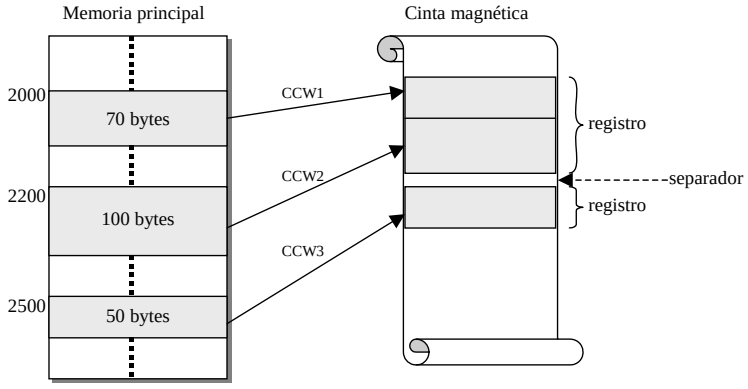
SKIP (Salto): este bit, cuando está a 1, hace que el programa de canal salte un número de palabras igual al especificado en el campo *nº de palabras*. Cuando se utiliza con la orden de *lectura,* este *flag* hace que los datos se lean del dispositivo sin que se transfieran a la memoria principal.

PCI (Interrupción controlada por programa): el canal produce una interrupción cuando ejecuta una *CCW* con el *flag PCI* a *1*. Si esta *CCW* va precedida por otra *CCW* con encadenamiento de comandos, la interrupción se genera después de que concluyan todas las transferencias de datos.

Ejemplo 1

En la siguiente figura se muestra un ejemplo de programa que transfiere tres bloques de datos de la memoria principal a una cinta magnética. *CCW1* lee 70 bytes de memoria, a partir de la posición *2000*, y los escribe en cinta. La operación de *Escritura* y la unidad de cinta a utilizar se especifican en el campo de operación. Puesto que el bit *CD* está *1*, *CCW2* debe interpretarse como una continuación de *CCW1*. Se realiza la misma operación de E/S, esto es, una *Escritura* en el mismo dispositivo, pero con los datos especificados en *CCW2*. Obsérvese que en CCW2 no se tienen en cuenta los bits de operación. Se transfieren a la cinta 100 bytes de memoria, a partir de la posición 2.200, como continuación del registro iniciado con *CCW1*. En *CCW2* el bit *CC* está 1, lo que significa que sigue otra *CCW* (*CCW3*), pero que no está relacionada con la *CCW2*. Por lo tanto, al finalizar CCW2, se inserta un espacio de separación de registros en la cinta. De nuevo, *CCW3* especifica una *Escritura* en la misma cinta. El resultado es una transferencia a la cinta de *50 bytes* a partir de la posición *2.500* de memoria. Puesto que los *flags CD* y *CC* valen *cero*, se trata del final del programa de canal.

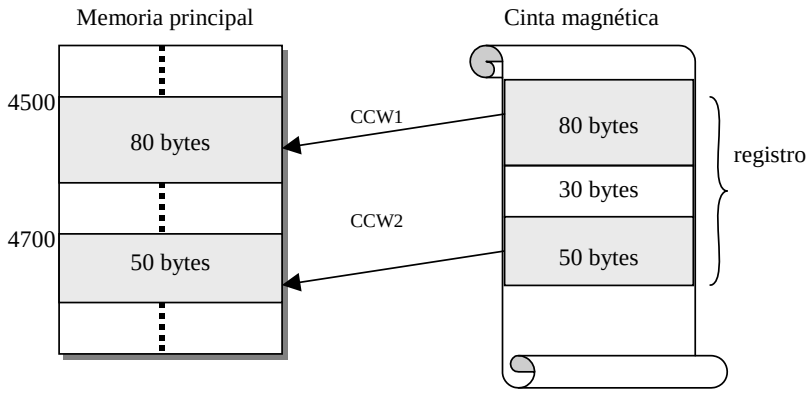
OPERACION	DIRECCION	FLAGS				Nº PALABRAS
		CD	CC	SK	PCI	
CCW1 ESCRITURA	2000	1	0	0	0	70
CCW2	2200	0	1	0	0	100
CCW3 ESCRITURA	2500	0	0	0	0	50



Ejemplo 2

En la siguiente figura aparece otro programa de canal que muestra el empleo del *flag* de salto (*SKIP*). La primera *CCW* (CCW1) transfiere 80 bytes de la unidad de cinta especificada en el campo de *operación* a la memoria, a partir de la posición *4.500*. Puesto que *CD* = 1, *CCW2* realiza la misma operación. Sin embargo, como en *CCW2* el *flag de salto* está 1, significa que el canal no transmitirá los siguientes *30 bytes* que lea de la cinta. Puesto que *CD* sigue *1*, continúa la misma operación de *lectura* con *CCW3*, provocando la transferencia de los siguientes *50 bytes* a la memoria, a partir de la posición *4.700*. Por lo tanto, el programa transfiere dos bloques de datos a la memoria principal desde un registro de la cinta, con un salto de *30 bytes* a la mitad del registro.

OPERACION	DIRECCION	FLAGS				Nº PALABRAS
		CD	CC	SK	PCI	
CCW1 LECTURA	4500	1	0	0	0	80
CCW2		1	0	1	0	30
CCW3	4700	0	0	0	0	50

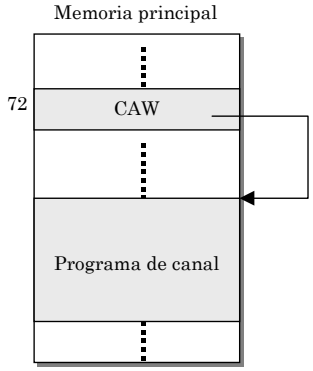


Inicio y control de los programas de canal

Hemos visto como se utilizan los programas de canal para realizar operaciones de E/S. Estos programas residen en la memoria principal del computador y se ejecutan en el canal. Vamos a examinar ahora la forma en que la CPU inicia y supervisa las operaciones de E/S, es decir, el programa de canal. En el IBM S/370 existen cuatro instrucciones máquina que la CPU puede utilizar para estos fines. Son las siguientes:

START I/O	Inicia una operación de *E/S*. El campo de dirección de la instrucción se emplea para especificar el canal y el dispositivo de *E/S* que participa en la operación.
HALT I/O	Finaliza la operación del canal.
TEST CHANNEL	Prueba el estado del canal.
TEST I/O	Prueba el estado del canal, el subcanal y el dispositivo de *E/S*.

Una operación de E/S se inicia con la instrucción START I/O. La ubicación del programa de canal en la memoria principal viene definida en la *palabra de dirección de canal* (*CAW*: *Channel A ddress word),* que siempre está almacenada en la posición *72* de la memoria principal.



La *CAW* debe cargarse, pues, en la posición 72 antes de la ejecución de la instrucción *START E/S*. El dispositivo indica su disponibilidad para participar en la operación de *E/S* devolviendo su dirección al canal. A partir de este momento el dispositivo permanece conectado al canal hasta que finaliza la operación de E/S. Establecida la disponibilidad del dispositivo, el canal envía una señal a la *CPU*. En este momento finaliza la instrucción *START E/S*, y la *CPU* deja de ocuparse de la operación de *E/S*. Al finalizar la instrucción de *START E/S*, el canal comienza a ejecutar el programa de canal.