Cika Electrónica

Tutorial: CTU-002
Título: **Rabbit 2000**
Autor: Sergio R. Caprile, Senior Engineer

Revisiones	Fecha	Comentarios
0	25/8/03	Edición inicial
1	10/08/04	Corrección Timer A e Interrupciones

El presente es un tutorial sobre Rabbit 2000, el primer miembro de la familia Rabbit, basado en la arquitectura del Z-80/180. Cubrimos las principales características, diferencias y mejoras con respecto al conocido Z-80. Se recomienda la lectura de información al respecto para un estudio más profundo. Si bien se trata de un documento sobre el microprocesador en sí, también hacemos aclaraciones sobre Dynamic C, la herramienta oficial de desarrollo sobre Rabbit.

Índice de contenido

Introducción 2
- Características principales 2
- Características del diseño del procesador Rabbit 3

Mejoras 3
- CPU 3
  - Registros 4
  - Instrucciones 4
    - Direccionamiento relativo16 bits 4
    - Operaciones lógicas y aritméticas 16 bits 4
    - Instrucciones de I/O 5
  - Estructura de Interrupciones 5
- Manejo de Memoria 5
  - Definiciones 5
  - Unidad de Manejo de Memoria (MMU) 5
  - Unidad de Interfaz a Memoria (MIU) 7
  - Cómo maneja la memoria el compilador de Dynamic C 7
- Control de bancos de I/O 8
- Periféricos en chip 8
  - Ports Serie 9
  - System Clock 9
  - Reloj de Tiempo Real 10
  - Ports I/O paralelo 10
  - Slave Port 10
  - Timers 10
- Bootstrap 11

Diferencias 11
- Registros 12
- Instrucciones 12
  - Instrucciones de I/O 13
  - Instrucciones privilegiadas 13
    - Manejo del stack 14
    - Manipulación del registro IP: secciones críticas 14
    - Acceso al registro XPC: saltos largos calculados 14
    - Uso de semáforos 15
- Interrupciones 15

Introducción

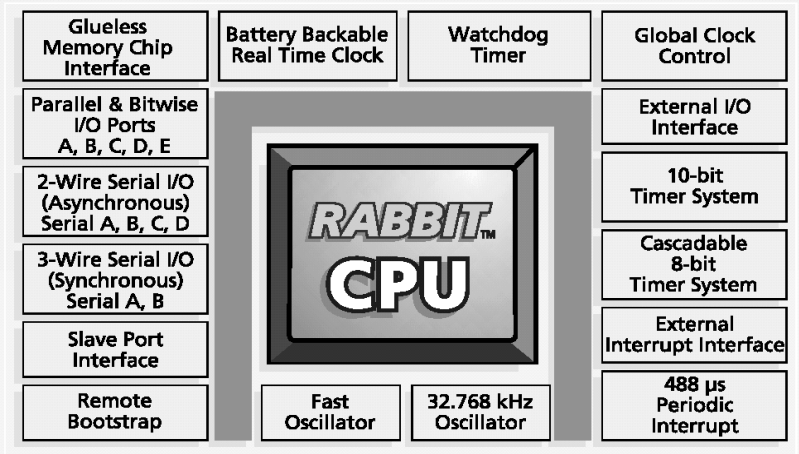
Rabbit Semiconductor se forma expresamente para diseñar un microprocesador orientado a control de pequeña y mediana escala; su primer producto es el microprocesador Rabbit 2000. Los diseñadores de este micro son desarrolladores con muchos años de experiencia en Z80, Z180 y HD64180, por ello el Rabbit posee una arquitectura similar, tiene un alto grado de compatibilidad con estos procesadores, y los supera en muchas otras áreas.

El diseño de Rabbit fue realizado con cooperación de Z-World, experimentado fabricante de placas de control de bajo costo, basadas en Z80/180, y soportadas por un innovador sistema de desarrollo y programación en C (Dynamic C).

El Rabbit es fácil de usar; las interfaces de hardware y software son simples, y su potencia de cálculo es impresionante para un microprocesador con un bus de 8 bits. Esto es así porque su set de instrucciones (derivado del de Z-80) es muy compacto y el diseño de la interfaz de memoria permite máxima utilización del ancho de banda.

Los usuarios de Rabbit se encuentran con un simple y poderoso entorno de desarrollo de hardware y software, y no necesitan emuladores en circuito, ya que una simple conexión al port serie de una PC les permite hacer depuración paso a paso en el circuito definitivo.

Características principales



- ✔ Encapsulado de 100 pines, PQFP. Operación a 3 ó 5V, clock de hasta 30MHz. Rango de temperatura comercial e industrial.
- ✔ Direccionamiento de memoria de 1 Megabyte, capacidad de más de 50.000 líneas de código. El set de instrucciones extendido (basado en el Z-80) es C-compatible, con instrucciones cortas y rápidas para las operaciones más comunes en C.
- ✔ Cuatro niveles de prioridad de interrupciones; el tiempo máximo de latencia es de cerca de 1us a un clock de 25MHz.
- ✔ El acceso a I/O se realiza mediante instrucciones de acceso a memoria con un prefijo, con lo cual es más fácil que en otros microprocesadores con set de instrucciones reducido.
- ✔ Las reglas de diseño de hardware son simples. Pueden conectarse hasta 6 chips de memoria estática (RAM, flash EPROM) directamente sin glue-logic. El Rabbit funciona sin ciclos de espera (wait-states) con una memoria de 70ns de tiempo de acceso, a un clock de 24MHz. Los periféricos generalmente se conectan también sin glue-logic, ya que existen pines que pueden configurarse como chip selects, read strobes o write strobes.
- ✔ Puede bootear del port serie o del slave port paralelo. Esto significa que la memoria de programa (flash) puede soldarse sin programar, y ser grabada luego, sin que sea necesario escribir un programa o BIOS. Un Rabbit que funcione como esclavo de otro procesador puede operar con RAM solamente, dependiendo del master para obtener su programa.

- ✔ Posee 40 entradas y salidas paralelo (compartidas con los port serie). Algunas de las salidas pueden ser sincronizadas con los timers, lo que permite generar pulsos de precisión.
- ✔ Posee cuatro ports serie. Los cuatro pueden operar asincrónicamente, dos pueden operar además en modo sincrónico. Las velocidades llegan a 1/32 de la frecuencia de clock para modo asincrónico y 1/6 para modo sincrónico (1/4 si la fuente de clock es interna). En el modo asincrónico, Rabbit soporta, como el Z180, el envío de caracteres especiales de 9 bits para indicar el inicio de un nuevo mensaje.
- ✔ Reloj de tiempo real con pila de respaldo, funcionando con un cristal externo de 32,768KHz. Puede utilizarse además para generar una interrupción periódica cada 488us. El consumo típico de la batería es de 25uA con el circuito estándar. Existe una opción circuital adicional para reducir este valor.
- ✔ Numerosos temporizadores y contadores (6 en total) que pueden ser usados para generar interrupciones, velocidad de ports serie y pulsos.
- ✔ El oscilador principal de reloj utiliza un cristal o resonador externo. Las frecuencias típicas cubren un rango de 1,8 a 29,5MHz. En casos en que la precisión necesaria pueda obtenerse del oscilador de 32,768KHz, es posible utilizar un resonador cerámico de tipo económico con resultados satisfactorios. Para funcionamiento a ultra bajo consumo, el reloj del procesador puede tomarse del oscilador de 32,768KHz, apagando el oscilador principal. Esto permite que el procesador opere a aproximadamente unas 10.000 instrucciones por segundo, como alternativa a interrumpir la ejecución como hacen otros procesadores.
- ✔ La performance en aritmética de punto flotante es excelente, debido a la poderosa capacidad de procesamiento de Rabbit y la biblioteca de funciones diseñada específicamente de Dynamic C. Por ejemplo, a un clock de 25MHz, una operación de raíz cuadrada demora 40us, una suma 14us y una multiplicación 13us. En comparación, un 386EX con bus de 8 bits a igual frecuencia de clock, usando código compilado con Borland C, es unas 10 veces más lento.
- ✔ Posee un Watchdog Timer
- ✔ El port de programación estándar de 10 pines elimina la necesidad de emuladores en circuito. Un conector de 10 pines y una simple conexión al port serie de una PC es todo lo que se necesita para instalar el código y depurar en circuito

Características del diseño del procesador Rabbit

El Rabbit es un diseño evolucionario. Su set de instrucciones y sus registros son los del Z80/180. El set de instrucciones se ve extendido por un gran número de instrucciones nuevas; algunas instrucciones obsoletas o redundantes han sido eliminadas, a fin de hacer lugar para disponer de opcodes de 1 byte para las instrucciones nuevas más importantes. La ventaja de este tipo de evolución es que los usuarios familiarizados con Z80 inmediatamente comprenden al Rabbit. El código existente puede ensamblarse o compilarse para el Rabbit con cambios mínimos.

El cambio tecnológico ha hecho que algunas características de la familia Z80/180 sean ahora obsoletas, y han sido eliminadas. Por ejemplo, el Rabbit no incorpora soporte especial para RAM dinámicas, pero sí para memorias estáticas. Esto es así debido a que el precio de la memoria estática ha descendido al punto de hacerla preferible para sistemas dedicados de mediana escala. Tampoco incorpora soporte para DMA, debido a que la mayoría de los usos tradicionales del acceso directo a memoria no son aplicables a sistemas dedicados, o las mismas tareas pueden resolverse mejor por otros medios, como rutinas rápidas de interrupciones o procesadores esclavos.

La experiencia de Z-World al desarrollar compiladores ha puesto en evidencia las deficiencias del set de instrucciones del Z80 a la hora de ejecutar código en C. El problema principal reside en la falta de instrucciones para manejar palabras de 16 bits y acceder datos en una dirección calculada, particularmente cuando el stack contiene los datos. Rabbit incorpora nuevas instrucciones que resuelven este problema.

Otro inconveniente con muchos procesadores de 8 bits es su baja velocidad de ejecución y la falta de habilidad para procesar rápidamente grandes cantidades de datos numéricos. Una buena aritmética de coma flotante es una característica importante para la productividad en sistemas pequeños; resulta fácil resolver muchos problemas de programación si se dispone de capacidad de coma flotante. El set de instrucciones mejorado del Rabbit permite procesar rápidamente números en formato de coma flotante y por supuesto también enteros.

Mejoras

CPU

El Rabbit ejecuta instrucciones en menos clocks que el Z80 o el Z180. Estos requieren un mínimo de cuatro clocks para opcodes de 1 byte y tres ciclos extra por cada byte adicional en los opcodes de tipo multibyte; Rabbit requiere sólo dos clocks por cada byte de opcode y cada byte de acceso a memoria. Las operaciones de escritura demandan tres clocks, y se requiere un clock adicional si debe computarse una dirección de memoria o se utiliza alguno de los registros índice para direccionar. Sólo unas pocas instrucciones no siguen esta regla, como *MUL*, que es un opcode de 1 byte y requiere doce ciclos de clock.

Comparado con el Z180, el Rabbit no sólo requiere menos clocks sino que en situaciones típicas soporta clocks de mayor frecuencia y sus instrucciones son más poderosas.

Registros

Los registros de la CPU Rabbit son casi idénticos a los de Z80 ó Z180. Debido a la existencia de una unidad de manejo de memoria y una nueva estructura de interrupciones, como veremos más adelante, encontramos algunos registros con funciones nuevas como *XPC*, *IP*, *IIR* y *EIR*. Desarrollaremos este tema en la sección sobre diferencias.

Instrucciones

Las mejoras más importantes por sobre el Z180 se encuentran en las siguientes áreas:

- ➔ Acceso lectura/escritura relativo al stack pointer o los registros *IX*, *IY*, *HL*, particularmente palabras de 16 bits
- ➔ Operaciones lógicas y aritméticas de 16 bits.
- ➔ Comunicación entre sets de registros: en general, es posible cargar datos en los registros alternativos, pero no utilizarlos como fuente.
- ➔ Instrucciones especiales para acceso a memoria física, evitando el uso de bancos de memoria.
- ➔ Instrucciones de I/O.

A continuación daremos una muestra de las nuevas instrucciones en las diferentes áreas. La presente es a título de demostración, para un completo detalle se recomienda al lector consultar el manual del usuario y el set de instrucciones del Rabbit 2000

Direccionamiento relativo16 bits

Las siguientes instrucciones son una muestra de las nuevas capacidades. El registro *HL* es siempre la fuente de los datos, el destino puede ser *HL* o *HL'*:

```
LD HL,(SP+d)      ; d es un offset de 0 a 255.
                  ; carga 16-bits en HL or HL'
LD (SP+d),HL      ; operación complementaria
LD HL,(HL+d)      ; d es un offset entre -128 y +127,
                  ; usa el valor original de HL para direccionar
                  ; L=(HL+d), H=(HL+d+1)
LD HL',(HL+d)
LD (HL+d),HL
LD (IX+d),HL      ; d: -128 a +127
LD HL,(IX+d)
LD (IY+d),HL
```

Operaciones lógicas y aritméticas 16 bits

Las siguientes instrucciones son una muestra de las nuevas capacidades:

```
;Shifts
RR HL       ; rota HL a la derecha con carry, 1 byte, 2 clocks
            ; usar ADC HL,HL para rotar a la izquierda, o ADD HL,HL si
            ; no se necesita el carry.
RR DE       ; 1 byte, 2 clocks
RL DE       ; rota DE a la izquierda con carry, 1-byte, 2 clocks
;Operaciones lógicas
```

```
AND HL,DE   ; 1 byte, 2 clocks
AND IX,DE   ; 2 bytes, 4 clocks
OR HL,DE    ; 1 byte, 2 clocks
OR IY,DE    ; 2 bytes, 4 clocks
```

Instrucciones de I/O

Rabbit utiliza un interesante esquema para acceder dispositivos de entrada/salida. Cualquier instrucción que acceda a memoria, puede utilizarse para acceder a dos espacios de I/O: uno interno (periféricos en chip) y otro externo (periféricos externos). Esto se realiza anteponiendo a la instrucción un prefijo.

Mediante esta posibilidad, todas las instrucciones de acceso a memoria en 16 bits están disponibles para leer y escribir posiciones de I/O. La unidad de mapeo de memoria no se utiliza para I/O, por lo que se dispone de un direccionamiento lineal de 16 bits.

Desarrollaremos este tema en la sección diferencias.

Estructura de Interrupciones

El manejo de interrupciones en Rabbit incorpora tres niveles de prioridad de interrupción (uno más que en Z80/180) y cuatro niveles de prioridad a los que opera el microprocesador. Al producirse una interrupción, si la prioridad de ésta es mayor que la prioridad en que está operando el procesador, entonces será atendida al final de la ejecución de la instrucción en curso (excepto instrucciones privilegiadas).

En sistemas donde todas las interrupciones funcionan al mismo nivel de prioridad, se genera latencia cuando las rutinas deben esperarse entre sí. La intención de Rabbit es que la mayoría de los dispositivos utilicen el nivel de prioridad 1. Como el código que corre a prioridad 0 (normal) ó 1 (interrupciones) permite interrupciones a nivel 2 y 3, estos niveles tienen garantía de ser atendidos en 20 clocks, lo que demora la secuencia más larga de instrucciones privilegiadas, seguida de una no-privilegiada.

La intención de Rabbit es que la mayoría de los dispositivos utilicen interrupciones de prioridad 1. Aquellos que necesiten una respuesta extremadamente rápida utilizarán prioridad 2 ó 3. Es importante que el usuario tenga cuidado y no inhabilite en demasía las interrupciones en secciones críticas. La prioridad del procesador no debe ser elevada más allá del nivel 1 excepto en situaciones consideradas cuidadosamente. El efecto de este esquema, resulta en que una rutina de interrupciones sólo puede ser interrumpida por otra de mayor prioridad (a menos que el programador explícitamente disminuya la prioridad).

La CPU incorpora además dos espacios de vectores: uno para dispositivos en chip y otro para dispositivos externos. Profundizaremos sobre este tema en la sección diferencias.

Manejo de Memoria

Rabbit incorpora una unidad de manejo de memoria (MMU) que controla el modo en que las direcciones de memoria lógica mapean en direcciones de memoria física, y una unidad de interfaz de memoria (MIU) que controla cómo las direcciones de memoria física mapean en el hardware, es decir, los chips de memoria en sí.

El compilador Dynamic C y sus bibliotecas de funciones se encargan de este mapeo de modo que la mayoría de los usuarios no necesitan estar al tanto de esto. No obstante, existe información avanzada para aquellos usuarios que necesiten manipular estos dispositivos.

Definiciones

- **Direcciones de memoria física o lineal (PA)**: Son direcciones de 20 bits que representan un espacio de 1Megabyte en el cual se mapean las direcciones lógicas. Las direcciones físicas por defecto en el sistema de desarrollo de Dynamic C son: 0x00000 para el inicio de la memoria flash y 0x80000 para el inicio de la RAM.
- **Direcciones de memoria lógica (LA)**: Son direcciones de 16 bits representando un espacio de direccionamiento de 64Kbytes. La mayoría de las instrucciones utilizan direcciones lógicas. La ubicación física donde estas direcciones mapean depende de la configuración de la MMU.
- **Segmento**: es un bloque de memoria lógica, su tamaño es múltiplo de 4K.
- **Banco**: Es un bloque de 256K de memoria física, comenzando en una posición múltiplo de 256K. Existen cuatro bancos disponibles, las posiciones de inicio son: 0x00000, 0x40000, 0x80000 y 0xC0000.

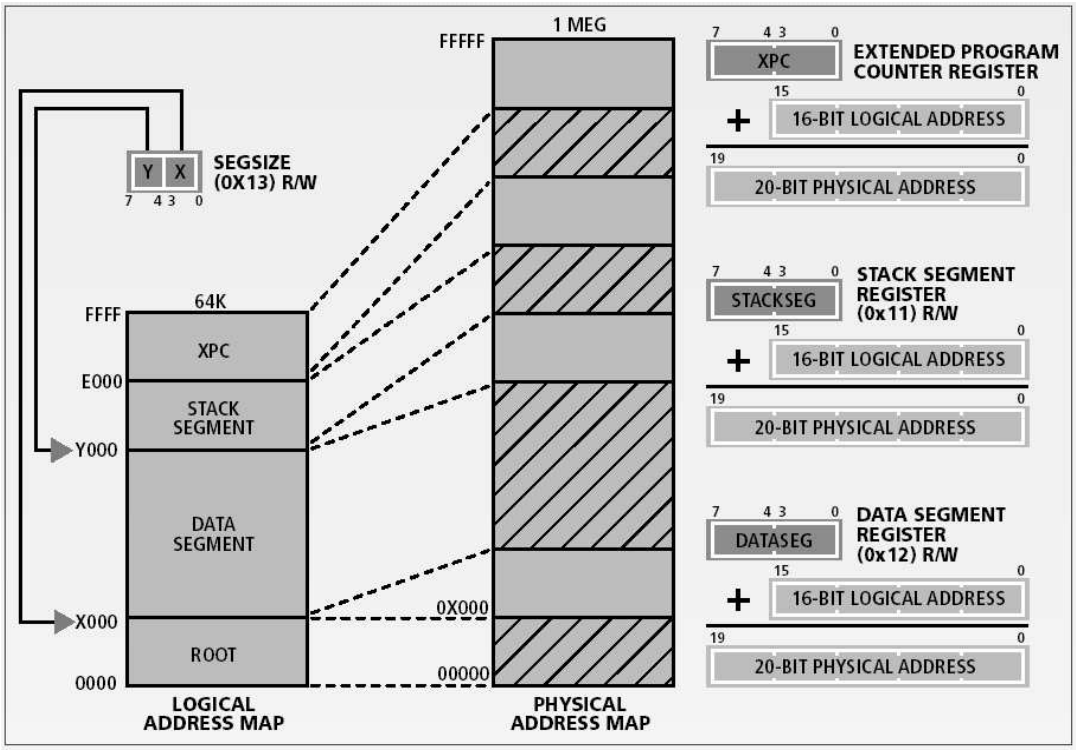
Unidad de Manejo de Memoria (MMU)

La unidad de manejo o mapeo de memoria (MMU) traduce una dirección lógica de 16 bits a una dirección física de 20 bits. El espacio de direccionamiento lógico está dividido en cuatro segmentos: *xmem*, *stack*, *data* y *base*. El segmento *xmem* siempre ocupa el área de 0xE000 a 0xFFFF. Los otros segmentos se distribuyen en el espacio restante y su tamaño es ajustable, de modo que el total de los tres es siempre 0xE000. El segmento *stack* comienza donde termina el segmento *data* y se extiende hasta 0xDFFF. El segmento *data* comienza donde termina el segmento *base*. Estos límites se configuran en un registro de la MMU llamado *SEGSIZE*, en direcciones múltiplo de 0x1000 (4K). El nibble más significativo de *SEGSIZE* decide el límite entre *data* y *stack*, mientras que el nibble menos significativo hace lo mismo entre *base* y *data*.

Cada uno de los tres segmentos superiores (*xmem*, *data*, *stack*) tiene un registro asociado que se utiliza para mapear las direcciones lógicas que caen dentro del segmento en direcciones físicas. Los mismos son, respectivamente: *XPC*, *DATASEG* y *STACKSEG*. En cada caso, la dirección física se calcula desplazando el registro de segmento 12 bits a la izquierda (multiplicar por 4K) y sumando la dirección lógica, restringiendo el resultado dentro del espacio de 20 bits. Podemos ejemplificarlo con el siguiente algoritmo:

```
Let SEGSIZE = XYh
If LA >= E000h
PA = LA + (XPC x 1000h)
Else If LA >= X000h
PA = LA + (STACKSEG x 1000h)
Else If LA >= Y000h
PA = LA + (DATASEG x 1000h)
Else PA = LA
```

Lo dicho hasta aquí puede observarse, de forma gráfica, en el diagrama siguiente:



Si bien cualquiera de los registros de segmento puede cargarse con cualquier valor válido en cualquier momento, deben manejarse con sumo cuidado. Por ejemplo, si la CPU está ejecutando código en el segmento *xmem* y se cambia el valor del registro *XPC*, la ejecución no continuará en la posición siguiente sino en la dirección en memoria física donde mapea la dirección lógica de la siguiente instrucción. Esto es así debido a que el *Program Counter* contiene una dirección lógica.

No nos interiorizaremos demasiado en este tema, pero clarificaremos conceptos con un pequeño ejemplo:

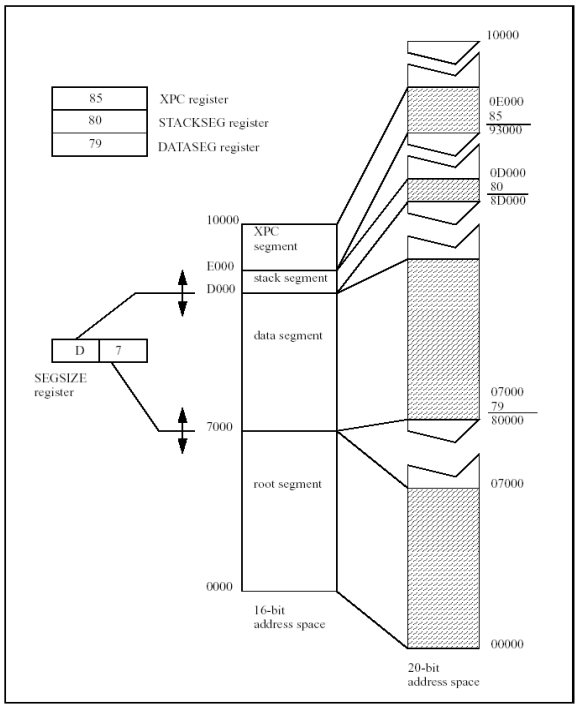
Si los registros de la MMU se setean de la siguiente forma:

XPC = 0x85

SEGSIZE = 0xD7

STACKSEG = 0x80

DATASEG = 0x79



La dirección física del piso del segmento xmem es:

0x85000 + 0x0E000 = 0x93000

La dirección física del piso del segmento stack es:

0x80000 + 0x0D000 = 0x8D000

La dirección física del piso del segmento data es:

0x79000 + 0x07000 = 0x80000

Como puede apreciarse en el diagrama que figura a la derecha.

Existen funciones especiales de Dynamic C para acceder datos en memoria física (*root2xmem* y *xmem2root*). Por lo demás, el direccionamiento es siempre en direcciones lógicas. En cuanto al assembler, existen instrucciones especiales para acceder a datos (*LDP*) o ejecutar saltos y llamadas a subrutinas (*LJP, LCALL* y *LRET*) en memoria física, estas instrucciones ignoran la MMU. El resto de las instrucciones direcciona lógicamente en 16 bits, a través de la MMU.

Unidad de Interfaz a Memoria (MIU)

La unidad de interfaz a memoria (MIU) controla el acceso a la memoria luego de que la MMU determina la dirección física. Tiene cinco registros asociados: *MMIDR*, *MB0CR*, *MB1CR*, *MB2CR* y *MB3CR*.

La función principal del registro *MMIDR* es permitir al sistema que mantenga el pin CS1 permanentemente habilitado. Esto permite acceder la RAM de forma más rápida al mantenerla siempre en su estado activo. El control de la misma se realiza entonces mediante OE y WE, las cuales no deben compartirse con otros dispositivos.

Cada uno de los registros de control de banco de memoria (Memory Bank Control Register) *MBxCR* controla un banco de 256K en el espacio físico de 1Megabyte. El control incluye inserción de wait-states (si fuera necesario), generación de CS/WE/OE y protección contra escritura. Un acceso a memoria emplea un mínimo de dos ciclos de clock para lectura y tres para escritura, pudiendo extenderse configurando wait-states adicionales en los registros de control de la MIU, sin necesidad de emplear hardware externo.

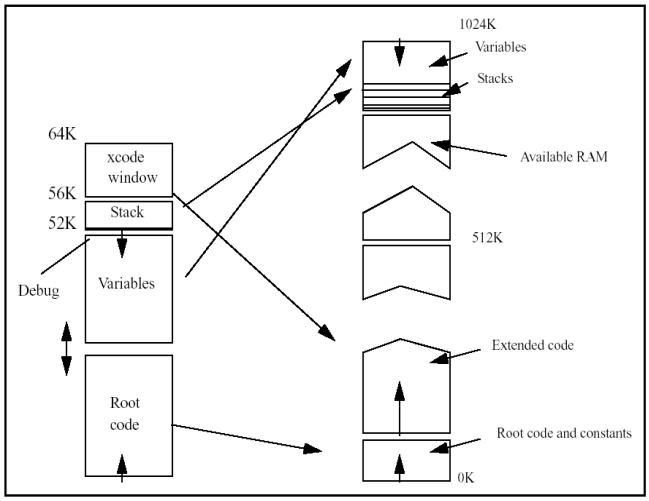
Existe, además, la posibilidad de configurar estos registros para realizar la inversión de algunas líneas de address. No desarrollaremos este tema aquí, pero comentaremos que esto permite manejar diferentes sub-bancos dentro de cada banco, y extender el direccionamiento total a 6MB sin glue-logic.

Cómo maneja la memoria el compilador de Dynamic C

Si el programa es pequeño, el código se ubica en el área denominada *root*, que sería el segmento *base*. Cuando el programa es más extenso, la mayor parte del mismo es compilado a la memoria extendida. Este código se ejecuta en una ventana de 8Kbytes en el espacio de 0xE000 a 0xFFFF, es decir, el segmento *xmem.* El registro *XPC* controla la alineación de esta ventana de 8K dentro de las 256 páginas posibles de 4K existentes en memoria física. La ventaja de este sistema es que la mayoría de las instrucciones continúan utilizando direccionamiento de 16 bits (lógico); sólo cuando es necesario transferir el control a una dirección fuera de rango es necesario utilizar instrucciones de 20 bits. Al ser el segmento de 8K y las páginas de 4K, el

código puede compilarse sin saltos, dado que al movernos a la página siguiente aún seguimos viendo la anterior dentro de la ventana del segmento (asumiendo que se ha cruzado la mitad de la página al mover la alineación de ventana).

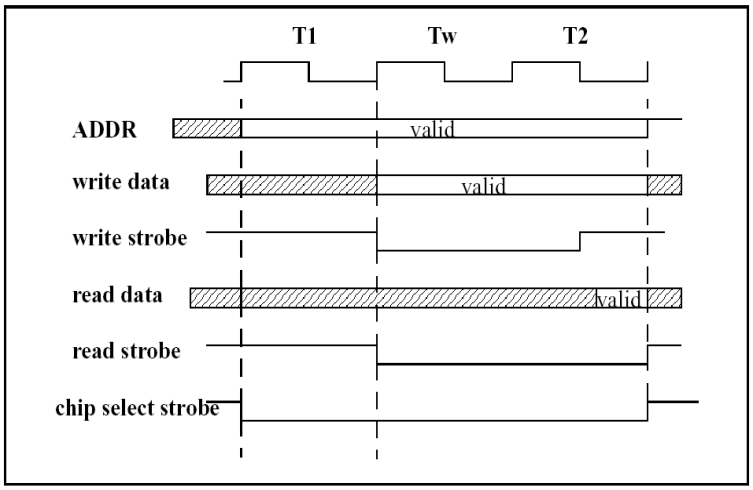
Cuando genera código en la ventana de memoria extendida, el compilador chequea si el éste pasa de la mitad de la ventana (0xF000), si es así, entonces inserta código para desplazar la ventana una página (4K) hacia abajo, de modo que el código en 0xF000+x ahora se ve en 0xE000+x. Esto da por resultado un código dividido en segmentos de entre 0 y 8K, pero típicamente 4K. La transferencia de control dentro de cada segmento se hace mediante instrucciones de 16 bits, utilizándose las instrucciones de 20 bits para transferir el control entre segmentos.



En cuanto a los datos y variables, los mismos pueden ser ubicados en el área *root* (por defecto) o en *xmem*, si se antepone la directiva correspondiente; en este caso, el área root corresponde al segmento *data*. Como hemos adelantado al analizar la MMU, las funciones especiales de Dynamic C para acceder datos en memoria física son *root2xmem* y *xmem2root,* las mismas realizan la copia entre secciones para permitir el acceso dentro del área root.

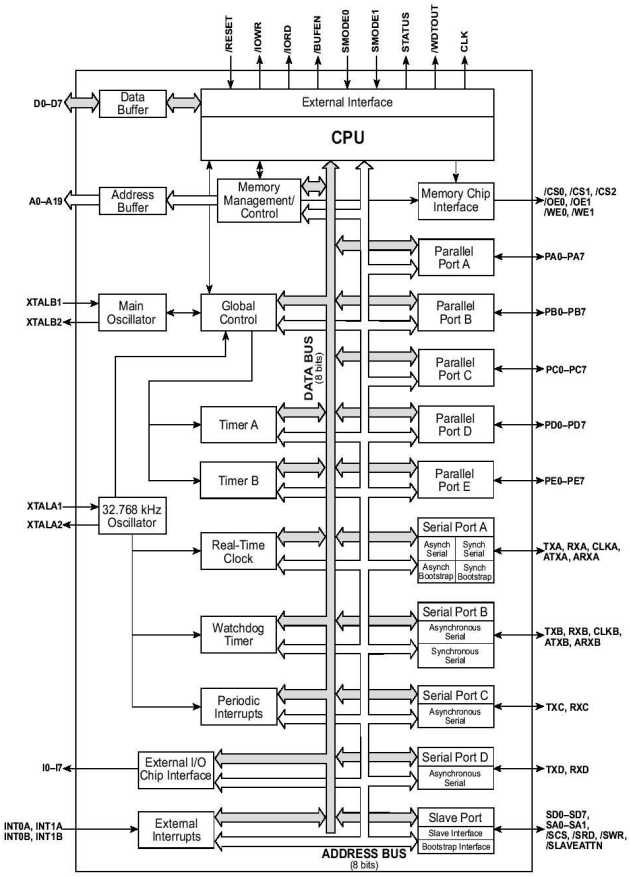
Control de bancos de I/O

Los pines del port E (que veremos en la sección sobre ports de I/O) pueden configurarse de forma independiente para funcionar como I/O strobes. Cada uno tiene un registro de control asociado que define el tipo de strobe (read, write, chip select) y la cantidad de ciclos de espera a ser insertados (si fuera necesario), así como también la posibilidad de protección contra escritura. Cada I/O strobe es activo en un espacio de direcciones correspondiente a 1/8 del espacio total externo de 64K. Esto simplifica notablemente el diseño de I/O, dado que en la mayoría de los casos tampoco se requiere glue-logic para conectar periféricos externos.



Periféricos en chip

Los periféricos a incorporar en el chip fueron elegidos tomando como base la experiencia de Z-World diseñando sistemas dedicados. Los más frecuentemente utilizados fueron incluidos en Rabbit 2000 y son: ports serie, generador de clocks, reloj de tiempo real, ports I/O, port esclavo (slave port) y timers. El diagrama que figura a continuación muestra un esquema de la estructura interna del Rabbit 2000, donde pueden observarse, entre otras cosas, los diferentes bloques internos que lo componen y los diversos periféricos que han sido incluidos en el mismo.



Ports Serie

Existen cuatro ports serie, designados como *port A,B,C,* y *D*. Los cuatro pueden operar en modo asincrónico hasta 1/32 la frecuencia de reloj del sistema. Soportan 7 ú 8 bits de datos, más un modo especial para señalizar inicio de mensaje mediante un noveno bit.

En muchas UARTs, incluyendo las del Z180, determinar cuándo se ha terminado de transmitir el último byte de un mensaje resulta difícil. Esto es muy importante en comunicación half-duplex, como por ejemplo RS-485, en que el software debe detectar esta condición y cambiar de modo transmisión a modo recepción, controlando los transceivers. Con Rabbit, esto resulta simple de realizar, debido a que existe un bit de estado que señala expresamente esta condición.

Los ports serie no soportan bit de paridad y múltiples bits de stop directamente en el hardware, pero pueden simularse con drivers adecuados.

Los ports A y B pueden operar además en modo sincrónico, hasta 1/6 de la frecuencia de clock del sistema, ó 1/4 si es el Rabbit quien provee el reloj.

El port A tiene una característica especial: puede ser utilizado para bootear el sistema después de un reset. Esta es la condición normal de desarrollo en entorno Dynamic C.

System Clock

El circuito principal de reloj utiliza un cristal o resonador externo. Las frecuencias típicas cubren un rango de 1,8 a 29,5MHz. En casos en que la precisión necesaria pueda obtenerse del oscilador de 32,768KHz, es posible utilizar un resonador cerámico de tipo económico con resultados satisfactorios. La frecuencia del oscilador de reloj puede duplicarse o dividirse por 8, a modo de elegir entre velocidad de ejecución y potencia consumida. El reloj de I/O se controla por separado, de modo que la operación anterior no afecta el timing de los dispositivos periféricos como por ejemplo el baud rate de los ports serie. Para funcionamiento a ultra bajo consumo, el reloj del procesador puede tomarse del oscilador de 32,768KHz, apagando el oscilador principal. Esto permite que el procesador opere a aproximadamente unas 10.000 instrucciones por segundo, lo cual se

presenta como una alternativa frente a la opción de directamente interrumpir la ejecución, como hacen otros procesadores.

El oscilador de 32,768KHz se utiliza además para el watchdog timer y para generar el baud rate clock del port A al bootear por el port serie (modo asincrónico).

Reloj de Tiempo Real

El reloj de 32,768KHz alimenta un contador de 48 bits que oficia de reloj de tiempo real (RTC). Este contador posee un pin diferente para su alimentación, permitiendo el uso de pila de respaldo para el RTC mientras el resto del chip se encuentra sin alimentación.

El contador puede setearse y leerse por software y permite mantener información unívoca por un período de más de 100 años[1]. Si bien su lectura puede resultar un poco complicada, particularmente si el procesador también se encuentra funcionando a 32,768KHz, debido a que se trata de un ripple counter; existen funciones de Dynamic C que resuelven esta incomodidad.

Ports I/O paralelo

Rabbit 2000 dispone de 40 líneas de entrada/salida, repartidas en cinco ports de 8 bits designados como *port A*, *B*, *C*, *D*, y *E*. La mayoría de los pines utilizados tiene funciones alternativas, como port serie o chip select strobe.

Los ports D y E tienen la capacidad de sincronizar sus salidas con un timer. Estando compuestos por dos registros en cascada, una escritura en el port carga el registro de primer nivel, mientras que la transferencia al segundo nivel (el verdadero port de salida) se realiza comandada por una señal de uno de los timers. Esto, además, puede generar una interrupción que puede utilizarse para preparar el próximo estado del port al instante siguiente. Esta característica puede utilizarse para generar pulsos cuyos flancos pueden posicionarse con gran precisión. Las aplicaciones posibles incluyen, entre otras, señalización de comunicaciones, modulación de ancho de pulso y control de motores paso a paso.

El port D permite, además, utilizar sus salidas como open drain, para aplicaciones en lectura de teclados, por ejemplo.

Slave Port

El port esclavo o *slave port* está diseñado para permitir que el Rabbit pueda funcionar como esclavo de otro procesador, el cual puede, o no, ser otro Rabbit. Comparte sus pines con el port A y es bidireccional.

El port está compuesto por tres registros de lectura y tres de escritura, seleccionados mediante dos líneas de selección, que forman la dirección del registro, un read strobe (que ocasiona la transferencia de los datos en el registro al port en sí) y un write strobe (que ocasiona la transferencia de los datos en el port al registro interno). Los mismos registros pueden escribirse y leerse respectivamente como registros de I/O por el Rabbit (slave). El Rabbit puede enterarse que el master ha escrito un registro observando los bits de estado en un registro interno, o configurando una interrupción ante esta situación. Cuando el Rabbit (slave) escribe un registro, una línea de atención cambia de estado, permitiendo que el master detecte esta condición.

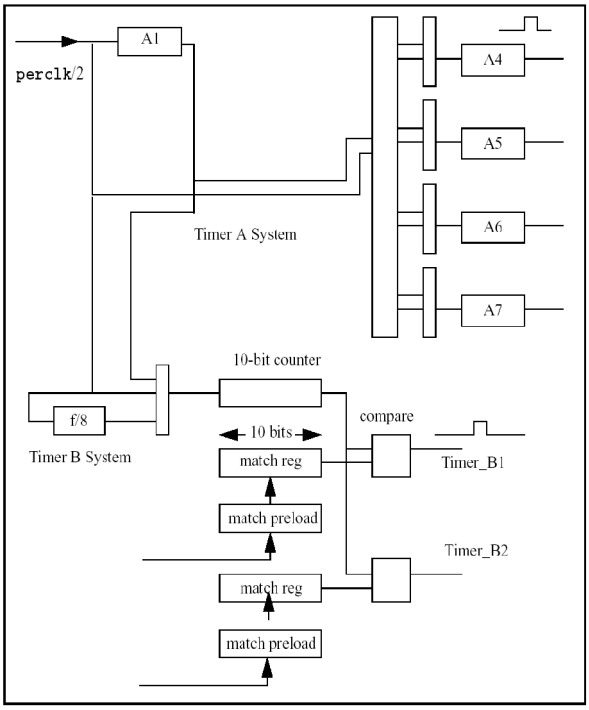
El Rabbit puede bootear de este port, permitiendo operación con RAM solamente, obteniendo el código del procesador maestro en el momento de arranque.

Timers

Existen varios sistemas de timer. Uno de ellos divide el reloj del oscilador de 32,768KHz por 16, obteniendo una señal de 488us que puede habilitarse como interrupción periódica. Sin embargo, cuando nos referimos genéricamente a los timers, lo hacemos específicamente a dos grupos o sistemas denominados *Timer A* y *Timer B*.

El diagrama a continuación muestra la estructura de los mismos.

1 El espacio de 2^{48} cuentas, a una frecuencia de 32,768KHz, es suficiente como para 272 años. Por convención, se toma como cero las 0 hs del 1º de enero de 1980, y además, el software ignora el bit más significativo, el contador tiene espacio para 136 años a partir de la fecha cero, es decir, poco más de 112 años a partir del momento de escribir este documento.



El *timer A* está compuesto por cinco registros de 8 bits de cuenta regresiva, que pueden recargarse, pudiendo ser conectados en cascada hasta dos niveles. Cada uno de estos registros puede configurarse para dividir por un número entre 1 y 256. La salida de cuatro de estos timers se utiliza para proveer los baud clocks de los ports serie. Cualquiera de ellos puede generar interrupciones, A1 puede controlar la transferencia sincronizada de los ports D y E.

El *timer B* está compuesto por un contador de 10 bits que puede leerse pero no escribirse y dos registros de comparación de 10 bits que generan un pulso cuando el valor del contador iguala al del registro. De este modo, el timer puede programarse para generar un pulso a una determinada cuenta en el futuro. Este pulso, a su vez, puede utilizarse para generar una interrupción o controlar la transferencia sincronizada de los ports D y E.

Bootstrap

El Rabbit provee la opción de elegir la fuente de donde carga su programa, al reiniciar. Esto se realiza mediante la combinación de dos pines: *SMODE0* y *SMODE1*. La combinación 00 corresponde a inhabilitar la operación de bootstrap, lo cual significa que el dispositivo opera normalmente, arrancando en la posición 0 y esperando leer allí una memoria no volátil con instrucciones.

La operación de bootstrap inhibe el funcionamiento normal y desvía la ejecución a una ROM interna, que provee un programa que controla la operación de bootstrap, leyendo grupos de tres bytes del periférico seleccionado: dirección (byte alto, byte bajo) y dato correspondiente, procediendo a escribir el dato en la posición especificada. El bit más significativo de la dirección especifica si el destino es memoria (0) o I/O interno (1), permitiendo configurar al procesador para el hardware. Todas las funciones de mapeo y control de memoria operan normalmente durante el bootstrap, esto permite acceder a toda la memoria física. El programa en ROM espera a que el periférico tenga datos disponibles, pero como el watchdog timer también está en operación, y es reseteado cada vez que llega un dato, éstos deben llegar a frecuencia suficiente como para impedir el reset.

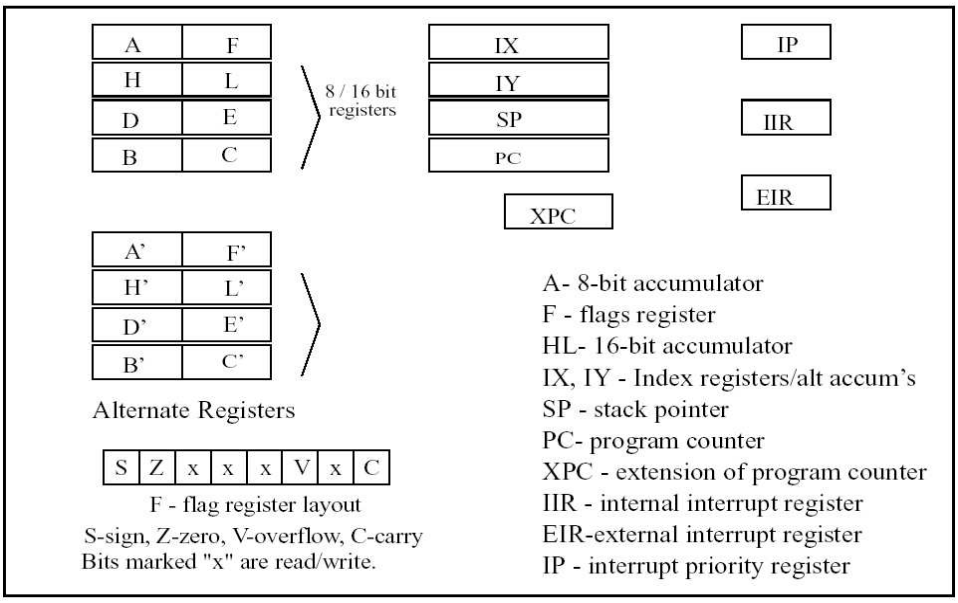
Las opciones de bootstrap son:

- Slave port: utiliza los ports A, B y E para el mismo. Sólo el registro 0 es utilizado
- Serial port A sincrónico: el clock para la interfaz serie debe proveerse externamente.
- Serial port A asincrónico: el reloj se toma del oscilador de 32,768KHz, dando una referencia de 2400bps.

Diferencias

Registros

Los registros del Rabbit son casi idénticos a los del Z80 ó Z180. Los mismos pueden observarse en el diagrama que figura a continuación. Los registros *XPC* e *IP* son nuevos; el registro *EIR* es equivalente al registro *I* del Z80 y se utiliza para apuntar a la zona donde están las rutinas de interrupción para interrupciones generadas de forma externa. El registro *IIR* ocupa la posición lógica que ocupaba el registro *R* en el Z80, pero su función es apuntar a la zona de rutinas de interrupción para interrupciones generadas internamente.



Una característica interesante e innovadora del Z80 fue su set alternativo de registros. Su función era facilitar el intercambio de contexto (context switching) mediante una forma rápida de salvar todos los registros de la CPU, por este motivo, solamente se previeron instrucciones de intercambio de sets. El Rabbit incorpora nuevas instrucciones que mejoran la comunicación entre ambos sets de registros, duplicando efectivamente la cantidad de los mismos. La intención no es que se utilice el set alternativo como set de registros para una rutina de interrupciones, ni es esta condición soportada por Dynamic C, dado que éste utiliza ambos sets de registros libremente.

El registro *XPC* ha sido descripto al analizar la MMU.

El registro *IP* es el encargado de la prioridad de interrupciones (Interrupt Priority Register); contiene cuatro campos de 2 bits que guardan la historia de la prioridad a la que estuvo ejecutando código la CPU. El Rabbit soporta cuatro niveles de prioridad, e instrucciones especiales para actualizar este registro. Ampliaremos este tema en la sección sobre interrupciones, más adelante.

Instrucciones

El Rabbit es altamente compatible con el Z80 y el Z180, en cuanto a código se refiere, y resulta fácil portar código que no dependa del uso de I/O. Las áreas de mayor incompatibilidad se centran donde encontramos instrucciones de I/O o implementaciones particulares de hardware. Excepto una instrucción en particular, en la que se ha cambiado el opcode, todas las instrucciones de Z180 que se han mantenido tienen compatibilidad binaria (idéntico opcode). Las instrucciones más importantes que se han eliminado son simuladas automáticamente por una secuencia de instrucciones en el assembler de Dynamic C. Sin embargo, algunas instrucciones no muy útiles han sido eliminadas y no es posible simularlas fácilmente; el código que utilice estas instrucciones deberá re-escribirse. Se trata de las siguientes instrucciones:

```
DAA, HALT, DI, EI, IM 0, IM 1, IM 2, OUT, IN, OUT0, IN0, SLP, OUTI,
IND, OUTD, INIR, OTIR, INDR, OTDR, TESTIO, MLT SP, RRD, RLD, CPI,
CPIR, CPD, CPDR
```

La mayoría de estas instrucciones están relacionadas con I/O y representan código que no es transportable. Las únicas que no caben en esta categoría son *MLT SP, DAA, RRD, RLD, CPI, CPIR, CPD,* y *CPDR.* La instrucción *MLT SP* no es en sí de mucha aplicación práctica. En cuanto a las concernientes a aritmética

decimal (*DAA*, *RRD* y *RLD*), sería posible simularlas, pero la simulación no sería muy eficiente[2]. Usualmente, el código que utiliza estas instrucciones debe re-escribirse. Las instrucciones de comparación repetida (*CPI*, *CPIR*, *CPD* y *CPDR*), también resultaría ineficiente simularlas y se sugiere volver a escribir el código que las utiliza, lo cual, en la mayoría de los casos, no debería revestir dificultad alguna.

Respecto a las instrucciones *RST*, tres de ellas (*RST 0*, *RST 8* y *RST 30h*) han sido eliminadas. Las restantes se han mantenido, pero la dirección de ejecución corresponde a una dirección variable, cuya base es determinada por el registro *EIR*. Si bien los *RST* pueden ser simulados por instrucciones *CALL*, esto no se hace automáticamente en el assembler dado que Dynamic C utiliza la mayoría de estas instrucciones para propósitos de depuración (debugging).

Existe una instrucción cuyo opcode se ha cambiado:

```
EX (SP),HL  ; pasa de E3 a ED 54
```

Estas instrucciones utilizan nombre de registro diferentes:

```
LD A,EIR
LD EIR,A    ; era registro R
LD IIR,A
LD A,IIR    ; era registro I
```

Por último, las instrucciones que siguen a continuación han sido eliminadas y no están soportadas. En todos los casos, existe una secuencia alternativa de instrucciones Rabbit que provee la misma función:

```
Z80                 Rabbit
CALL cc,ADR               JR (JP) ncc,xxx ; condición inversa
                          CALL ADR
                    xxx:

TST r ((HL),n)      PUSH DE
                    PUSH AF
                    AND r ((HL), n)
                    POP DE            ; Carga A original en D
                    LD A,d
                    POP DE
```

Instrucciones de I/O

El Rabbit emplea un esquema totalmente diferente para el acceso de entradas/salidas, respecto a otros microprocesadores de 8 bits. Cualquier instrucción de acceso a memoria puede ser precedida por uno de dos prefijos posibles que, de utilizarse, transforman a la instrucción en una instrucción de I/O. Un prefijo elige espacio de I/O interno (periféricos en chip), y el otro espacio externo (direccionamiento en bus). Veamos algunos ejemplos:

```
IOI LD A,(85h)    ; A = contenido de registro interno en dirección 0x85
LD IY,4000h
IOE LD HL,(IY+5)  ; HL = contenido leído en I/O externo, dirección 0x4005
```

Debido al uso de los prefijos, es posible utilizar cualquiera de las instrucciones que normalmente acceden a memoria para acceder espacio de I/O. Como es de esperar, el sistema de mapeo de memoria (MMU+MIU) no funciona en este caso, y el espacio lógico y físico se funden en uno solo. El timing de acceso es de dos clocks para espacio interno y tres para espacio externo, este último con posibilidad de agregar wait-states en espacios definidos como I/O strobe.

Instrucciones privilegiadas

Normalmente, una interrupción se atiende al final de la instrucción en curso. Sin embargo, si esa instrucción es una de las llamadas *privilegiadas*, se demorará el servicio hasta que se haya ejecutado una instrucción *no-privilegiada*. Esto se ha hecho así para permitir que ciertas operaciones sean realizadas de forma atómica, evitando el inhibir explícitamente las interrupciones en secciones de código donde sería un derroche de tiempo, o imposible, debido a que el propósito de la operación es justamente manipular las interrupciones.

2El bit en el status register utilizado para el medio acarreo o acarreo parcial (half carry) se encuentra disponible y puede setearse y borrarse utilizando las instrucciones *PUSH AF* y *POP AF* para accederlo.

Podemos agrupar estas instrucciones según su función.

Manejo del stack

```
LD SP,HL
LD SP,IY
LD SP,IX
```

Estas instrucciones permiten que pueda ejecutarse luego una instrucción de manejo de algún registro de segmento, de forma atómica, por ejemplo:

```
LD SP,HL
IOI LD SSEG,A
```

Manipulación del registro IP: secciones críticas

```
IPSET n     ; desplaza IP a la izquierda y setea prioridad n en bits 1,0
            ; n puede ser 0,1,2 ó 3
IPRES       ; desplaza IP 2 bits a la derecha, restablece prioridad previa
RETI        ; extrae IP del stack y luego extrae la dirección de retorno
POP IP      ; extrae IP del stack
```

En secciones críticas, puede ser necesario inhabilitar las interrupciones. El ejemplo de código a continuación muestra cómo inhabilitar interrupciones de prioridad 1, subiendo la prioridad de ejecución a 1:

```
IPSET 1     ; salva prioridad previa y asigna prioridad 1
;....sección critica....
IPRES       ; restablece prioridad previa
```

Esto es seguro sólo si el código en la sección crítica no contiene, a su vez, otra sección crítica... Si este código es anidado, existe el riesgo de desbordar el registro IP. El ejemplo siguiente muestra cómo puede resolverse el problema:

```
PUSH IP     ; salva prioridad previa en stack
IPSET 1     ; salva prioridad previa (no interesa) y asigna prioridad 1
;....sección critica....
POP IP      ; restablece prioridad previa
```

Acceso al registro XPC: saltos largos calculados

```
LD A,XPC
LD XPC,A
```

Las instrucciones de acceso al registro XPC se han hecho privilegiadas de forma de permitir saltos largos a direcciones calculadas:

```
LD XPC,A
JP (HL)
```

En este caso, *A* contiene el nuevo valor de *XPC* y *HL* el de *PC*. Este tipo de código debe ejecutarse en el segmento *root*, de modo que la ejecución de *LD XPC,A* no ocasione un desplazamiento de la ventana y no pueda ejecutarse *JP (HL)*, perdiendo el control de la CPU.

El siguiente ejemplo muestra cómo realizar una llamada a una subrutina cuya dirección se ha calculado previamente:

```
; A=XPC, IY=address
LCALL DOCALL            ; llama a rutina utilitaria en segmento root
;
; Rutina DOCALL
DOCALL:
LD XPC,A                ; carga XPC
JP (IY)                 ; salta a subrutina
```

El regreso, a su vez, deberá hacer uso de la instrucción *LRET* para restablecer el valor original del registro *XPC* antes de la ejecución de *LCALL*.

Uso de semáforos

```
BIT b,(HL)
```

Esta instrucción se ha hecho privilegiada de modo de permitir la construcción de un semáforo mediante código como el siguiente:

```
BIT b,(HL)              ; comprueba estado de un bit particular en memoria
SET b,(HL)              ; setea el bit sin alterar el flag Z
; si Z=1, el recurso es nuestro (el bit era 0 y lo seteamos)
; si Z=0, el bit ya era 1, significa que alguien ya controlaba el recurso
```

Utilizamos semáforos para controlar el acceso a recursos que solamente pueden ser utilizados por una tarea a la vez. Esto se realiza seteando el bit para indicar la posesión o pertenencia del recurso. No pueden permitirse interrupciones entre la operación de test y la de set, debido a que esto podría ocasionar que dos tareas diferentes crean ambas que poseen control de ese recurso.

Interrupciones

Las interrupciones en Z80 y Z180 tienen dos niveles de prioridad: *enmascarables* y *no-enmascarables*. La interrupción no-enmascarable (NMI, Non-Maskable Interrupt) no puede inhabilitarse y su recepción ocasiona el salto a una posición fija en memoria (0x66) donde se ejecuta la rutina de interrupción. La enmascarable, puede ser inhabilitada por programa, y puede funcionar en uno de tres modos: *modo 0*, en el que el periférico debe proveer una instrucción válida en el bus; *modo 1*, en el que se transfiere la ejecución a la posición 0x38; y el *modo 2*, o vectorizado, en el que el periférico provee la parte baja de la dirección del vector (siendo la parte alta provista por el registro *I*), del cual extraerá la dirección del handler.

El Rabbit, en cambio, tiene tres niveles de prioridad de interrupciones y cuatro niveles de prioridad de ejecución. Una interrupción sólo se tomará en cuenta si la prioridad de la misma es superior a la prioridad en la que se encuentra funcionando en ese momento la CPU. Si bien generalmente la interrupción es atendida al final de la instrucción en curso, existen instrucciones denominadas "privilegiadas", las mismas demoran la atención de las interrupciones a la instrucción subsiguiente, permitiendo operaciones de stack, context switching y semáforos con mayor facilidad y seguridad.

El Rabbit no utiliza vectores sino offset fijos[3]. Según se trate de una interrupción causada por un dispositivo interno o externo, la CPU provee la dirección base en el registro *IIR* o *EIR* respectivamente, y el periférico determina el offset. La dirección así formada contiene el inicio de la rutina de interrupciones, en vez de su dirección como en el modo *2* del Z80. El offset de cada periférico interno es fijo para cada dispositivo, y los periféricos externos disponen de pines separados para su identificación. Las posiciones usadas están separadas cada 16 bytes, de modo de permitir que las rutinas pequeñas entren en su totalidad, sin saltos.

Las direcciones mapean en el área *root*, sin embargo, pueden realizar saltos a *xmem* de ser necesario. El registro *XPC* deberá ser salvado en el stack para permitir el retorno posterior a la condición de ejecución previa.

Las interrupciones tienen prioridad 1, 2 ó 3. El procesador opera a prioridades 0, 1, 2 ó 3. Si ocurre una interrupción con una prioridad mayor a la prioridad corriente del procesador, ésta será atendida luego de la instrucción en curso (excepto instrucciones privilegiadas). La interrupción automáticamente eleva la prioridad del procesador a la suya, siendo la prioridad anterior salvada en el stack de cuatro posiciones contenido en el registro *IP*. Este registro puede ser desplazado a la derecha, restableciendo la prioridad anterior, mediante una instrucción especial: *IPRES*. Debido a que sólo la prioridad corriente y tres anteriores pueden ser contenidas en el registro *IP*, se proveen además instrucciones especiales para salvar y recuperar este registro en el stack (*PUSH IP*, *POP IP*). Es posible, además, forzar una nueva prioridad con otra instrucción especial: *IPSET n* (n=0,1,2,3).

La prioridad de una interrupción se establece, generalmente, mediante algunos bits en un registro de control asociado al hardware que crea la interrupción. El registro *IP* contiene la prioridad actual del procesador en los dos bits menos significativos. Al momento de producirse la interrupción, este registro es desplazado a la izquierda dos posiciones y los dos bits menos significativos se setean con el valor de la prioridad de la interrupción. Esto resulta en que una rutina de interrupciones sólo puede ser interrumpida por otra de mayor prioridad (a menos que el programador la disminuya explícitamente).

3 La literatura de Rabbit considera a estas posiciones como vectores; aunque, estrictamente, se trata de offsets