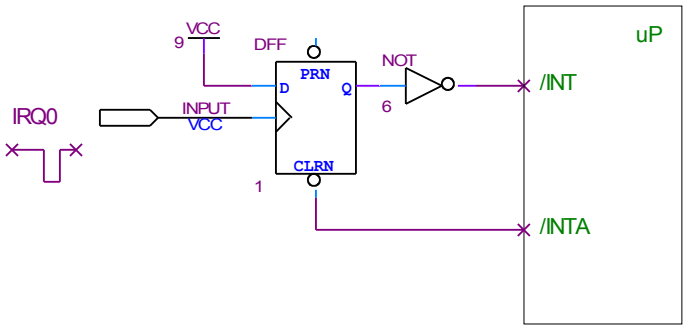
INTERRUPCIONES

Cuando la E/S es controlada por programa, el FF de estado asociado a una transferencia de datos condicional se utiliza para sincronizar al microprocesador con el dispositivo externo. Dicho FF de estado es verificado por el microprocesador a través de un puerto de entrada en un bucle de programa a la espera de un pedido de atención por parte del dispositivo. Este bucle de espera insume una parte apreciable del tiempo de procesador, incluso cuando el dispositivo no solicita ser atendido, afectando negativamente el *throughput* o cantidad total de información procesada o comunicada por unidad de tiempo del sistema. Desde este punto de vista es deseable entonces que el pedido de atención del dispositivo externo sea monitoreado automáticamente de alguna forma y se utilice tiempo de procesador solamente cuando se atiende al dispositivo después que se ha detectado una solicitud de atención.

Esto es precisamente lo que sucede cuando se utiliza interrupciones. Básicamente una interrupción es un mecanismo por el cual se invoca a una subrutina en respuesta a un evento de petición iniciado por un dispositivo hardware externo. A la subrutina se le llama rutina de atención o rutina de servicio a la interrupción. Su ejecución queda intercalada entre dos instrucciones consecutivas del programa principal.



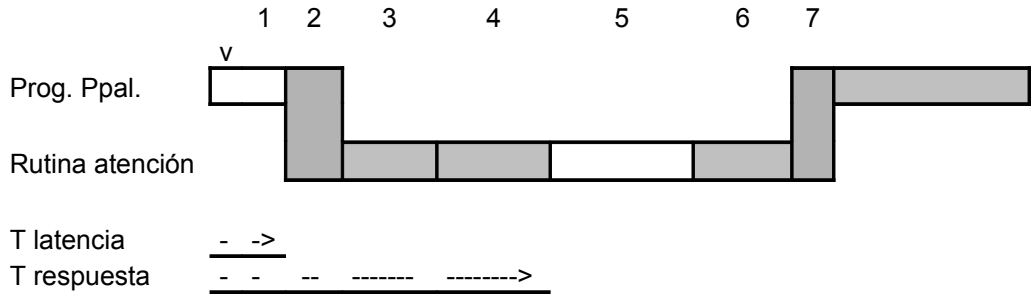
La figura muestra el circuito necesario para que un dispositivo externo solicite interrupciones a un microprocesador. El dispositivo externo activa la solicitud poniendo a "1" el *flip-flop de petición* de interrupción. Este flip-flop funciona en forma similar al flip-flop de estado en una transferencia condicional, salvo que en lugar de ser leído por el microprocesador a través de un puerto de entrada está conectado directamente a un pin del microprocesador dedicado a recibir las solicitudes de interrupción. El procesador consulta la entrada de interrupción al final de cada instrucción. Cuando se activa dicha entrada, el microprocesador completa la instrucción que estaba siendo ejecutada, reconoce la solicitud a través de alguna señal observable desde el exterior (/INTA en la figura) y transfiere el control del programa a la rutina de atención a la interrupción para que ejecute las tareas solicitadas por el dispositivo externo. Cuando el periférico es atendido debe borrarse el flip-flop de petición, o bien con la señal de reconocimiento como en el ejemplo o bien mediante un pulso de selección de dispositivo activado explícitamente dentro de la rutina de atención. Una vez finalizadas sus tareas la rutina de atención retorna a la siguiente instrucción del programa principal con una instrucción RET o alguna de sus variantes.

Secuencia de atención

A continuación se analiza más en detalle la secuencia de eventos que se producen cuando el dispositivo externo activa su flip-flop de petición. La secuencia se describe para una situación genérica planteando variantes que dependerán de la complejidad del sistema y del microprocesador particular utilizado.

1. Se **completa la ejecución de la instrucción**. Los procesadores no aceptan ser interrumpidos en medio de una instrucción. En el caso del Z80 las entradas de interrupción son sensadas en el último período T del último ciclo M de cada instrucción.
2. Se ejecuta un **ciclo de reconocimiento** de la interrupción. Durante el mismo:
   - Se genera una señal hardware de reconocimiento de interrupción para avisar al periférico que va a ser atendido.
   - Se salva el contador de programa en el stack para poder retornar a la siguiente instrucción.
   - Se transfiere el control a la dirección de comienzo de la rutina de atención.
   - En algunos casos antes de transferir el control se identifica cuál fue el dispositivo que pidió interrupción para determinar a cuál es la rutina de atención que le corresponde.
   - En algunos casos en este paso se borra el flip-flop de petición utilizando la señal hardware de reconocimiento.

3. Al comienzo de la rutina de atención a la interrupción se **preserva el estado** del microprocesador (registros, banderas) usualmente en el stack. Esto es **responsabilidad del programador**. Como se desconoce qué uso estaba haciendo de los registros el programa principal cuando fue interrumpido, los mismos deben ser preservados al comienzo de la rutina de atención para poder restaurarlos antes de retornar y así no perturbar el funcionamiento del programa principal.
4. Eventualmente la rutina de servicio identifica **cuál fue el dispositivo** que solicitó atención para ejecutar el trozo de programa que lo atiende.
5. Se **atiende** al dispositivo. Cuando no fue borrado en el paso (2), el flip-flop de petición es borrado en esta etapa mediante un pulso generado mediante alguna instrucción de lectura o escritura de E/S.
6. Se **restaura el estado** del microprocesador. Esto también es **responsabilidad del programador**. En algunos casos es necesario también volver a habilitar las interrupciones como vamos a ver más adelante.
7. Se **retorna al programa principal** utilizando alguna variante de instrucción RET que recupera la dirección de retorno desde el stack. En el Z80 se utiliza alguna de las instrucciones RET, RETI o RETN según el caso.



La figura representa las etapas de la secuencia en un diagrama de tiempos. De todas las etapas de la secuencia representada, solamente en (1) y en (5) se está realizando procesamiento útil. El resto es tiempo de sobrecarga u *overhead* necesario para poder transferir el control a la subrutina de atención adecuada. Sobre ese diagrama pueden definirse los siguientes tiempos:

- **Tiempo de latencia**: desde que el periférico solicita atención hasta que el microprocesador reconoce la solicitud.
- **Tiempo de respuesta**: desde que el periférico solicita atención hasta que es efectivamente atendido.

El tiempo de latencia va a variar dependiendo del instante en que llegue la solicitud y de qué instrucción estaba siendo ejecutada en ese momento. En principio, el peor caso sería que la solicitud llegue cuando está comenzando a ejecutarse la instrucción más larga del repertorio de instrucciones. Vamos a ver más adelante que en un procesador puede deshabilitarse temporalmente la atención de interrupciones. El peor caso correspondería entonces a la mayor sección de código en que las interrupciones están deshabilitadas. Los dispositivos que interrumpen a un sistema pueden tener diferentes requerimientos en cuanto al tiempo de respuesta. Suponiendo que llega la solicitud de un dispositivo en el instante $t = treqi$, por lo general puede determinarse un instante posterior *tdeadi*, usualmente llamado *deadline* a partir del cual si el procesador todavía no ha atendido el requerimiento comienzan a afectarse notoriamente la performance o la funcionalidad del sistema. Dependiendo de las características del dispositivo que pide atención y del sistema el *deadline* será más o menos exigente y serán más o menos graves las consecuencias de no cumplirlo.
Un ejemplo interesante es el de un reloj implementado mediante una interrupción periódica en la que se incrementa una variable que lleva la cuenta del tiempo transcurrido. Esta variable puede ser consultada por el resto del sistema para conocer la hora actual. En este caso cada solicitud de interrupción debe ser atendida antes de que llegue la solicitud siguiente. De lo contrario volverá a setearse el flip-flop de petición, que en realidad todavía estaba seteado desde la solicitud anterior que aún no fue atendida. Cuando finalmente la interrupción es atendida no hay como distinguir que en realidad hubo dos peticiones de interrupción. La interrupción es atendida solamente una vez (y en consecuencia la variable es incrementada una sola vez), perdiéndose una interrupción y provocando que el reloj de tiempo real "atrase" respecto a la realidad.

Interrupciones enmascarables

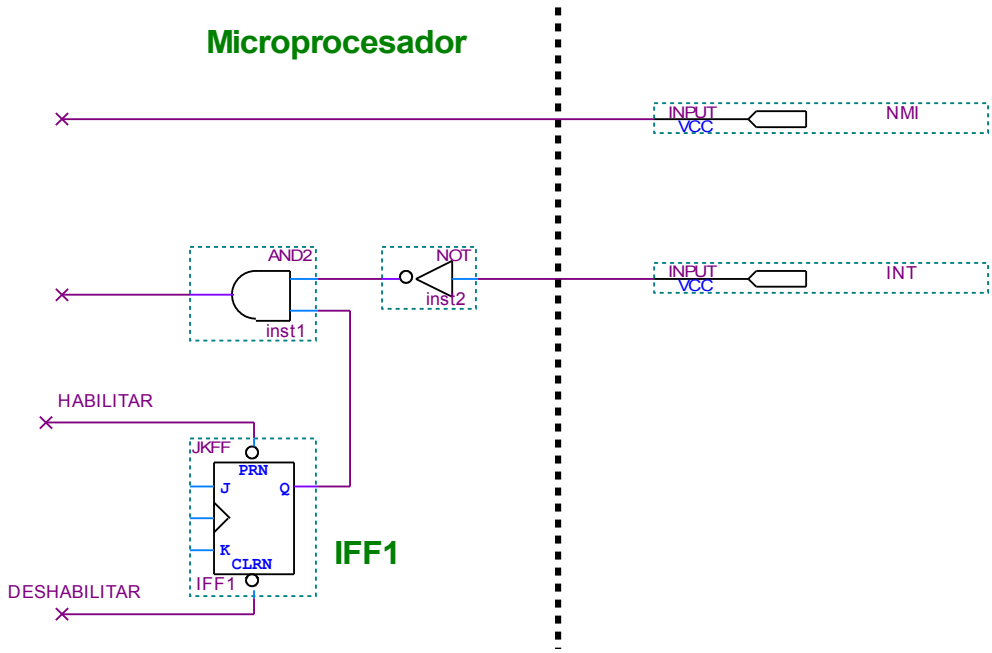
A menudo es necesario inhibir transitoriamente la atención de interrupciones por parte del procesador. Antes de poder atender en forma correcta una interrupción es necesario que varios elementos sean inicializados y configurados:

- El stack debe ser inicializado para poder almacenar correctamente la dirección de retorno.
- El hardware externo involucrado en las interrupciones debe ser inicializado.
- A menudo la propia estructura de interrupciones del microprocesador necesita ser inicializada. P. ej. en el caso del Z80 debe elegirse uno de varios modos de funcionamiento posible y en algunos de esos modos es necesario inicializar el registro I y una tabla con las direcciones de comienzo de las rutinas de atención para los diferentes dispositivos.
- Algunas variables deben ser inicializadas antes de comenzar a aceptar interrupciones.

Otro motivo para deshabilitar interrupciones es querer proteger una determinada sección de programa para evitar que sea interrumpida.

- El motivo más usual para querer proteger una sección de programa es evitar que la rutina de atención a la interrupción acceda a estructuras de datos que están siendo manipuladas por la sección crítica de programa que se está protegiendo, ya que en ese caso se correría el riesgo de que la información leída desde la rutina de atención a la interrupción sea inconsistente.
- Otro caso es que la sección de programa sea crítica en tiempo de ejecución, esto es que interese o bien que demore el menor tiempo posible en ejecutarse o bien que demore siempre lo mismo. En los dos casos la inserción del tiempo de ejecución de una rutina de atención a interrupción afecta negativamente.

La solución para inhibir temporalmente la atención de interrupciones es enmascarar la entrada de solicitud de interrupción con la salida de un flip-flop. Este flip-flop debe estar seteado para habilitar la atención de interrupciones y borrado para impedir que se active la solicitud de interrupción. Luego de un RESET este flip-flop debe quedar borrado inhibiendo las interrupciones. Luego de completar todas las inicializaciones necesarias, el programa puede llevar a uno explícitamente este flip-flop para permitir la atención de interrupciones. A menudo este mecanismo es incluido por el fabricante del microprocesador dentro del chip, agregando además las instrucciones necesarias para habilitar o deshabilitar la atención de interrupciones. En ese caso se dice que la entrada de interrupción es *enmascarable*.



Por ejemplo, el Z80 de Zilog tiene la entrada INT de interrupción enmascarable equipada con un flip-flop interno (denominado IFF1) que puede manipularse con las instrucciones EI (Enable Interrupts) y DI (Disable Interrupts).
El Z80 cuenta además con una segunda entrada de solicitud de interrupción a través del pin NMI que es una entrada de interrupción *no enmascarable*. Esta entrada se reserva en general para atender a eventos catastróficos o de muy alta prioridad para el sistema. Un ejemplo clásico de interrupción de muy alta prioridad es un aviso temprano de falla en la alimentación de energía: en los pocos milisegundos

remanentes hasta que la fuente de alimentación se descargue totalmente se debe almacenar información crítica en alguna memoria no volátil y bajar el sistema en forma ordenada.

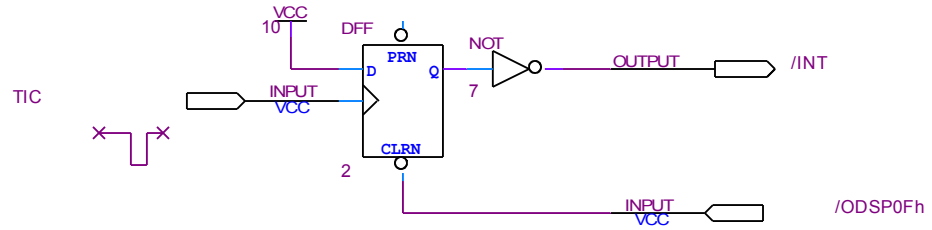
Cómo se evita que una única solicitud provoque múltiples interrupciones anidadas

Habitualmente durante el ciclo de reconocimiento a una interrupción enmascarable, el microprocesador deshabilita la atención de interrupciones hasta que el programador las vuelva a habilitar explícitamente con una instrucción EI o similar. Esto da tiempo a quien escribe la rutina de atención a la interrupción para borrar el flip-flop de petición de interrupción antes de ejecutar la instrucción EI para habilitar la atención de interrupciones. Si no se deshabilitaran automáticamente las interrupciones, al final de la primera instrucción de la rutina de atención de interrupción el procesador volvería a sensar el pin de interrupciones y se desencadenaría un nuevo ciclo de reconocimiento de interrupción. Esto continuaría en una recursión sin fin, impidiendo llegar a la parte de la rutina de atención a la interrupción en que se borra el flip-flop de petición.

Para el caso interrupciones no enmascarables como es la entrada NMI del Z80, la solución al problema debe ser diferente ya que esta entrada es muestreada siempre al final de cada instrucción. En este caso la entrada NMI es sensible a flanco de bajada ("negative edge triggered"), por lo que una vez que se atiende una interrupción NMI el pin debe subir a nivel alto y volver a bajar para que una nueva interrupción sea atendida. Como precaución adicional, el fabricante del Z80 exige que la entrada se mantenga en nivel bajo en el momento en que es sensada en el flanco de subida del último período de reloj de la instrucción.

Ejemplo: interrupción periódica

Supongamos que tenemos una señal periódica **tic** y queremos que con cada flanco de subida de la misma se genere una interrupción al procesador. Una aplicación típica de esto es la actualización de un reloj de tiempo real que se ejemplifica más abajo. Otro ejemplo puede ser la transferencia de datos con un conversor A/D o D/A a determinada frecuencia de muestreo. Supongamos por ahora que esta es la única petición de interrupción que se realiza al procesador.



El hardware necesario es un flip-flop de petición, similar al del ejemplo inicial, salvo que en este caso se optó por borrar la petición mediante un pulso de selección de dispositivo de salida (generado a partir de IORQ, WR y decodificación de direcciones) que puede activarse explícitamente con una instrucción out dentro de la rutina de atención a la interrupción.
La rutina de atención a la interrupción debe preservar los registros que va a modificar, incrementar la variable y volver a dejar todo en su estado inicial antes de retornar.
De esta manera el valor de la variable puede ser consultado desde cualquier parte del programa principal e indica la cantidad de períodos de tic que han transcurrido desde la inicialización. En caso de ser necesario la variable puede ser de más bits, y puede convertirse a otras unidades en el momento de ser desplegada (p. ej. a hs:min:seg).

```
rutint:   ;preservo estado
          push af
          ; incremento variable
          ld a, (VARIABLE)
          inc a
          ld (VARIABLE), a
          ; borro FF de petición
          out (0Fh), a
          ; restauro estado y retorno
          pop af
          ei
          ret
```

Estructura de Interrupciones del Z80

Como ya fue adelantado, el Z80 tiene dos entradas de interrupción: la entrada de interrupción no enmascarable NMI y la entrada de interrupción enmascarable INT. La entrada NMI es sensada por el procesador al final de la ejecución de cada instrucción. La entrada INT es sensada en el mismo instante, pero solamente cuando las interrupciones enmascarables están habilitadas. La entrada NMI tiene prioridad frente a la entrada INT, de manera que si al final de una instrucción ambas entradas están activas se atenderá a la interrupción no enmascarable NMI.

NMI. Interrupción no enmascarable

Cuando se activa la entrada NMI siempre es atendida. Para que el pedido de interrupción sea atendido, debe producirse un flanco de bajada en NMI y el nivel bajo debe mantenerse hasta que la solicitud es detectada por el procesador en el último período T de la instrucción que estaba siendo ejecutada.
La rutina de atención a NMI debe estar previamente cargada en memoria en la dirección 66h.
En respuesta a una solicitud de interrupción en NMI se produce lo siguiente:

- El ciclo de reconocimiento a la interrupción es idéntico a un ciclo M1 (salvo porque dura un T más), pero el OPCODE leído es ignorado. Este ciclo está descrito como un ciclo de máquina más en la hoja de datos.
- Luego se realiza lo equivalente a lo que haría (si existiera) una instrucción RST 66H, es decir se salva la dirección de retorno en el stack y se salta a la dirección fija 66H

Todo este proceso demora al menos el tiempo del ciclo de reconocimiento (5T) más los dos ciclos de escritura en memoria para guardar la dirección de retorno (3T+3T).
Durante el ciclo de reconocimiento se deshabilitan las interrupciones enmascarables (se borra el flip-flop IFF1), de manera que estamos seguros que al entrar a la rutina de atención a interrupción NMI las interrupciones correspondientes a la entrada INT están deshabilitadas.
Al retornar hay que restaurar el estado del microprocesador. Dado que NMI siempre es atendida no sabemos si cuando llegó la solicitud NMI las interrupciones enmascarables estaban habilitadas o no. Para eso el fabricante equipó al Z80 con un segundo flip-flop interno, denominado IFF2. Durante el ciclo de reconocimiento a NMI se borra IFF1 para deshabilitar las interrupciones enmascarables, pero el valor que tenía se preserva en el flip-flop IFF2.
En el ciclo de reconocimiento a NMI, IFF1 e IFF2 se modifican de la siguiente forma:

IFF2 <--- IFF1
IFF1 <--- 0

Para retornar de la rutina de atención a NMI el fabricante agregó la instrucción RETN. Esta instrucción además de hacer lo mismo que la instrucción RET vuelve a copiar el contenido de IFF2 a IFF1, de modo que si retornamos con la instrucción RETN la habilitación de interrupciones enmascarables vuelve a quedar como estaba antes de atenderse la interrupción NMI.
En la ejecución de la instrucción RETN:

IFF1 <--- IFF2

Los flip-flops IFF1 e IFF2 también son modificados por las instrucciones EI y DI, por el ciclo de reconocimiento a INT y en el reset del procesador. Por ejemplo la instrucción DI borra tanto IFF1 como IFF2 para permitir deshabilitar expresamente las interrupciones enmascarables desde dentro de la rutina de atención a NMI.

INT. Interrupción enmascarable

La entrada de interrupción enmascarable INT del Z80 responde diferente a las solicitudes de interrupción dependiendo del modo de interrupciones que esté activo. Existen 3 modos: Modo 0, Modo 1 y Modo 2. Luego de un reset el procesador arranca trabajando en Modo 0. Se puede cambiar de modo con las instrucciones IM 0 o IM 1 o IM 2.
El más sencillo de los modos de funcionamiento es el modo 1, que está pensado para que exista un solo dispositivo solicitando interrupción por lo que hay una única rutina de atención a interrupción en la dirección prefijada 38h.
El modo 0 y el modo 2 en cambio están pensados para atender solicitudes de varios periféricos, diferenciando durante el ciclo de reconocimiento cuál es el periférico que solicitó la interrupción y debe ser atendido. En el modo 0 el funcionamiento es idéntico al del procesador 8080 de Intel con el cual el Z80 es compatible. Por ese motivo el modo 0 es el modo que queda seleccionado luego de un RESET.

Ciclo de reconocimiento

En cualquiera de los 3 modos se ejecuta un ciclo de reconocimiento de interrupción en el que se activan simultáneamente M1 y IORQ. Este ciclo dura normalmente 5T y está descrito en la hoja de datos como un ciclo M.
En los casos de modo 0 y modo 2, es durante este ciclo que el dispositivo que solicita interrupción debe poner sobre el bus de datos la información que permita al Z80 identificarlo para así transferir el control a la rutina de atención correspondiente.
Veamos que sucede a partir de ahí en cada uno de los modos, comenzando por el modo 1 que es el más sencillo.

Modo 1

Luego del ciclo de reconocimiento se produce lo mismo que sucedería en una instrucción RST 38h, esto es se salva la dirección de retorno en el stack y se salta a la dirección fija 38h donde debe estar cargada la rutina de atención a la interrupción.
Todo este proceso demora al menos el tiempo del ciclo de reconocimiento (5T) más los dos ciclos de escritura en memoria para guardar la dirección de retorno (3T+3T).

El modo 1 es el más adecuado para manejar un único periférico ya que el hardware externo necesario es mínimo (se limita al flip-flop de petición), pero si se desea manejar varios periféricos que interrumpen es responsabilidad de la rutina de atención a interrupción identificar cuál es el periférico que está pidiendo servicio y hacer las tareas que correspondan.

Modo 0

Permite manejar interrupciones de varios periféricos a través de la misma señal INT. Este modo es compatible con el funcionamiento de las interrupciones en el 8080, un microprocesador anterior de INTEL. Luego de un RESET el Z80 nace en este modo para permitir el funcionamiento correcto de programas escritos para el 8080.
En este modo durante el ciclo de reconocimiento de interrupción en que se activan IORQ y M1 el Z80 lee el bus de datos, interpreta lo que lee como el código de operación de una instrucción y luego ejecuta esa instrucción.
Lo habitual es que el periférico coloque sobre el bus el código de operación de una de las varias instrucciones RST para provocar la transferencia de control del programa a direcciones diferentes para los diferentes periféricos. Si el byte leído es el código de operación de una instrucción de varios bytes (p. ej. una instrucción CALL) entonces el Z80 genera nuevos ciclos INTA y el periférico debe poner sobre el bus los restantes bytes de la instrucción. Si nos limitamos a instrucciones de un solo byte entonces de forma relativamente sencilla se puede distinguir entre 7 periféricos diferentes, utilizando 7 de las 8 instrucciones RST ya que la instrucción RST 0 hace ejecutar el código de inicialización.

Modo 2

Es el modo de funcionamiento del Z80 para manejar solicitudes de varios periféricos. En este modo durante el ciclo de reconocimiento a la interrupción se lee un byte - denominado vector de interrupciones - que es utilizado como índice dentro de una tabla. Esta tabla almacena las direcciones de comienzo de las rutinas de atención de los diferentes dispositivos.
Una vez obtenido el vector y almacenado el contenido del PC en el stack, el Z80 forma un puntero a memoria con el contenido del registro I como byte alto y el vector leído del bus como byte bajo. Desde el lugar apuntado por ese puntero lee la dirección de comienzo de la rutina de atención y le transfiere el control.
La tabla ocupa una página de 256 lugares de memoria. Cada entrada en la tabla ocupa dos bytes ya que lo que se almacena son direcciones de 16 bits. Lo usual es que las entradas de la tabla estén alineadas a direcciones pares, o lo que es lo mismo se eligen los vectores con el bit menos significativo en 0. De esta forma se pueden distinguir 128 dispositivos diferentes y los 7 bits más significativos del vector indican la posición del dispositivo en la tabla. En algunos casos es a ese nro. de 7 bits a lo que se le llama vector.
Antes de poder comenzar a trabajar con interrupciones en modo 2 la tabla debe estar correctamente inicializada. Esto incluye el registro I para determinar la dirección de comienzo de la tabla y los lugares de la tabla asignados a los dispositivos que estén instalados.

En resumen, el Z80 lee el vector durante el ciclo de reconocimiento, almacena la dirección de retorno en el stack, lee desde la tabla la dirección de comienzo de la rutina de atención y finalmente le transfiere el control. El procedimiento completo demora 17T: un ciclo INTA para la lectura del vector (5T), dos ciclos de escritura en memoria para almacenar el PC en el stack (3T+3T) y dos ciclos de lectura de memoria para leer la dirección de comienzo desde la tabla (3T+3T).

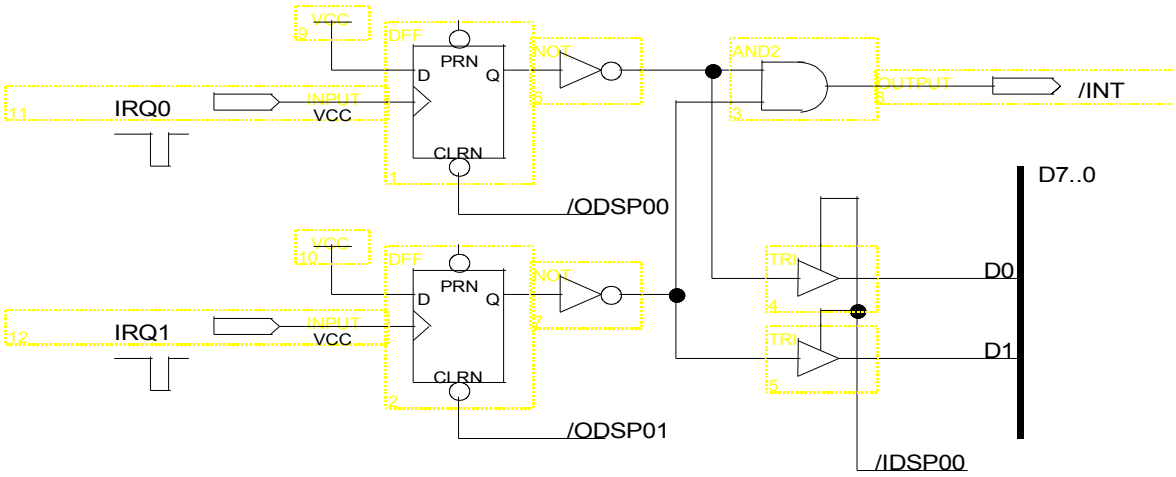
Ejemplos

Ejemplo modo 1.

- O bien: Examen DIC 91. Display 4 dígitos
- O bien: en modo 1, interrupción periódica borrada con INTA. Aplicaciones (Real Time Clock, conversor AD o DA). Rutina de atención para conversor DA que toma datos de cola circular. Comunicación con prog. Ppal. (datos en cola circular) y estado (índice en la cola).

Ejemplo: dos dispositivos en Modo 1

La identificación de cuál es el dispositivo que solicitó interrupción debe realizarse por software dentro de



la rutina de atención a la interrupción. Para manejar correctamente las dos solicitudes se necesita un flip-flop de petición separado para cada dispositivo, que sea borrado en forma independiente cuando se atienda al dispositivo correspondiente. Ambos flip-flops se combinan con una compuerta para activar la entrada INT si cualquiera de las dos solicitudes está pendiente.

```
pendientes      EQU   0
borro0          EQU   0
borro1          EQU   1

        org 38H
rutint: push af
        in a, (pendientes)
        bit 0, a
        jr z, atiendo0
atiendo1:
        ei
        out (borro1), a
        call atiendo1
        pop af
        ret
atiendo0:
        out (borro0), a
        call atiendo0
        pop af
        ei
        ret
```

Dentro de la rutina de atención debo identificar cuál de los dispositivos es el que ha interrumpido. Para eso necesito un puerto de entrada que me permita leer el estado de los FF de petición.

Al atender a un dispositivo debo borrar su FF de petición. No puedo hacerlo con /IORQ+/M1 porque no podría identificar dentro de la rutina de atención al dispositivo que interrumpió (siempre leería el FF borrado).

En el recuadro se presenta una posible solución para el esqueleto de la rutina de atención a la interrupción, en donde se identifica al dispositivo solicitante, se borra el flip-flop correspondiente y se invoca a una subrutina para que lo atienda.
Si en el momento de atender la interrupción están ambos flip-flops de petición activos, se puede priorizar un dispositivo frente a otro eligiendo el orden en que se testean los flip-flops de petición. En la solución del recuadro se atiende primero al dispositivo 0.

Otra situación en la que se puede priorizar a un dispositivo frente a otro es cuando llega una solicitud de interrupción de un dispositivo cuando se está atendiendo al otro. En el ejemplo, en el trozo de programa en que se atiende al dispositivo 1 se pone la habilitación de interrupciones al comienzo (en la etiqueta *atiendo1*) y entonces se permite que una solicitud del dispositivo 0 sea atendida mientras se estaba atendiendo al dispositivo 1. En ese caso quedan dos instancias de la rutina de atención a interrupción

anidadas. En el trozo de programa en que se atiende al dispositivo 0 en cambio se eligió poner la instrucción **ei** al final, impidiendo de esa manera que el dispositivo 1 interrumpa cuando se está atendiendo al dispositivo 0. Claramente este mecanismo de manejo de prioridades se hace más complejo a medida que se aumenta la cantidad de dispositivos, haciéndose rápidamente inmanejable.

Ejemplo: inicialización modo 2 (Pr. 7 Ej. 6 – Ex. Set. 94 Ej. 4)

*Un sistema basado en el Z80 tiene 3 dispositivos de E/S que utilizan interrupciones en **modo 2**. Los dispositivos tienen configurado en su hardware los vectores de interrupción 0, 2 y 5 respectivamente. Las rutinas de atención a las interrupciones comienzan respectivamente en las direcciones **rutint_0**, **rutint_2** y **rutint_5** de ROM. El sistema tiene 32K de ROM y 32K de RAM. Escribir un trozo de código que se ejecute enseguida de un RESET del procesador y que haga todas las inicializaciones necesarias para que funcione el sistema de interrupciones del Z80.*

La solución difiere según donde ubiquemos la tabla de interrupciones. Si la tabla se ubica en RAM entonces debe cargarse explícitamente los lugares correspondientes a los dispositivos 0, 2 y 5 con las correspondientes direcciones de comienzo de rutina de atención. Si en cambio la tabla se ubica en ROM hay que escribir las directivas al ensamblador para generar un archivo con el contenido de la tabla y grabarlo en la ROM.

Solución con la tabla en RAM:

```
        ORG 0

        ld sp, 0                ; inicializo sp al fin de RAM + 1

        ld a, 80h
        ld i, a                 ; tabla de interrupciones al comienzo de la RAM
        ld bc, rutint_0
        ld (8000h), bc          ; cargo rutint_0 en el lugar 0 de la tabla
        ld bc, rutint_2
        ld (8000h+2*2), bc      ;cargo rutint_2 en el lugar 2 de la tabla
        ld bc, rutint_5
        ld (8000h+2*5), bc      ; cargo rutint_5 en el lugar 5 de la tabla
        im 2                    ; modo 2 de interrupciones
        ;inicializacion de los dispositivos
        ...
        ei                      ; habilito interrupciones
        ...
```

Solución con la tabla en ROM:

```
        ORG 0

        ld sp, 0                ; inicializo sp al fin de RAM + 1

        ld a, 20h
        ld i, a                 ; tabla de interrupciones grabada previamente en ROM
        im 2                    ; modo 2 de interrupciones
        ;inicializacion de los dispositivos
        ...
        ei                      ; habilito interrupciones
        ...

        ORG 2000H               ; en ROM
        DW      rutint_0        ; directivas para generar el contenido a grabar en la ROM
        DW
        DW      rutint_2
        DW
        DW
        DW      rutint_5
```

PRIORIDADES

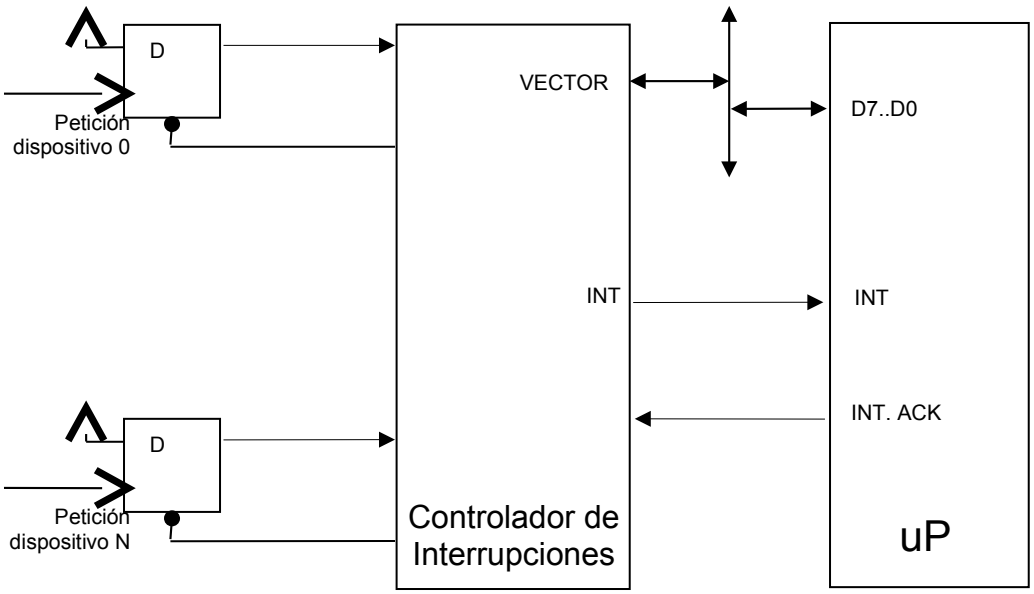
Arbitración de prioridades en solicitudes de interrupción

Como quedó esbozado con el ejemplo de dos dispositivos compartiendo la entrada INT en modo 1, al manejar varios periféricos que solicitan interrupciones al microprocesador surgen dos situaciones en las que hay que darle preferencia a algunos periféricos sobre otros.

- En el momento en que se realiza el ciclo de reconocimiento a la interrupción (en el caso del Z80 en modo 2 se activan M1 e IORQ simultáneamente) el microprocesador espera leer el vector que identifica al periférico que debe ser atendido. **Si en ese momento hay más de una petición de interrupción pendiente de ser atendida** se debe determinar cuál de los periféricos que esperan ser atendidos es el que tiene derecho a identificarse poniendo su vector sobre el bus.
- En general se desea que un periférico más importante o con fuertes requerimientos en tiempo de respuesta **pueda interrumpir a la rutina de atención de otro periférico menos importante** pero no a la inversa.

Si se trata solamente de dos dispositivos, la segunda situación puede resolverse manejando en forma adecuada la **habilitación de interrupciones** (habilitando interrupciones dentro de la rutina de atención del periférico de menos prioridad) o utilizando la entrada NMI para el dispositivo de mayor prioridad.
La primer situación señalada puede manejarse averiguando al comienzo de la rutina de atención, cuales dispositivos están esperando atención y manejar por software la prioridad entre ellos. Esto agrega *overhead* y empeora el tiempo de respuesta; y rápidamente se hace inmanejable.
Salvo que se estén manejando 2 o 3 dispositivos, la solución usual es utilizar un dispositivo externo al microprocesador que, por mecanismos hardware, determina en cada momento cuál es el periférico que debe ser atendido.

Controlador de interrupciones. Circuito externo de arbitración de prioridades.



Un controlador de interrupciones es un dispositivo que por un lado recibe las solicitudes de interrupción de los diferentes dispositivos periféricos y por otro lado maneja hacia el microprocesador la entrada de interrupciones y el vector de identificación de los dispositivos durante los ciclos de reconocimiento de interrupciones. Para eso realiza las siguientes acciones:

- Cuando llega un ciclo de reconocimiento de interrupción se encarga de poner sobre el bus de datos el vector correspondiente al dispositivo de más prioridad de los que están pendientes de ser atendidos.
- Decide si pasa al microprocesador una solicitud de interrupción (activando INT) dependiendo de si el dispositivo que solicita la interrupción tiene más o menos prioridad que el dispositivo que está siendo atendido.

El dispositivo controlador de interrupciones más difundido es el 8259 producido por Intel para los procesadores x86 y utilizado en los computadores personales.

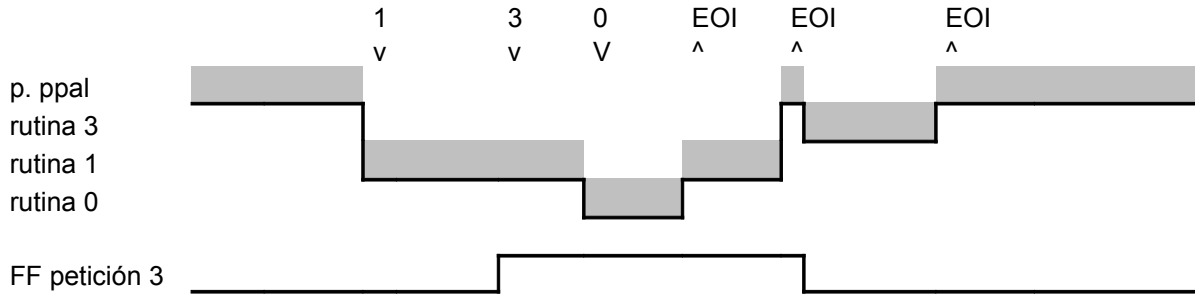
Para el manejo de prioridades pueden utilizarse diferentes estrategias, a continuación se presentan algunas de las más usuales.

Jerarquía fija de prioridades ("*Fully Nested*")

Se establece entre los periféricos una jerarquía fija de prioridades de manera que un periférico puede interrumpir a otro de menor prioridad pero no a la inversa. La asignación de las prioridades correspondientes a cada dispositivo es un problema de ingeniería complejo porque se deben mapear varias características (importancia, requerimientos de tiempo de respuesta, gravedad de no respetar el tiempo de respuesta, periodicidad, duración de la rutina de atención) de los diferentes dispositivos en una única escala jerárquica.

El objetivo de esta estructura de prioridades es garantizar que en cada instante se esté atendiendo al periférico de mayor prioridad de los que hayan solicitado atención. El árbitro externo recibe las solicitudes de interrupción de los periféricos y debe decidir si la pasa o no al microprocesador, dependiendo de las prioridades del solicitante y de la prioridad de lo que se esté ejecutando en ese momento en el microprocesador. Además, para que el microprocesador efectivamente atienda una interrupción de un dispositivo de mayor prioridad, todas las rutinas de atención a interrupciones deben habilitar las interrupciones al comienzo.

Ejemplo:



Supongamos que tenemos 4 periféricos (p0, p1, p2 y p3) a los que asigno prioridades decrecientes (p0 es el de más prioridad) y llegan solicitudes de interrupción en el orden indicado en el diagrama.
Mientras se está ejecutando el programa principal llega una solicitud del periférico p1 que es atendida, pasando a ejecutarse la rutina de atención a p1. Si mientras se atiende a p1 llega una solicitud de p3 deseamos que la rutina de atención a p1 no sea interrumpida, dado que p3 tiene menor prioridad. Para eso el controlador de interrupciones no debe activar INT hasta que no termine de atenderse p1. En cambio si en la misma situación llega una solicitud de p0, el controlador de interrupciones en este caso sí debe activar INT de manera que se interrumpa la rutina que atiende p1 para pasar a atender a p0. Cuando llegue el ciclo de reconocimiento a la interrupción en respuesta a la activación de INT, el controlador de interrupciones debe poner sobre el bus el vector de interrupciones que identifica a p0, que es el dispositivo de mayor prioridad de los dos que tienen una solicitud pendiente (p0 y p3).
Para que el microprocesador efectivamente atienda la interrupción solicitada las interrupciones deben habilitarse al comienzo de la rutina de atención a p1 con una instrucción EI.
Como se muestra en el diagrama la solicitud del periférico de menor prioridad p3 recién es atendida una vez que termina la ejecución de las rutinas de atención a p0 y p1 y se vuelve a ejecutar el programa principal. Que esto suceda así es responsabilidad del controlador de interrupciones, que debe activar la entrada INT del procesador recién cuando se retorna al programa principal desde la rutina de atención a p1.

Para tomar las decisiones que se describen líneas arriba, el controlador de interrupciones debe saber cuál rutina de interrupción se está ejecutando en cada momento. Solo así va a ser capaz cuando recibe una nueva solicitud de saber si tiene que pasarla al microprocesador, o por el contrario esperar porque se está atendiendo a un dispositivo de mayor prioridad. Para eso el controlador de interrupciones debe llevar la cuenta para cada periférico de si su rutina de atención está todavía "en servicio". Para eso no es suficiente con saber en que instante comienza la atención a cada periférico, sino que además se necesita algún mecanismo para que el controlador se entere de la finalización de la ejecución de la rutina de atención, para así borrar la bandera "en servicio" correspondiente a ese periférico. En algunos controladores de interrupciones ese mecanismo es explícito, la rutina de atención a interrupción debe enviar un comando

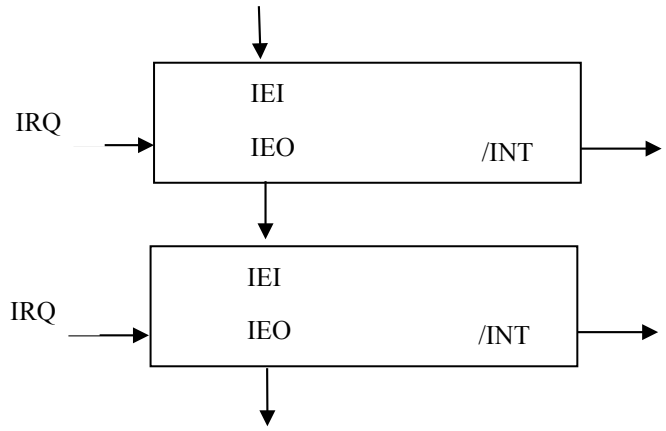
de “fin de interrupción” (*End Of Interrupt* en inglés, o abreviado *EOI*) escribiendo en algún puerto del controlador.
Los fabricantes del Z80 idearon otro mecanismo: la instrucción RETI tiene un funcionamiento idéntico al de la instrucción RET. Sin embargo, RETI tiene un código de operación diferente. Si se utiliza la instrucción RETI solamente para retornar de rutinas de atención a interrupción, entonces un controlador de interrupciones externo (o el dispositivo que cumpla esa función) puede detectar el “fin de interrupción” observando el bus. Si detecta un ciclo M1 con el código de operación de la instrucción RETI significa que ha finalizado una rutina de atención a interrupción (estrictamente son dos ciclos M1, RETI se codifica con los dos bytes de opcode ED 4D, a diferencia de RET cuyo opcode es C9).

Rotación automática

El esquema de prioridades “*Fully Nested*” presentado arriba no siempre es el más adecuado. Cuando hay varios periféricos con la misma importancia una alternativa es modificar la jerarquía después de atender a una solicitud, pasando al periférico que acaba de ser atendido al último lugar en el orden de prioridades. Si hay dos periféricos compitiendo por ser atendidos se atenderá primero al que hace más tiempo que fue atendido por última vez.
De esta manera se garantiza que el tiempo de respuesta a un periférico no es mayor que la suma de los tiempos de atención del resto de los periféricos.
Nótese que en el caso del esquema de prioridades “Fully Nested” no es posible establecer una cota superior para el tiempo de respuesta a un dispositivo de baja prioridad: si continuamente llegan solicitudes para dispositivos de mayor prioridad, el dispositivo de baja prioridad no es atendido.

Daisy Chain. Un mecanismo distribuido de arbitración de prioridades.

En lugar de utilizar un bloque controlador de interrupciones centralizado, Zilog ideó un esquema distribuido para realizar la función de asignación de prioridades en una jerarquía fija al cual denominó Daisy Chain.



Cada bloque tiene una entrada IEI (Interrupt Enable Input) que le informa si está habilitado para solicitar interrupciones al microprocesador. La entrada IEI de un bloque está manejada por la salida IEO (Interrupt Enable Output) del bloque anterior, de manera que todos los bloques quedan conectados en una cadena, ordenados por prioridad decreciente. La entrada IEI del dispositivo de mayor prioridad se conecta a “1” de forma que siempre está habilitado para solicitar interrupción.

Las señales utilizadas en un sistema *daisy chain* son las siguientes:

- IEI (Interrupt Enable Input): Esta entrada cuando está inactiva indica que hay algún periférico de mayor prioridad que aún no ha sido terminado de atender. Es activa en nivel alto.

- IEO (Intrerrupt Enable Output): Salida para ser conectada al IEI del siguiente periférico en la cadena. Debe desactivarse cuando este dispositivo tiene una solicitud que no ha sido completada o si la entrada IEI está inactiva.

- INT: Se activa para solicitar interrupción al microprocesador. La entrada INT del microprocesador debe activarse si cualquiera de las señales INT se activan, por lo cual debe manejarse con el AND de las salidas INT de todos los bloques. En general los dispositivos para

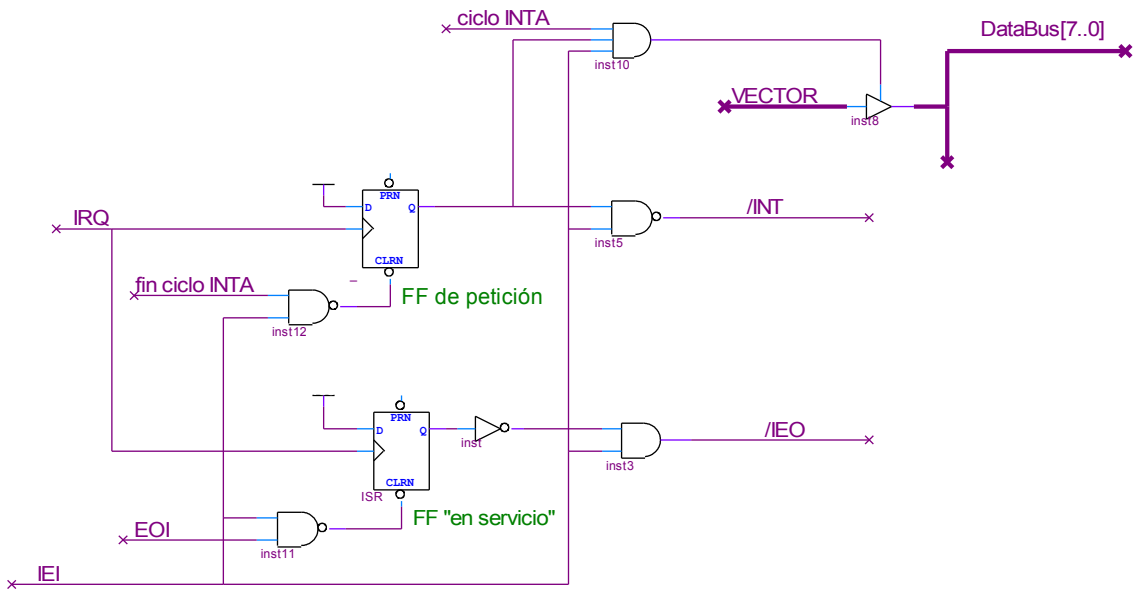
conectarse en Daisy Chain manejan esta salida en "colector abierto", de manera que pueden conectarse juntas con una resistencia de "*pull-up*" en una conexión "*and cableado*".

- Además el bloque debe ser capaz de detectar un ciclo de reconocimiento de interrupción (/M1 e /IORQ activas) para poner, cuando le corresponda, su vector de identificación sobre el bus de datos, y debe además saber reconocer en el bus la ejecución de una instrucción RETI como forma de detectar el fin de la ejecución de las rutinas de atención a interrupción (End of Interrupt o EOI).

Cuando le llega una solicitud de interrupción, el bloque debe memorizar la petición y desactivar IEO para avisar a los dispositivos de menor prioridad que hay solicitudes (al menos la de este bloque) aún no completadas. Si IEI está activo significa que no hay solicitudes de mayor prioridad sin completar, por lo que en ese caso puede activar INT inmediatamente.

Cuando un bloque que tiene una solicitud propia pendiente de ser atendida detecta un ciclo de reconocimiento de interrupción (/IORQ + /M1 = 0) y además la entrada IEI está activa significa que su periférico es el de mayor prioridad con solicitud pendiente. En ese caso el ciclo de reconocimiento está dirigido a él y por tanto debe poner el vector que lo identifica sobre el bus de datos y desmarcar la petición como pendiente. Sin embargo el bloque debe seguir manteniendo su salida IEO inactiva para indicar a los dispositivos de menor prioridad que todavía no están autorizados a solicitar interrupciones. Para eso por lo general se utiliza un FF adicional que indica que la solicitud todavía está siendo atendida. En algunos controladores de interrupciones a esta bandera se le denomina In Service Register (ISR).

Finalmente, cuando la solicitud de un bloque fue atendida pero aún no completada (la bandera "en servicio" encendida) sabe que el procesador no puede estar ejecutando una rutina de atención de menor prioridad. Si en esa situación detecta un fin de interrupción (ejecución de una instrucción RETI) y la entrada IEI está activa entonces sabe también que no hay ninguna solicitud de mayor prioridad sin completar y por lo tanto la instrucción RETI detectada corresponde al fin su propia rutina de atención. En ese caso debe borrar el flip-flop que indica solicitud "en servicio" y activar la salida IEO para habilitar a los dispositivos de menor prioridad a interrumpir.



El circuito de la figura es una implementación preliminar del protocolo *Daisy Chain*. Sin embargo hay algunos detalles de implementación que veremos más adelante que no son tenidos en cuenta por este circuito.

El comportamiento del controlador puede describirse también como una máquina de estados con tres estados básicos: *inactivo*, *pendiente* y *en_servicio*. El bloque controlador está inicialmente en estado *inactivo* y pasa al estado *pendiente* cuando recibe una solicitud por su entrada IRQ. La solicitud queda pendiente hasta que llegue un ciclo de reconocimiento a interrupción con la entrada IEI activa, momento en que pasa al estado *en_servicio*. Luego, cuando se detecta la ejecución de una instrucción RETI y la entrada IEI se encuentra activa puede concluirse que la solicitud fue completada y retornar al estado

inactivo.
La salida INT debe activarse mientras se esté en estado pendiente y la entrada IEI esté activa.
La salida IEO puede activarse solamente si IEI está activa y el bloque está en estado inactivo.

Ejercicio recomendado: Pr. 8 Ej. 1 (Ex. Feb. 92 Ej. 4)

Hilando más fino: Propagación IEI-IEO en Daisy Chain.

En principio el modo 2 de interrupciones de Z80 permite diferenciar entre 128 dispositivos diferentes que solicitan interrupción a través de la entrada /INT. Sin embargo, si el arbitraje de prioridad se realiza con *daisy chain*, el retardo de propagación en la cadena de señales IEI e IEO introduce limitaciones adicionales.
La especificación de *daisy chain* requiere que si una solicitud para un bloque llega durante un ciclo de reconocimiento de interrupción, su consideración se postergue hasta el final del ciclo. Es decir que si un bloque recibe una solicitud de interrupción inmediatamente después de que bajan /IORQ y /M1, el bloque no debe activar INT ni desactivar IEO hasta que el ciclo de reconocimiento haya terminado.
El motivo por el cual se introduce este requerimiento es evitar situaciones en las que más de un dispositivo cree tener derecho a poner su vector sobre el bus.
Consideremos la siguiente secuencia:

1. un dispositivo de baja prioridad solicita interrupción
2. su bloque daisy chain activa INT
3. un dispositivo de mayor prioridad solicita interrupción y desactiva su IEO.
4. inmediatamente comienza el ciclo de reconocimiento de interrupción.

La duración del ciclo de reconocimiento de interrupción debe ser suficiente para que la información se propague por la cadena IEO-IEI hasta el dispositivo de menor prioridad, de manera que este retire su vector del bus antes del momento en que el microprocesador lea el bus de datos. De lo contrario el dispositivo de menor prioridad sigue teniendo su entrada IEI activa y ambos dispositivos creen que el ciclo de reconocimiento les corresponde. En esa situación ambos ponen su vector sobre el bus provocando por un lado que el valor leído por el microprocesador sea basura y por otro lado que se corra el riesgo de quemar las salidas de los dispositivos en conflicto.

Como el retardo de propagación depende de la cantidad de dispositivos conectados en la cadena debemos acotar la cantidad de dispositivos. Si la duración normal del ciclo de reconocimiento a la interrupción no es suficiente para que las señales se propaguen por toda la cadena y el dispositivo de menor prioridad se retire a tiempo, se puede prolongar la duración del ciclo de INTA solicitando tiempos de espera con la entrada /WAIT igual que para los otros ciclos M.

Periféricos programables Zilog

A menudo los fabricantes de microprocesadores además del microprocesador producen diferentes dispositivos periféricos para simplificar el diseño de un sistema completo.
Por lo general estos dispositivos son “programables”, es decir configurables para implementar diferentes modos de funcionamiento. Algunos de los dispositivos periféricos que fabricó Zilog para sus procesadores Z80 son representativos de las funcionalidades más comunes incluidas en este tipo de dispositivos:

- PIO (Paralel Input Output). Dos puertos de 8 bits con su lógica de *handshake*, que pueden configurarse en varios modos (entrada, salida, bidireccional)
- SIO (Serial Input Output). Implementa un puerto serial.
- CTC (Counter Timer Clock). Cuatro contadores de 8 bits programables que pueden utilizarse como temporizadores o como contadores de eventos externos.
- DMA Controller. Controlador de acceso directo a memoria.

En todos los casos la interfaz hacia el microprocesador incluye:

- Puerto de escritura para configurar o enviar **comandos** al periférico
- Puerto de lectura de **status**
- Generación de interrupciones
- Puertos de datos para la función específica del periférico

En el caso de los periféricos de Zilog, la lógica para generación de interrupciones está diseñada para trabajar en modo 2 de interrupciones e incluye el manejo de prioridades con el protocolo *daisy chain*.