**INSTITUTO POLITÉCNICO DE BRAGANÇA**
**ESCOLA SUPERIOR DE TECNOLOGIA E DE GESTÃO**

# Sistemas Digitais II

**Filipe Moreira**

# 1. Detecção e Correcção de Erros

## *1.1 Paridade*

O modo mais simples de detectar erros de um só bit é adicionar um bit de paridade a cada byte transmitido. Esta *verificação redundante* é feita de modo a que o número total de bits com o valor lógico "1" (incluindo o bit de paridade) seja par (paridade par) ou ímpar (paridade ímpar).

Este processo é muito simples de implementar e pode ser mostrado que a adição de um bit de paridade melhora a integridade da informação num factor de 357. Contudo este processo não contempla erros em múltiplos bits.

## *1.2 Checksum*

Devido à necessidade de adicionar mais um bit por cada byte transmitido, a paridade diminui a taxa de caracteres bem como requer mais memória, podendo atingir proporções consideráveis.

Assim sendo, tornou-se popular o *checksum* aquando da transferência de blocos de informação. O *checksum* é usualmente um byte enviado como o último byte num bloco de informação. O receptor calcula a soma do bloco de informação recebido (que inclui o byte de checksum) e compara-o com o byte de *checksum*; se não se equivalerem é pedida a retransmissão do bloco.

Isto traz como vantagem um menor *overhead* quando comparado com a paridade. Outra vantagem é o facto de os erros introduzidos em blocos de informação aquando da transmissão, ocorrerem em explosões (*bursts* ) (Ex.: um relâmpago ou um interruptor com ruído). Como o *checksum* é característico de todo o bloco, é mais provável detectar esses erros do que com um simples teste de paridade.

## *1.3 Cyclic Redundancy Checks (CRC)*

Esta técnica é usada na detecção de erros num bloco de informação. É usualmente usada aquando da leitura e escrita em disquetes e para garantir a integridade da informação em EPROM's.

Mas, ao contrário da *checksum*, o CRC não é *byte-oriented*, ou seja, não é necessário que o bloco de informação seja constituído por bytes. Neste caso o bloco de informação é visto como uma "cadeia" de bits, sendo os *n* bits deste bloco considerados como os coeficientes de um **polinómio característico** (usualmente referido como *M(X)*). Este *M(X)* será da forma:

$$M(X) = b_n + b_{n-1}X^1 + b_{n-2}X^2 + \ldots + b_1X^{n-1} + b_0 X^n$$

onde $b_0$ é o bit menos significativo (LSB) e $b_n$ é o bit mais significativo (MSB).

Ex.: Cálculo de *M(X)* para a cadeia de informação 26F0h.

26F0 = 0 0 1 0 0 1 1 0 1 1 1 1 0 0 0 0

$M(X) = X^2 + X^5 + X^6 + X^8 + X^9 + X^{10} + X^{11}$

Este polinómio *M(X)* é característico do bloco 26F0h. Se qualquer bit mudasse então haveria uma mudança do polinómio. Os bytes CRC são calculados por aplicação da seguinte equação:

$$\frac{M(X)\times X^n}{G(X)} = Q(X) + R(X)$$

Nesta equação G*(X)* é denominado por polinómio gerador. Para o protocolo bysinc é

$$G(X) = X^{16} + X^{15} + X^2 + 1$$

enquanto que o protocolo SDLC usa

$$G(X) = X^{16} + X^5 + X^{12} + 1$$

Quando se efectua esta divisão o resultado será um quociente *Q(X)* e um resto *R(X)*. A técnica CRC consiste em calcular *R(X)* para a cadeia de informação e de seguida juntar *R(X)* ao bloco de informação. Quando o resultado é calculado no receptor deve ser *R(X)* = 0. Dado o facto de *G(X)* ter ordem 16, então *R(X)* não pode ser de ordem superior a 15, representando deste modo 2 bytes, independentemente do comprimento do bloco de informação original.

### *1.4 Código de Hamming*

As técnicas descritas até agora só efectuam a detecção de erros. Contudo Hamming enunciou uma técnica que não só detectava os erros mas como também os corrigia. Como resultado, os códigos de Hamming passaram a ser a base de todos os esquemas de correcção erros actualmente utilizados.

Esta tarefa consiste em realizar múltiplos testes de paridade em cada palavra de informação. Os bits adicionais são transmitidos juntamente com a palavra. A Fig. 1.1 ilustra esta técnica para uma palavra de 8 bits. Para tal são necessários 4 bits (P0 a P3).

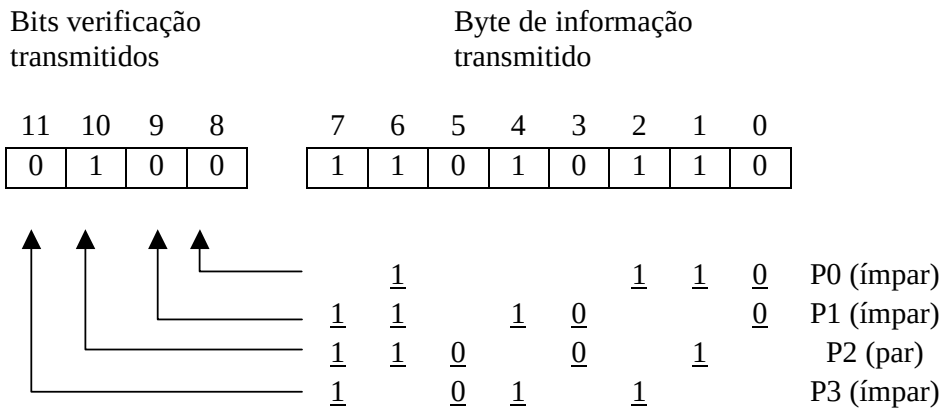



**Fig. 1.1**

É de referir que cada bit de paridade verifica um conjunto de bits diferente e que cada bit é verificado por pelo menos dois bits de verificação. Neste exemplo o byte de informação D6h seria transmitido como o número de 12 bits 4D6h.

A Fig. 1.2 mostra como é que a palavra recebida é testada. São novamente gerados 4 bits de paridade, mas desta vez incluem-se os 4 bits de verificação gerados pelo transmissor.

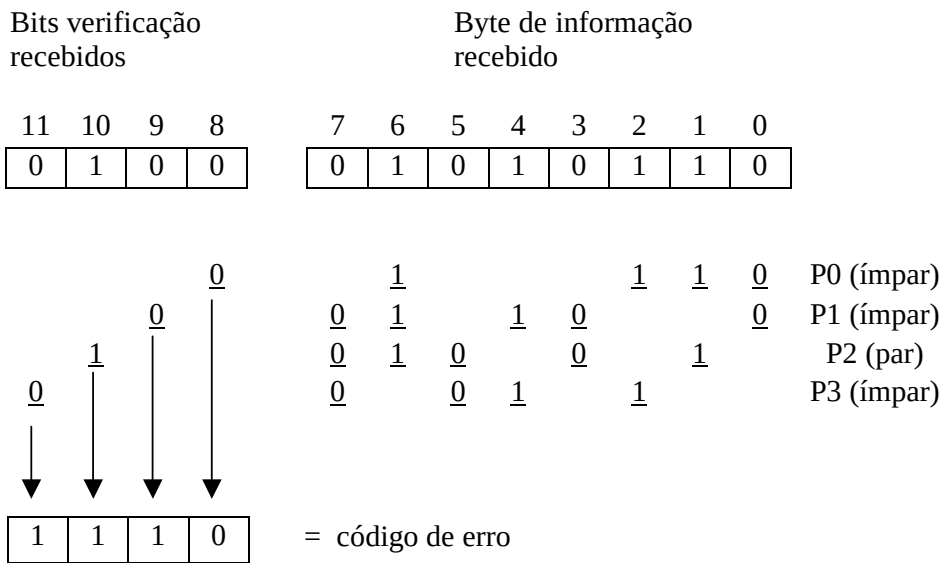



**Fig. 1.2**

Este exemplo foi feito de modo a ter sido introduzido um erro na transmissão (neste caso no bit 7). Os quatro bits gerados são denominados de código de erro. O seu significado pode ser explicado por uma tabela de erros para código de Hamming modificado para 8 bits. Um código de 0000 indica que não há erros detectados. Neste exemplo o código de erro é 1110, significando que o bit 7 está errado (ver tabela). Se o verdadeiro código de Hamming for utilizado então o código de erro deverá identificar qual a posição do bit errado. Por este motivo a técnica aqui ilustrada chama-se *Código de Hamming modificado*.

Apesar de a adição de 4 bits representar um acréscimo de 50%, esta técnica irá detectar e corrigir todos os erros de um só bit e consecutivamente 97% de todos os erros para múltiplos bits. Contudo quando se expande a palavra para 16 bits, só são necessários 5 bits de verificação,. Reduzindo-se, assim, o *overhead* para 32%.

Existem vários CI capazes de implementar esta técnica.

## 2. Memórias

Memórias são componentes destinados a guardar informação, geralmente sob forma binária, manipulada por sistemas digitais. Normalmente são constituídas por um conjunto de elementos de dois estados.

Existem três grandes tipos de memórias de semicondutores:

Memória de Acesso Aleatório (RAM)
Memória só de Leitura (ROM)
Memória de Série

Destas memórias algumas são voláteis, isto é, perdem a informação armazenada quando se desliga a alimentação (Ex.: RAM). Outras pelo contrário não perdem essa informação na ausência de alimentação (Ex.: ROM).

Das memórias enunciadas, nesta disciplina só se irão estudar as RAMs.

As memórias de acesso aleatório (Random Acess Memory) são aquelas memórias para as quais só é possível ler e/ou alterar cada posição de armazenagem. O nome mais correcto para este tipo de memória seria o Leitura/Escrita (Read/Write) uma vez que outras memórias, como as ROM, também são de acesso aleatório (random acess). Uma memória diz-se de acesso aleatório se se puder atingir directamente qualquer uma das suas posições. Pelo contrário, uma memória diz-se de acesso sequencial ou série quando é necessário passar por todas as posições para atingir uma qualquer posição.

Há dois tipos fundamentais de RAM's: estáticas e dinâmicas. As Estáticas são aquelas em que a informação é guardada pelo estado de condução ou não dum transístor ou conjunto de transístores. As dinâmicas são aquelas em que a informação é guardada pelo estado de carga ou descarga dum condensador. Nestas últimas, devido às fugas que sempre apresenta o condensador, torna-se necessário 'avivar' a informação armazenada com uma frequência estabelecida pelo fabricante – *refreshing*.

### *2.1 RAM's Estáticas*

O esquema simplificado de uma memória RAM estática está representado na Fig. 2.1.

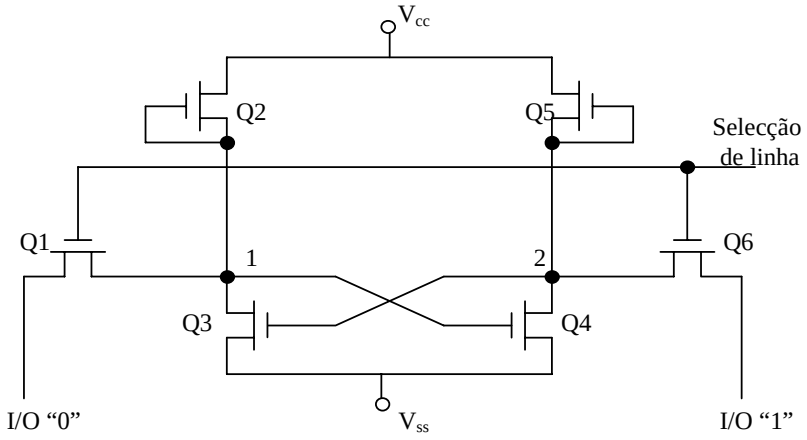



**Fig. 2.1**

A célula tem como componentes "activos" os transístores Q3 e Q4, funcionando Q2 e Q5 como cargas activas, sendo estes do tipo 'depletion'; tal categoria de MOS requer uma tensão de gate negativa para bloquear o transístor, o qual estará, portanto, normalmente em condução. O funcionamento desta célula pode ser descrito da seguinte

forma: suponha-se que a gate de Q3 está positiva colocando, assim, Q3 em condução (ON) o que determina que circulará corrente através de Q2 e Q3. Q2 e Q3 são fabricados de modo a que Q2 tenha maior impedância que Q3 pelo que a tensão no nodo 1 cairá para um valor próximo de $V_{ss}$. Note-se que a gate de Q2 está ligada ao nodo 1; deste modo à medida que a tensão do nodo 1 vai decrescendo, diminuirá o estado de condução de Q2, aumentando por conseguinte a impedância efectiva de Q2. Isto permite que a tensão do nodo 1 aproximar-se ainda mais de $V_{ss}$. Com o nodo 1 em zero a gate de Q4 também estará em zero pelo que Q4 estará em corte (OFF). A carga de Q3 é mantida por Q5. Note-se que através de Q5 apenas passarão correntes de fugas o que terá um efeito mínimo na tensão do nodo 2. Quanto mais positiva for a tensão do nodo 2 maior será o estado de condução de Q5. A tensão no nodo 2 será deste modo igual a $V_{cc}$ (nos transístores MOS com 'depletion' não há queda de tensão entre fonte e dreno).

Por definição o valor lógico "0" será guardado na célula de memória se Q3 estiver ON e o valor lógico "1" se Q4 estiver ON.

As células estão organizadas segundo uma matriz de n linhas x m colunas.

Como exemplo representa-se, a seguir, a organização de uma memória com 1024 posições. Pode ver-se que está organizada segundo uma matriz de 32 linhas x 32 colunas.

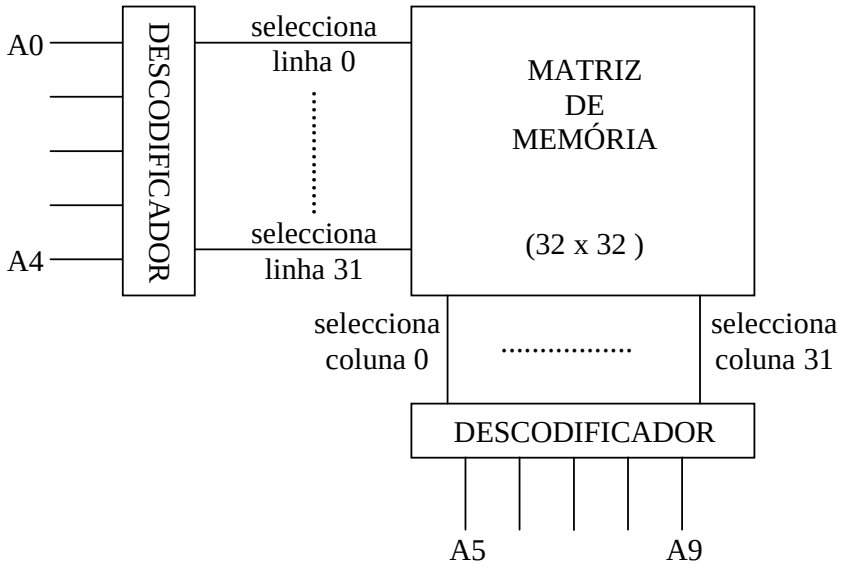



**Fig. 2.2**

A selecção de uma célula de memória é feita através da coincidência de selecção de uma linha (A0 – A4, 1 de 32) e duma coluna (A5 – A6, 1 de 32).

### *2.1.1 Acesso à célula de memória*

O acesso à célula de memória é efectuado do seguinte modo:

A célula de memória é interrogada por uma operação de leitura ou de escrita activando o correspondente sinal de selecção de linha o que colocará em condução Q1 e Q6.

Para uma operação de leitura, um *sense amplifier* é ligado às saídas I/O “0” e I/O “1” de cada coluna e detecta o estado da célula seleccionada nessa coluna (ver Fig. 2.3). Se Q3 estiver em condução (estado lógico “0”) a corrente circulará na linha I/O “0”. Se Q4 estiver em condução (valor lógico “1”) a corrente circulará em I/O “1”.

Para se escrever na célula um buffer de escrita aplica uma tensão próxima de $V_{cc}$ na linha I/O “0” para escrever o valor lógico “0” mantendo a linha I/O “1” ligada a $V_{ss}$; este buffer aplica um tensão próxima de $V_{cc}$ na linha I/O “1” para escrever o valor lógico “1” mantendo, neste caso, a linha I/O “0” ligada a $V_{ss}$.

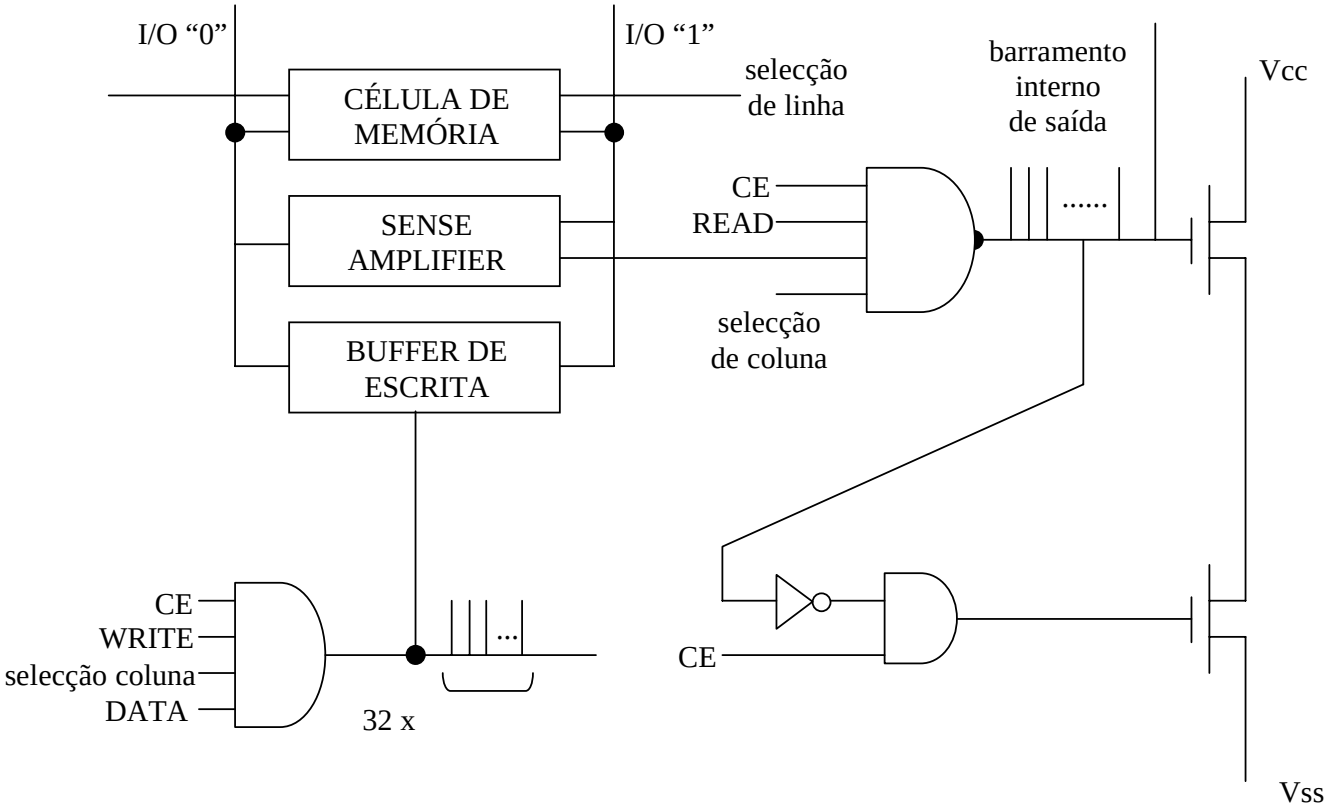



**Fig. 2.3**

### *2.1.2 Operações da memória (Leitura e Escrita)*

Esta memória, além das 10 linhas de endereço A0 - A9 anteriormente descritas e usadas para seleccionar univocamente uma das 1024 células, contém uma linha de entrada de DATA e outra de saída de DATA, duas entradas de controlo e duas entradas para alimentação $V_{cc}$ e $V_{ss}$.

As entradas de controlo são Read/Write (R/W) e Chip Enable (CE).

A linha R/W permite informar a memória se se pretende ler ou escrever nela. A entrada de CE quando no estado lógico “0” selecciona o chip.

Quando CE está no estado “1” a entrada de DATA está desligada electricamente do bus de data interno e o data buffer de saída da memória é colocado no estado de alta impedância. Deste modo o uso de CE permite expandir um sistema de memória.

O “timing” dum ciclo de leitura está ilustrado na Fig. 2.4.

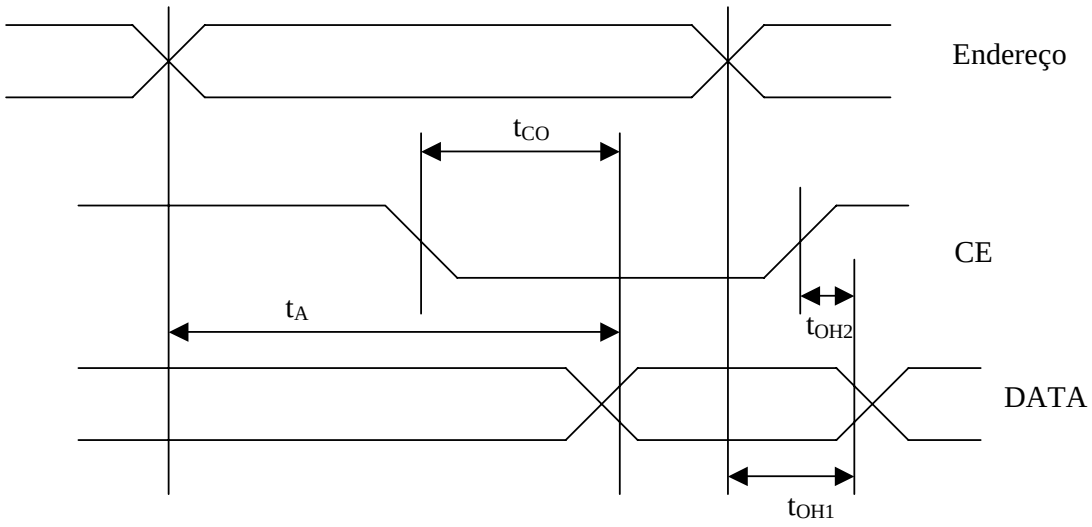



**Fig. 2.4**

Os parâmetros mais importantes são:

$t_A$ – tempo de acesso; é o mínimo tempo, contado a partir do instante em que o desejado valor endereço seja válido, ao fim do qual o DATA saído da memória é válido.

$t_{CO}$ – idêntico ao $t_A$ mas contido a partir do instante em que CE fica activo.

$t_{OH1}$ – tempo depois do endereço ter mudado, durante o qual o data ainda é válido.

$t_{OH2}$ – idêntico ao $t_{OH1}$ mas contado a partir do instante em que CE deixou de estar activo.

Dos valores de $t_A$ e $t_{CO}$ por um lado e de $t_{OH1}$ e $t_{OH2}$ por outro deve ser considerado em cada caso aquele que nas circunstâncias presentes for o mais desfavorável.

Note-se que, embora CE esteja representado como sendo um impulso que ocorre depois da mudança de endereço, não há nenhum tempo específico para o qual ele deverá ocorrer (quer antes quer depois da mudança de endereço).

Assim sendo, é permitido ligar CE a zero, se a saída de data não estiver ligada à saída de data de outras memórias e operar a memória somente com as linhas de endereço e de R/W. Por exemplo, se se pretender fazer uma série de ciclos de leitura e o pino de data de saída não estiver ligado a outros, CE pode ser mantido em zero e os endereços podem ser mudados em qualquer ordem para obter a informação armazenada. Durante este tempo, contudo, deverá manter-se a linha de R/W em “1”. Para este caso o data será válido ao fim de $t_A$.

Um outro método de ler a memória pode ser usado quando o endereço pretendido é conhecido antes de se poder tomar uma decisão de leitura. Quando tal decisão pode ser feita, CE deverá ser posto activo. Data será válido ao fim de $t_{CO}$.

O “timing” dum ciclo de escrita está representado na seguinte figura.

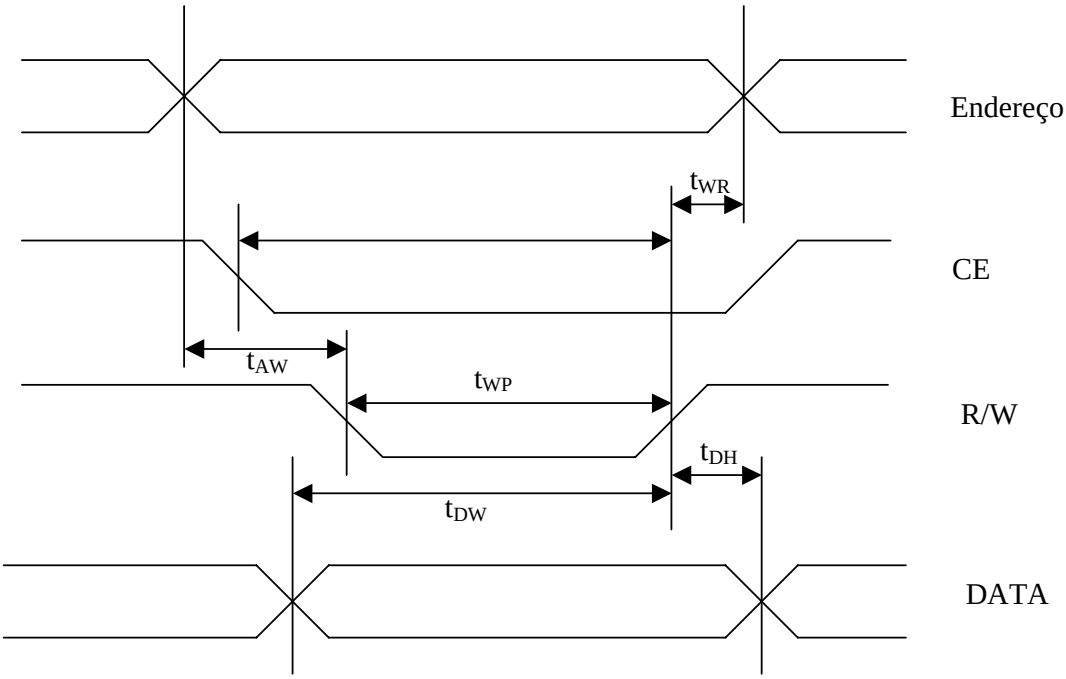



**Fig. 2.5**

Os parâmetros mais importantes são:

$t_{AW}$ – que é mínimo tempo, depois do endereço estar estável, ao fim do qual R/W deve ir ao estado "0".

$t_{WP}$ – tempo mínimo durante o qual o sinal R/W deve estar em "0".

$t_{DW}$ – tempo mínimo durante o qual data deve ser válido para poder ser escrito na memória.

$t_{DH}$ – tempo mínimo durante o qual "data" ainda deve ser válido depois de R/W ter voltado a "1".

$t_{WR}$ – tempo mínimo entre R/W ter voltado a "1" e o endereço deixar de ser válido.

Durante um ciclo de escrita não é permitido efectuar uma série sucessiva de ciclos mantendo CE e R/W em "0" e variar os endereços. Contudo, poder-se-á manter CE em "0" continuamente desde que o sinal de R/W seja variado de acordo com o timing da Fig. 2.5.

Quando se usa uma memória, o ciclo de escrita é o mais crítico, nomeadamente o controlo do sinal R/W, devendo tomar-se todas as precauções para evitar escritas extemporâneas.

### *2.2 RAM's dinâmicas*

A maior parte das considerações feitas anteriormente são válidas para este tipo de memória. A diferença fundamental está no processo de guardar a informação e de a manter.

Os circuitos de descodificação, de selecção de linha e de coluna, assim como a condução da informação da célula de memória para o exterior e deste para aquela são em tudo semelhantes aos das memórias estáticas. Em função disso os diagramas temporais são também semelhantes.

Como já se disse atrás, a diferença essencial entre as memórias dinâmicas e estáticas, reside na estrutura da célula de memória. Nas memórias dinâmicas ela é

constituída basicamente por uma capacidade. A informação guardada depende do estado de carga ou de descarga dessa capacidade.

Uma vez que é bastante fácil implementar essa capacidade parasita, o que noutros casos é mesmo indesejável, é possível realizar matrizes de memória que ocupam menos área e paralelamente mais rápidas, com menos consumo do que as correspondentes estáticas.

Há no entanto um detalhe fundamental: as correntes de fuga dessas capacidades de armazenamento são tais que ao fim de alguns milisegundos há uma diminuição significativa da carga das capacidades carregadas, o que pode levar os circuitos de leitura a interpretar a conteúdo dessas células de memória como sendo o correspondente estado descarregado. Para que tal não aconteça é necessário que periodicamente se efectue o refrescamento dessas células de memória, operação que internamente corresponde ao seguinte: para cada célula de memória, e se ela se apresentar ainda com uma tensão superior a um determinado limite, voltar a carregá-la com o valor máximo possível; se ela se encontrar descarregada não alterar o seu estado.

Na maior parte das memórias dinâmicas esta operação tem de ser realizada com um período máximo de 2ms e, pela maneira como a matriz e os dispositivos de leitura estão organizados, é possível realizar o refrescamento de todas as células de uma linha fazendo a leitura de uma das células dessa linha.

Por exemplo, se a organização da memória for de 4k x 1, a que corresponde uma organização de 64 x 64 (linhas x colunas) e, como regra geral a operação de refrescamento realiza-se linha por linha intervaladas regularmente no tempo, é necessário que em cada 2ms / 64 linhas = 30 μs a actividade normal da memória seja interrompida e se efectue a leitura de uma posição da linha que está a ser "avivada" e cujo endereço é indicado por um contador binário de 6 bits que é incrementado cada vez que esta operação é realizada. Esta operação pode ser realizada através de lógica avulso (contadores, gates e monoestáveis) implementados pelo utilizador de memória ou utilizando um circuito integrado próprio, designado controlador de memória dinâmica.

A Fig. 2.6 apresenta um esquema simplificado de uma célula (1 bit) de memória dinâmica.

Graças à simplicidade da estrutura da célula básica, este tipo de memória tem duas vantagens sobre a memória estática: menor custo por bit – implica que para a mesma capacidade de memória, o custo seja menor – e menor consumo. A sua maior desvantagem é a necessidade do refrescamento, o qual conduz a uma maior complexidade do circuito de controlo do sistema de memória.

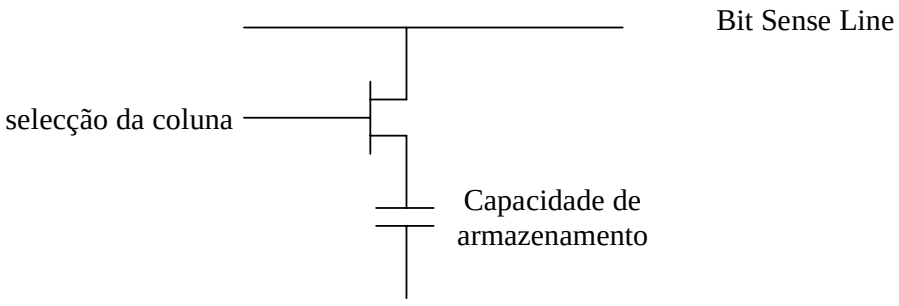



**Fig. 2.6**

Em virtude das suas características, as memórias RAM dinâmicas são essencialmente utilizadas nos sistemas de memória de grande capacidade e/ou quando o consumo deve ser muito reduzido.

### *2.2.1 Endereçamento de Memórias Dinâmicas*

Cada bit de uma memória dinâmica (estas memórias contêm, normalmente, apenas um bit em cada registo da memória) é endereçável individualmente. Assim, por exemplo, uma memória de 64 kbits de informação necessitaria de 16 linhas de endereço para permitir seleccionar qualquer um dos seus bits.

Atendendo à grande capacidade de armazenamento das memórias dinâmicas seriam necessárias muitas linhas de endereço para se efectuar o acesso a qualquer uma das suas células internas. No sentido de reduzir o número de linhas de endereço necessárias, este tipo de memórias têm um processo de endereçamento baseado na multiplexagem temporal dos endereços. Deste modo reduz-se o número de pinos de endereço a metade do que seria necessário. A estes pinos são aplicados, sucessivamente, as duas metades do endereço pretendido, sendo a multiplexagem assim efectuada controlada por 2 linhas auxiliares. A primeira metade do endereço aplicado designa-se por endereço de linha (row address) e a segunda metade por endereço de coluna (column address). Os dois sinais que controlam a multiplexagem são chamados **RAS** (Row Address Strobe) e **CAS** (Column Address Strobe).

O endereço de linha é adquirido pela memória no bordo descendente do sinal **RAS** e o endereço de coluna é adquirido no bordo descendente do sinal **CAS**. A Fig. 2.7 apresenta o esquema desta operação.

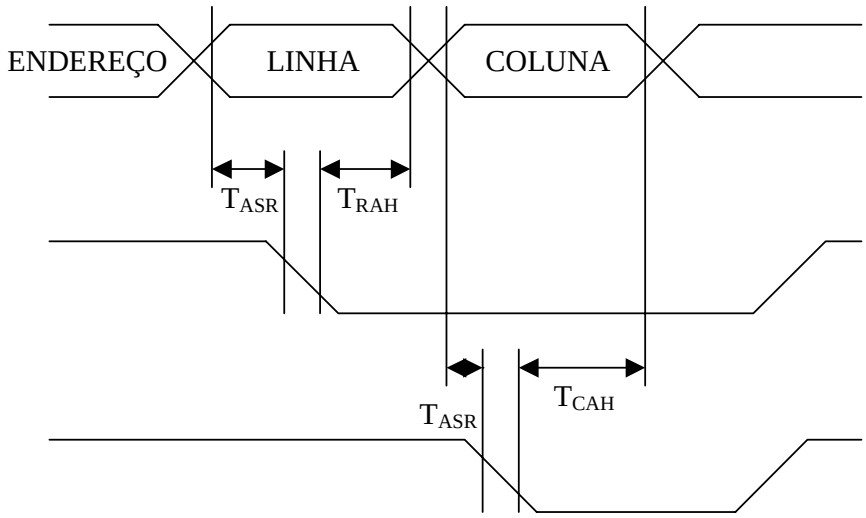



**Fig. 2.7**

Atendendo ao esquema apresentado, uma memória dinâmica de 64 kbits, por exemplo, precisa apenas de 8 pinos de endereço, como mostra a Fig. 2.8.

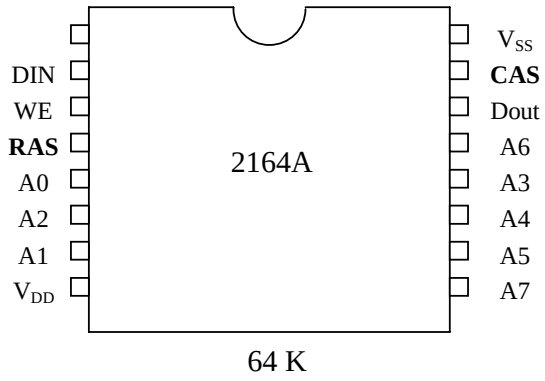



**Fig. 2.8**

Uma memória dinâmica pode ser vista como um “array” bidimensional de células de memória com um bit. No caso da memória de 64 kbits este “array” seria constituído por $2^8$ (256) linhas e$2^8$ (256) colunas (ver Fig. 2.9). Esta organização é a origem das designações “row address” e “column address”. Contudo, a implementação real da memória pode não corresponder rigorosamente a esta organização.

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| $0_H$ | $1_H$ | $2_H$ | $3_H$ | .... | $FC_H$ | $FD_H$ | $FE_H$ | $FF_H$ |
| $100_H$ | $101_H$ | $102_H$ | $103_H$ | .... | $1FC_H$ | $1FD_H$ | $1FE_H$ | $1FF_H$ |
| $200_H$ | $201_H$ | $202_H$ | $203_H$ | .... | $2FC_H$ | $2FD_H$ | $2FE_H$ | $2FF_H$ |
| $300_H$ | $301_H$ | $302_H$ | $303_H$ | .... | $3FC_H$ | $3FD_H$ | $3FE_H$ | $3FF_H$ |
| | | | | .... | | | | |
| ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ |
| | | | | .... | | | | |
| $FE00_H$ | $FE01_H$ | $FE02_H$ | $FE03_H$ | .... | $FEFC_H$ | $FEFD_H$ | $FEFE_H$ | $FEFF_H$ |
| $FF00_H$ | $FF01_H$ | $FF02_H$ | $FF03_H$ | .... | $FFFC_H$ | $FFFD_H$ | $FFFE_H$ | $FFFF_H$ |
| | | | | .... | | | | |

**Fig. 2.9**

### *2.2.2 Ciclos de Memória*

São dois os ciclos de memória mais comuns: leitura e escrita. Contudo, outros ciclos são possíveis nalgumas memórias dinâmicas como é o caso do **"READ-MODIFY-WRITE"**. Para o seu estudo dever-se-á consultar a respectiva *data sheet*.

A distinção entre um ciclo de escrita e de leitura é determinada pela entrada de controlo designada por **WE**. A informação é escrita na memória apenas quando **WE** estiver activo.

A Fig. 2.10 ilustra um ciclo de leitura de uma memória dinâmica. Este é caracterizado pela actuação de **RAS** e **CAS** em conformidade com o conteúdo das linhas de endereço. Durante este ciclo a linha **WE** deve ser mantida no seu estado inactivo.

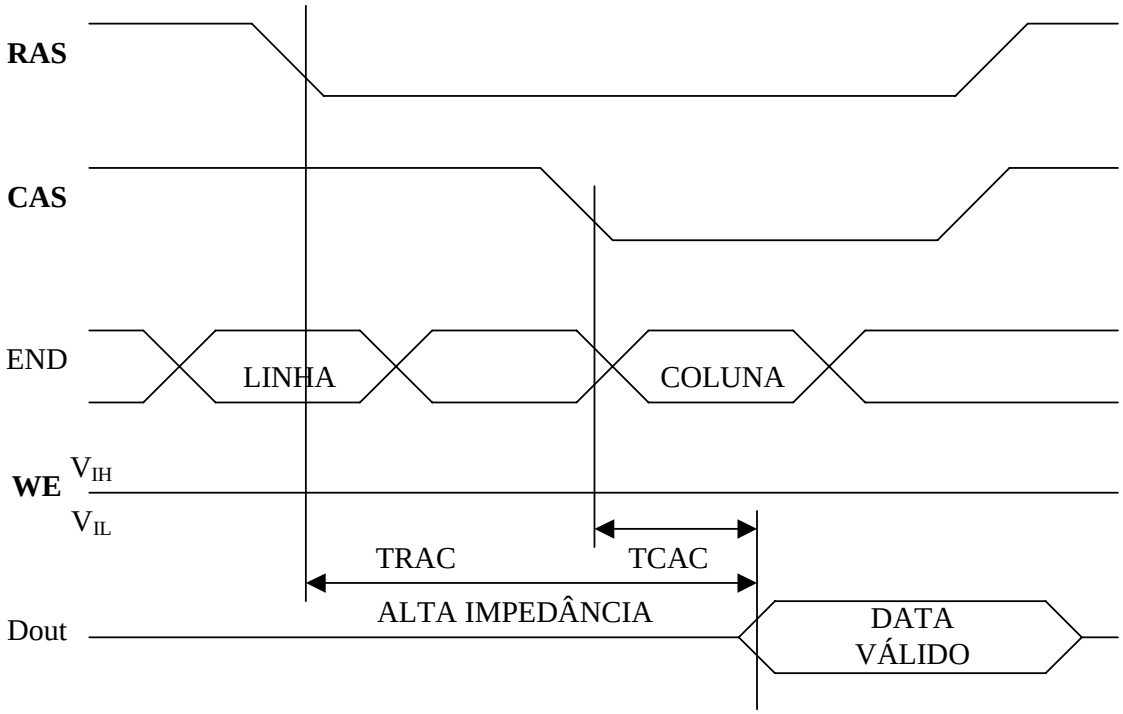
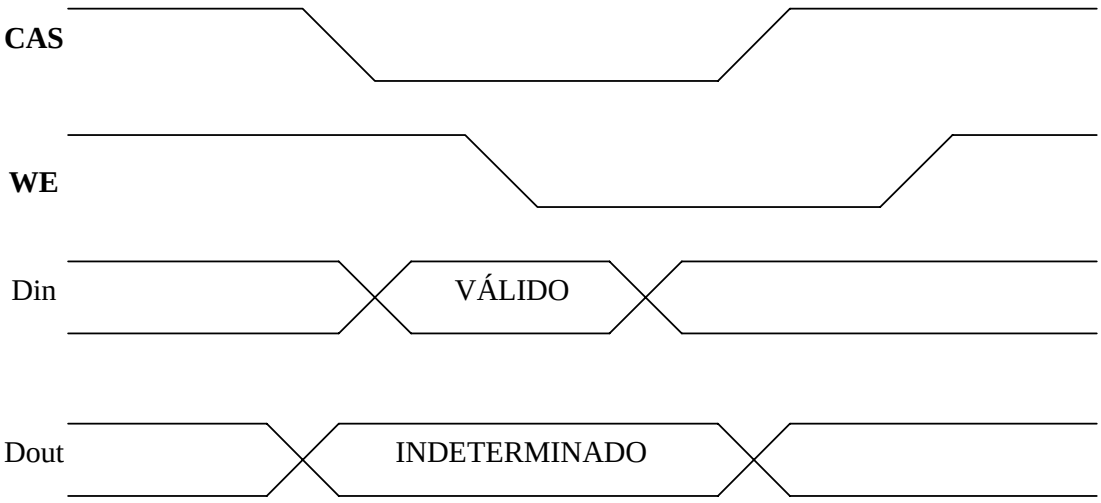



**Fig. 2.10**

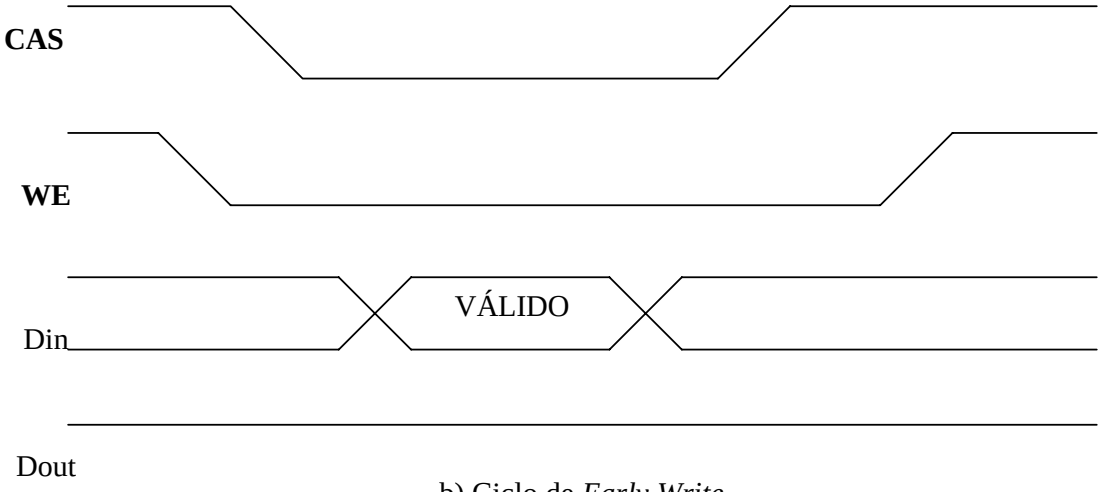
Durante um ciclo de leitura a linha **CAS** tem uma segunda função. De facto, **CAS** faz o *enable* do pino de "Data out" quando estiver activo e admitindo que **RAS** também está activo. Em qualquer outra circunstância a saída "Data out" estará no estado de alta impedância. Isto permite que várias memórias dinâmicas sejam ligadas com o pino "Data out" em comum de modo a permitir a constituição de sistemas de memória de grande capacidade de armazenamento.

Durante o ciclo de escrita a informação é introduzida (*latched*) na memória pelo bordo descendente que ocorrer em último lugar dos sinais **CAS** ou **WE**.

Se **WE** for activo antes de **CAS** ( o caso mais vulgar, designado por leitura temporã (*early write*), a informação é recolhida na memória no bordo descendente de **CAS**. Se **WE** for activado depois de **CAS** (ciclo designado por escrita tardia – *late write*), a informação é armazenada no bordo descendente de **WE**. A Fig. 2.11 apresenta um diagrama dos ciclos de escrita numa memória dinâmica.




a) Ciclo de *Late Write*




b) Ciclo de *Early Write*

**Fig. 2.11**

Os ciclos de *Late Write* são particularmente úteis nos sistemas onde se pretende dar o início ao ciclo de memória o mais cedo possível, para maximizar o desempenho (*performance*), mas o CPU não pode fornecer a informação suficientemente depressa para ser recolhida pelo bordo descendente da **CAS**.

Note-se que quando se executa um ciclo de *Late Write*, o sinal **CAS** é activado enquanto **WE** permanece inactivo; esta situação indica um ciclo de leitura, pelo que a RAM activa a sua saída Dout. Assim, se num sistema se utilizar o ciclo de *Late Write*, os pinos de "data in" e "data out" devem permanecer isolados electricamente. Se não se usar o ciclo de *Late Write* os pinos de "data in" e "data out" podem ser ligados entre si de modo a reduzir o número de pistas no circuito impresso, utilizando-se linhas de dados bidireccionais ainda dentro da placa de memória.

### *2.2.3 Tempos de Acesso*

As RAM's dinâmicas apresentam dois tempos de acesso diferentes:

1) Tempo de acesso a partir de **RAS** activo – $T_{RAC}$

2) Tempo de acesso a partir de **CAS** activo – $T_{CAC}$.

Estes dois tempos de acesso estão ilustrados na Fig. 2.10.
O tempo de acesso a considerar é o que corresponde ao caso mais desfavorável.

### *2.2.4 Refrescamento de memórias dinâmicas*

As memórias dinâmicas necessitam, com se disse anteriormente, de ser refrescadas periodicamente, isto é, efectuar a leitura de cada célula, amplificar a carga do condensador e recarregar este para o seu estado inicial.

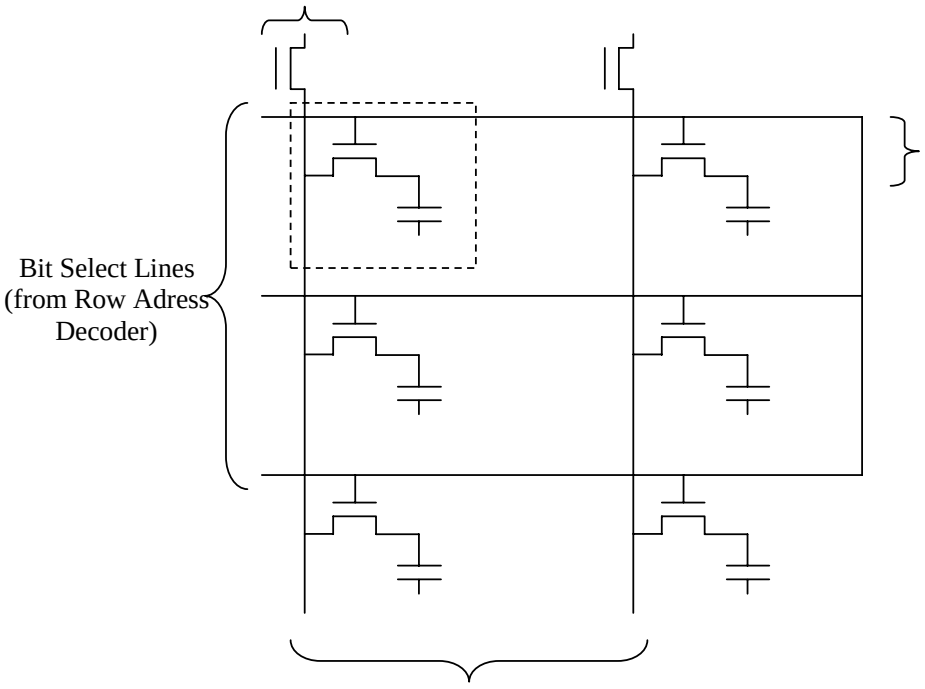



**Fig. 2.12**

O circuito que faz esta amplificação de carga é designado por "sense amp".
O refrescamento de cada célula deve ser executado cada 2 ms, ou menos, para evitar perdas de informação (ver Fig. 2.12).
O processo mais usual de efectuar o refrescamento é designado por **RAS-ONLY**, representado na Fig. 2.13. Durante este ciclo apenas é enviada à memória o endereço de linha. Não é activado o endereço de coluna nem é efectuada qualquer leitura ou escrita de informação.

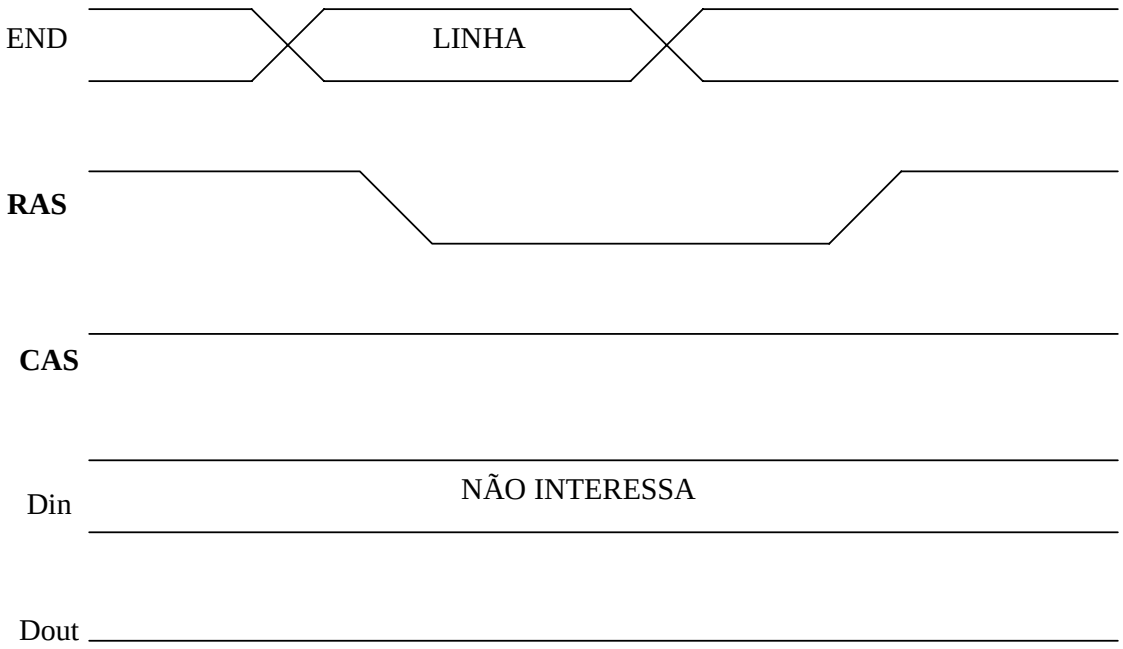



**Fig. 2.13**

Qualquer ciclo de leitura, escrita ou *read-modify-write* efectua o refrescamento da linha endereçada. Este facto poderia ser usado para refrescar a memória, sem necessidade de utilizar ciclos próprios, não fosse o caso de ser impossível assegurar que cada linha seria submetida a um ciclo de leitura ou escrita cada 2 ms.

No que diz respeito ao 'timing', os ciclos de refrescamento podem classificar-se e, três grupos: Refrescamento por "burst"; Refrescamento distribuído e Refrescamento transparente (ver Fig. 2.14).

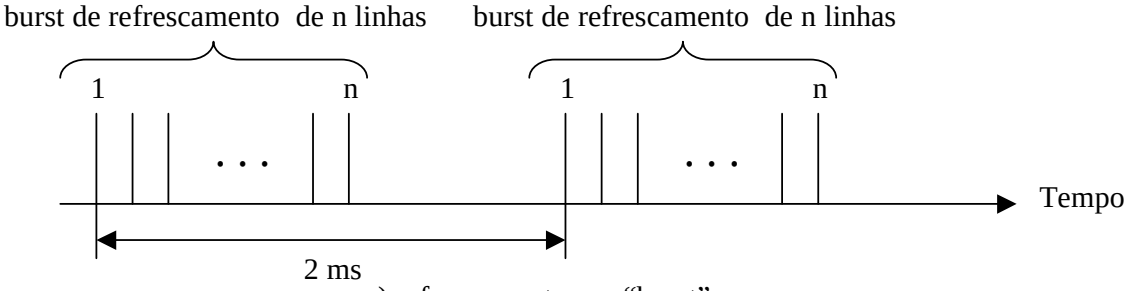
No refrescamento por "burst" o refrescamento de todas as linhas é efectuado sequencialmente e de uma só vez, ao fim de cada 2 ms de intervalo. Este método tem o inconveniente de não ser possível efectuar ciclos de leitura ou escrita da memória durante o período, relativamente longo, do "burst" de refrescamento. Por exemplo, no caso de memória com refrescamento por 128 linhas, a operação de refrescamento tomaria mais de 40 μs cada 2 ms. Esta temporização coloca limitações importantes no tempo de resposta a interrupções, o que a torna impraticável para muitas aplicações.

A técnica de refrescamento distribuído baseia-se no facto de não ser importante o modo como se distribui o refrescamento das diversas linhas dentro do período de refrescamento, desde que se assegure que cada linha é refrescada com a periocidade exigida. Deste modo, nesta técnica efectua-se o refrescamento das diversas linhas com intervalos de 2 ms / n, sendo n o número de linhas a refrescar. Este processo permite garantir a satisfação das exigências da memória ao mesmo tempo que não afecta seriamente a leitura ou a escrita na memória.

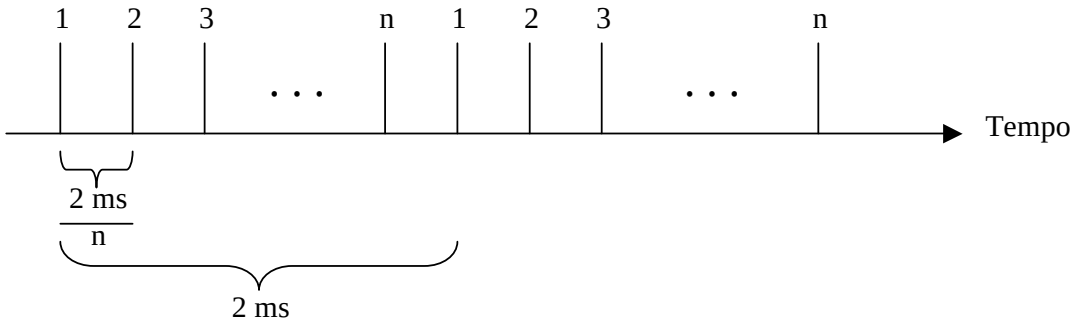
A terceira técnica, refrescamento transparente, aproveita o facto de muitos processadores esperarem um determinado tempo, fixo, depois de extraírem uma instrução e enquanto procedem à sua descodificação. Este tempo pode ser suficiente para efectuar um refrescamento. Se o estado do CPU pode ser examinado para determinar o ciclo de extracção de uma instrução, o refrescamento pode ser efectuado imediatamente após esse ciclo. Deste modo os refrescamentos não interferem com os

ciclos de leitura e escrita, e são "transparentes" para o processador. Contudo, há limitações ao uso do refrescamento transparente, nomeadamente:

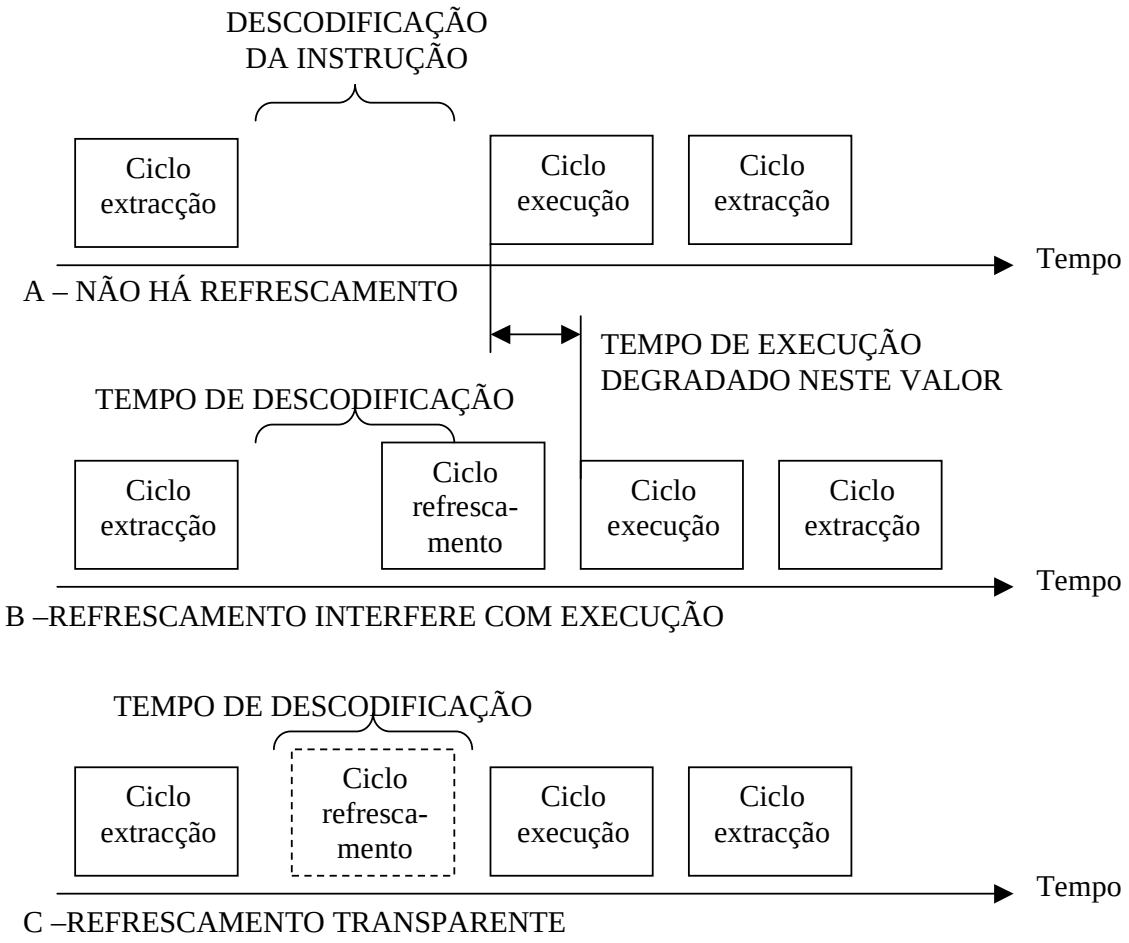
a) No caso do processador deixar de executar ciclos de instruções durante períodos prolongados, como acontece quando se efectua Acesso Directo à Memória por outra entidade exterior (processador em HOLD). Neste caso há o perigo de faltar o refrescamento durante períodos mais longos que o permitido.
b) Processadores que trabalham a muito alta velocidade não deixam tempo suficiente entre o final do ciclo de extracção da instrução e o ciclo de funcionamento seguinte. Neste caso teriam que esperar pelo fim do ciclo de refrescamento antes de começarem o seu funcionamento normal. Estes processadores não terão, pois, qualquer vantagem no uso desta técnica.




a) refrescamento por "burst"




b) refrescamento distribuído




c) refrescamento transparente

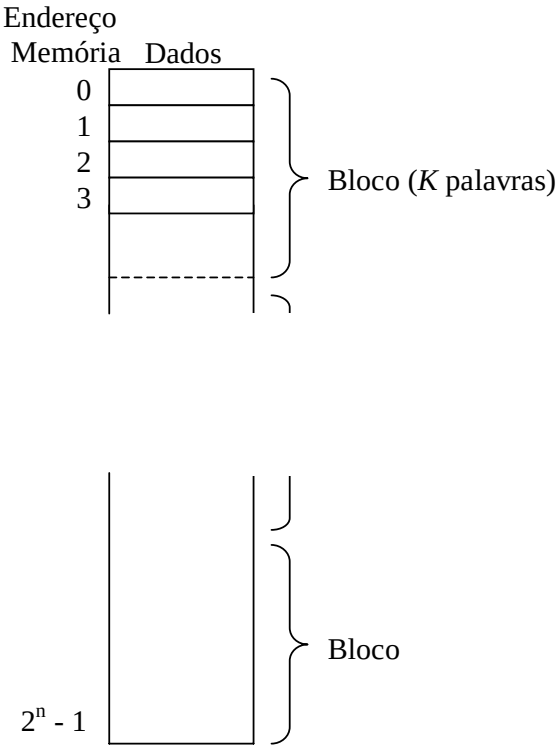
**Fig. 2.14**

### *2.3 Memórias 'CACHE'*

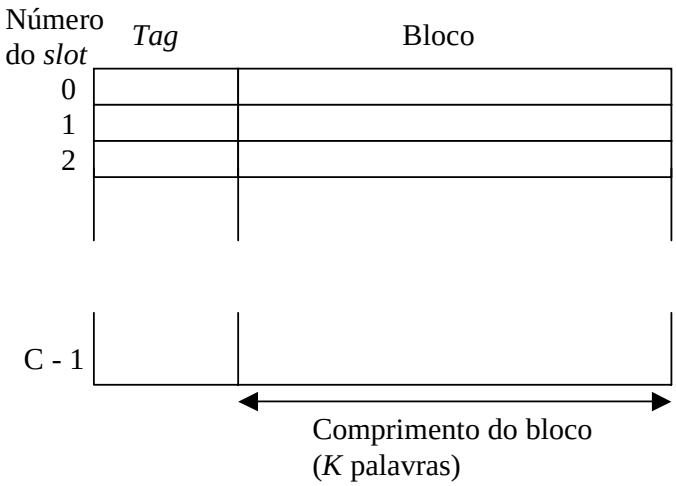
A memória *Cache* (escondida) tem como propósito permitir um rápido acesso do CPU (Central Processing Unit) à memória. A memória *cache* pode ser vista como uma memória intermédia entre o CPU e a memória principal.

A *cache* baseia o seu funcionamento no chamado fenómeno de vizinhança que consiste em considerar que durante a execução de instruções a memória vai utilizar endereços próximos entre si (vizinhos). Assim sendo a *cache* contem cópias de porções da memória principal. Deste modo, quando o CPU tenta ler uma palavra da memória verifica-se se a palavra pretendida está na *cache*; em caso afirmativo a palavra é fornecida ao CPU. Caso contrário um bloco da memória principal é lido primeiramente para a *cache* e posteriormente entregue ao CPU. A cópia efectuada para a *cache* é feita com a esperança que num futuro próximo o processador vai precisar de um endereço próximo do anterior (e por isso contido na *cache*) tornando assim mais rápida a execução do programa. Este procedimento origina taxas de acerto (*hit ratio*) superiores a 90% (i. é, em mais de 90 % das vezes o endereço pretendido pelo processador encontra-se na *cache*).

Na Fig. 2.15 pode ver-se a estrutura de um sistema *cache* / memória principal. A memória principal consiste em $2^n$ palavras endereçáveis, cada uma das quais com um único endereço de *n*-bits. Para efeitos de mapeamento, considera-se que esta memória é constituída por um número de blocos de comprimento fixo de *K* palavras cada. Ou seja, existem $M = 2^n / K$ blocos. A *cache* consiste em *C* parcelas (*slots*) de *K* palavras, e o número de parcelas, ou linhas, é consideravelmente menor que o do bloco da memória principal ($C << M$). Em qualquer altura um subconjunto de blocos de memória reside em parcelas na *cache*. Dado existir mais blocos que parcelas, não se pode dedicar uma parcela individual única e exclusivamente a um bloco em particular. Desse modo, cada parcela inclui uma etiqueta (*tag*) que identifica que bloco em particular é que está a ser armazenado nesse momento. Uma etiqueta é normalmente, uma porção do endereço da memória principal.




a) memória principal




b) *cache*

**Fig. 2.15**

Quanto ao tamanho da *cache* esta tem de ser suficientemente pequena de modo a que o custo médio total (*cache* + memória principal) por bit seja próximo do da memória sozinha e ao mesmo tempo tem de ser suficientemente grande de modo a que o tempo de acesso médio total seja próximo daquele que se obteria considerando só a *cache*. Existem outros factores para que o tamanho da *cache* seja minimizado: é que quanto maior for a *cache* maior o número de portas envolvidas no endereçamento da *cache*, resultando deste facto que as *caches* maiores tenderem a ser mais lentas. Outra condicionante do tamanho da *cache* é a área de chip e placa disponível.

### *2.3.1 Funções de Mapeamento*

Dado existirem menos linhas na *cache* que blocos da memória principal, é necessário um algoritmo para mapear blocos da memória principal em linhas da *cache*. De facto, é necessário um meio de determinar que bloco de memória principal ocupa num dado momento a *cache*. A escolha da função de mapeamento determina a forma como a *cache* é organizada. São utilizadas três técnicas: o mapeamento directo, associativo e set-associativo.

O **mapeamento directo** consiste em mapear cada bloco da memória principal em somente uma linha possível da *cache*. O mapeamento da função é implementada facilmente usando o endereço. Para efeitos de acesso à *cache* cada endereço da memória principal pode ser visto como contendo três campos. Os *w* bits menos significativos identificam uma única palavra ou byte dentro de um bloco da memória principal; os restantes *s* bits especificam um dos $2^s$ blocos da memória principal. A lógica da *cache* interpreta estes *s* bits como uma etiqueta de $s - r$ bits (a porção mais significativa) e um campo de linha de *r* bits. Este último campo identifica uma das $m=2^r$ linhas. Deste modo o uso de uma porção do endereço como número de uma linha fornece um mapeamento único de cada bloco de memória principal na *cache*. Quando um bloco é

lido para sua linha designada, é necessário etiquetar a informação para distingui-lo de outros blocos que possam ser associados com essa linha. Os bits mais significativos *s – r* servem para esta finalidade.

Esta técnica é simples de implementar, só que tem como grande desvantagem a existência de uma localização fixa na *cache* pela qualquer bloco. Se, por exemplo, acontecer que um programa referencie palavras de blocos diferentes mapeados na mesma linha, então os blocos estarão constantemente a alternar na *cache* e a taxa de acerto será baixa.

O **mapeamento associativo** supera a desvantagem do mapeamento directo ao permitir que cada bloco da memória principal seja mapeado em qualquer linha da *cache*. Neste caso o campo de etiqueta só identifica o bloco da memória principal. Para determinar se o bloco está na *cache* o controlador de lógica da *cache* tem de encontrar uma ocorrência da etiqueta em todas as linhas. Como grande desvantagem deste método temos a complexidade exigida em termos de circuitos requeridos para examinar a etiqueta de todas as linhas da *cache* em paralelo.

O **mapeamento set-associativo** é um compromisso que comporta as vantagens das duas técnicas atrás referidas sem as suas desvantagens. Neste caso a *cache* é dividida em *v* conjuntos (*sets*), cada um contendo *k* linhas. Com este mapeamento o bloco $B_j$ pode ser mapeado em qualquer uma das linhas do conjunto *i*. Neste caso a lógica da *cache* interpreta um endereço de memória simplesmente como três campos: etiqueta, conjunto e palavra. Os *d* bits de conjunto especificam um dos $v=2^d$ conjuntos. Os *s* bits dos campos de etiqueta e conjunto especificam um dos $2^s$ blocos de memória principal.

# 3. Interrupções

## *3.1 Introdução*

As interrupções são um mecanismo que permite a interrupção da execução normal de um programa quer por sinais externos – Entradas/Saídas, Memória – ou por instruções (operações) especiais de um dado programa. Em resposta a uma interrupção o microprocessador pára a execução do programa corrente e chama a rotina que "serve" a interrupção. Uma instrução IRET (*Interrupt RETurn*) no fim da rotina de serviço da interrupção devolve a execução ao programa interrompido, que prosseguirá a sua execução a partir do ponto em que foi interrompido. As interrupções permitem ao processador prosseguir com a execução de outras instruções enquanto os periféricos executam a sua operação, só respondendo aos periféricos quando estes estiverem prontos evitando, deste modo, esperas que se podem tornar muito longas, "desperdiçando" tempo ao processador.

As interrupções podem dividir-se em quatro grandes categorias de interrupções:

a) **Programa** – geradas por qualquer condição que ocorra como resultado da execução de uma instrução (exemplos: *overflow* aritmético, divisão por zero, etc.)
b) **Temporizador** – geradas por temporizadores dentro do processador, permitindo ao SO efectuar certas funções regularmente (mais adiante será referido o temporizador programável 8254A)
c) **Entrada / Saída (I/O)** – geradas por um controlador de Entrada/Saída, para sinalizar o término normal de uma operação ou para sinalizar uma variedade de erros
d) **Falhas de Hardware** – geradas por falhas, tais como falha de energia ou erro de paridade da memória.

Na maioria dos processadores existe um pino de entrada de interrupção que é verificada ao fim de cada ciclo de relógio do último ciclo de máquina de cada instrução. Se a entrada de interrupção estiver activa, então o controlo é transferido para uma Rotina de Atendimeto de Interrupção (ISR). Tal está ilustrado na Fig. 3.1.

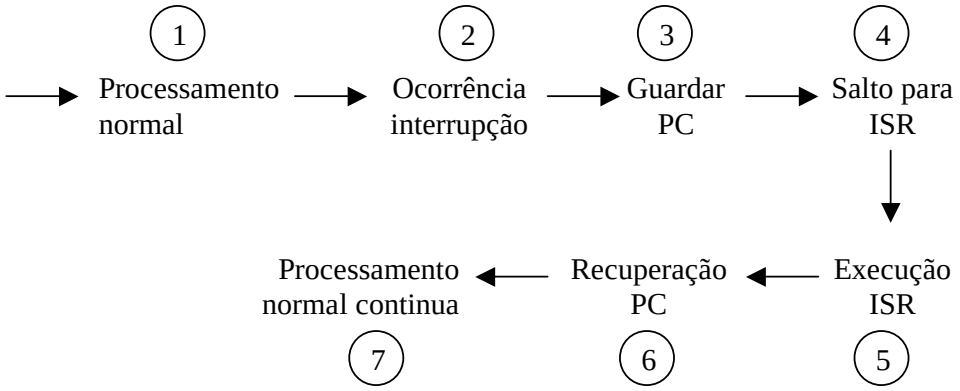



**Fig. 3.1**

De seguida irá ser descrito como funciona um ciclo com interrupção.

Em t=1 está a decorrer o processamento normal de um programa (editar um ficheiro do texto, por exemplo). No instante 2 ocorre uma interrupção. O processador

acaba a instrução corrente (a que está a desempenhar na altura) e guarda o *Program Counter* (PC) na pilha no instante 3. O controlo é então transferido para a ISR no instante 4 e esta rotina é executada durante 5. A ISR acaba com uma instrução de retorno (RET) no instante 6, fazendo com que o PC seja recuperado da pilha. Após ter recuperado o seu endereço antigo, o processamento normal continua em 7.

Quando se estabelecem comunicações com dispositivos de entrada/saída (vulgo periféricos), o uso de interrupções tem a vantagem de permitir ao processador responder somente quando o periférico está pronto, originando uma maior eficiência no uso do microprocessador.

Como exemplo de uma interrupção a seguir descrevem-se os passos que usualmente ocorrem aquando de uma interrupção provocada por um periférico, tendo as interrupções originadas por *software* uma evolução semelhante:

1. É requisitada uma interrupção activando, para tal a respectiva entrada do processador. A finalidade deste pedido é informar o processador que ele deve suspender a sua actividade, processar a informação do que originou a interrupção e finalmente retomar as operações que tinha suspendido.
2. O processador tem à sua escolha aceitar ou não o pedido de interrupção que lhe foi feito (regra geral os processadores possuem uma instrução para desactivar alguns tipos de interrupção – *disable interrupt*). Se aceitar o pedido, acaba, primeiramente, de executar a instrução que estava a ser realizada, assinalando exteriormente a aceitação do pedido, activando um sinal de controlo (normalmente designado por "INTERRUPT ACKNOWLEDGE").
3. O periférico pode usar este sinal IACK (*Interrupt ACKnowledge*) para iniciar a transferência de informação pela porta adequada. O periférico deve também, ao receber IACK, desactivar o pedido de interrupção, uma vez que ao ser servido pelo processador deixa de existir o motivo do pedido. O processador atende o pedido começando por guardar o conteúdo do *Program Counter*, de modo a poder regressar ao programa interrompido depois de ter servido o periférico. Seguidamente determina a endereço da memória onde se inicia o programa a executar em resposta ao pedido de interrupção (Rotina de Interrupção - IRS). Há vários processos de determinar este endereço, os quais serão analisados posteriormente.
4. Como, quando ocorreu o pedido de interrupção, o processador estava a executar um programa, é quase certo que os registos internos de trabalho e as *flags* (acumulador, registos vários, etc.) teriam valores que serão necessários conservar de modo a que mais tarde, ao voltar a esse programa, ele recomece exactamente no ponto em que se encontrava. Há dois modos de alcançar este objectivo. Num deles o pedido de interrupção desencadeia a execução de um microprograma (guardado na unidade de controlo) que guarda o conteúdo de todos os registos de trabalho na pilha (*stack*). No outro processo, a conservação dos valores dos referidos registos é feita por *software* no início da rotina de interrupção. Cada um destes processos tem vantagens e desvantagens. O primeiro liberta o programador de ter que escrever a rotina para guardar os registos poupando, portanto, bytes de memória. Em contrapartida guarda sempre todos os registos na pilha, mesmo que não seja necessário, podendo por isso, atrasar a

resposta a um pedido de interrupção. O segundo processo torna-se mais flexível, permitindo guardar-se apenas o conteúdo dos registos necessários e assim acelera a resposta ao pedido feito. Em contrapartida necessita de memória para guardar as instruções necessárias a esta operação pode tornar-se mais lenta que o anterior se se tiver que guardar todos os registos. Após esta operação o processador procederá à transferência de informação requisitada por meio de *software*.

5. Uma vez terminada a transferência de informação o processador deverá retornar ao programa principal, que tinha sido interrompido, devendo repor o conteúdo original das registos de trabalho, quer por *software* quer por execução de um microprograma disparado pela realização de uma instrução de retorno de interrupção (IRET, por exemplo).

Há, contudo, vários problemas que podem surgir aquando do uso de interrupções e que têm de ser resolvidos. Citemos alguns:

1. É necessário adicionar *hardware* para gerar interrupções dos periféricos.
2. O endereço da ISR tem de ser determinado aquando da altura da interrupção.
3. Existem actividades que não podem ser interrompidas, pois podem originar perda de informação (por exemplo, operações de disco).
4. Se forem utilizados vários periféricos cada um tem de ter a sua própria ISR.
5. O que é que acontece se dois, ou mais, dispositivos requererem interrupções simultaneamente?

Há ainda outro perigo aquando do uso de interrupções de I/O (entrada/saída), não presente em operações programadas: o fluxo do programa agora é controlado por *hardware* em vez de o ser por *software*. E como a fonte da interrupção é assíncrona relativamente ao processador, qualquer programa pode ser interrompido em qualquer altura. Acontece que demasiadas interrupções podem fazer com que o microprocessador se torne assíncrono, fazendo com que o programador perca o controlo da sua máquina.

Os processadores têm um pino de entrada de interrupção (INT) – havendo processadores que têm mais de um pino – que terá de ser activado aquando da ocorrência do pedido de interrupção. Tendo em conta que o processador não irá atender um pedido de interrupção até ao fim da instrução, é necessário ter cuidado para que o processador receba de facto o pedido de interrupção. Na maioria dos casos o processador irá gerar um sinal especial de reconhecimento de interrupção (*interrupt acknowledge*), chamado INTA. Este sinal pode ser usado para remover o pedido de interrupção assim que haja reconhecimento.

### *3.2 Interrupções mascaráveis e não-mascaráveis*

As interrupções podem ser classificadas como mascaráveis ou como não-mascaráveis. Uma interrupção mascarável é uma interrupção que pode ser desactivada pelo CPU. Quando o processador activa o comando de desactivação de interrupções então nenhuma das interrupções mascaráveis pode ser atendida. As interrupções são desactivadas sempre que ocorra um RESET ou após receber o pedido de interrupção. Por seu lado o CPU não pode ignorar uma interrupção não-mascarável; tem de ser

atendida. Este tipo de interrupções tem de ser usado com muito cuidado, estando geralmente reservado para indicação de falha de energia (capítulo 3.5).

## *3.3 Endereçamento das Rotinas de Atendimento de Interrupções*

Quando ocorre uma interrupção, o controlo tem de ser transferido para uma Rotina de Atendimento de Interrupção (*Interrupt Service Rotine* - ISR). Existem três técnicas de endereçamento para isto: Vectorial, Directo e Indirecto. Vamos ver como funciona cada uma destas técnicas.

### *3.3.1 Interrupções Vectoriais*

Quando se usa esta técnica o dispositivo que gera a interrupção fornece um vector indicando o endereço da ISR.

### *3.3.2 Interrupções Directas*

Esta técnica, só suportada em alguns processadores, faz com que quando for aplicado o pedido de interrupção, nos pinos apropriados do processador, o programa execute directamente um salto para uma localização específica da memória, onde existirá uma ISR, sem a necessidade de um vector de interrupção.

### *3.3.3 Interrupções Indirectas*

Também esta técnica não é suportada por todos os processadores. Entre os processadores mais comuns que usam esta técnica encontra-se o Z-80. Nesta técnica, semelhante à vectorial, é usada a combinação do vector fornecido pelo requerente da interrupção com o registo I do Z-80 para formar um apontador de 16 bits para uma tabela de vectores. Os dois bytes consecutivos armazenados nesta tabela são usados como o endereço da ISR.

As Interrupções Indirectas permitem uma maior flexibilidade pois a ISR pode estar em qualquer espaço da memória; as Interrupções Directas são mais fáceis de implementar pois não necessitam de qualquer *hardware* externo; as Interrupções Vectoriais oferecem a possibilidade de oito periféricos poderem partilhar a mesma linha de pedido de interrupção, mas cada um saltar para uma ISR separada; contudo, cada rotina está limitada a oito bytes na página 0.

## *3.4 Interrupções Múltiplas*

Já foi mencionado que uma ISR pode interromper outra. Isto só pode ocorrer em dois casos especiais: a primeira ISR executa uma instrução de EI (*Enable Interrupt*) anteriormente na rotina ou quando a segunda interrupção é não-mascarável. Recorde-se que quando uma interrupção é atendida as interrupções mascaráveis são automaticamente desactivadas. Isto é feito para proteger a ISR de ser interrompida antes de ter uma oportunidade de executar a sua tarefa. Contudo uma interrupção não-mascarável nunca pode ser bloqueada, mesmo se for executada a instrução DI (*Disable Interrupt*).

Podem se usar interrupções em ninho (*nested*) em que ocorrem interrupções dentro de interrupções. Neste caso é preciso ter muito cuidado de modo a que a área da pilha não cresça de modo a escrever por cima do programa ou nem a ocupar qualquer uma das suas tabelas de dados.

Outro problema que pode ocorrer é a possibilidade de duas interrupções chegarem simultaneamente. Dado o facto de o processador só testar a existência de pedidos de interrupções no fim da instrução corrente, não é necessário que dois, ou mais, pedidos de interrupção cheguem no mesmo instante para se apresentarem ao processador como simultâneos. Este problema tem uma solução fácil e uma solução difícil. A solução fácil é fornecida pelo próprio processador: este dá prioridades diferentes às interrupções. Obviamente que uma interrupção não-mascarável tem uma prioridade maior, mas esta atribuição de prioridades refere-se somente às interrupções pendentes, não à ISR em curso. Por exemplo, uma interrupção pode interromper uma ISR; quando tal não é desejado cabe ao programador assegurar-se que não são activadas quaisquer interrupções até que a ISR termine a sua tarefa. Deste modo só as interrupções não-mascaráveis podem interromper a ISR em execução.

Dar prioridades às interrupções funciona bem com interrupções directas. Contudo deve ser usada outra técnica quando uma linha de interrupção é partilhada por vários periféricos. Este é tipicamente o caso que ocorre quando é usada a linha de interrupção vectorial. No caso de vários periféricos estarem ligados ao pino de pedido de interrupção ocorre um problema para o processador que é determinar qual dos periféricos efectuou o pedido e resolver quaisquer pedidos simultâneos de dois ou mais dispositivos. Geralmente são usadas três técnicas.

1. Consulta (*Polling*): o sinal de pedido de interrupção de qualquer dispositivo pode ser usado para determinar um bit de um registador conectado como uma porta de entrada. Quando ocorre uma interrupção a ISR consulta esta entrada para "ver" quem fez a requisição do serviço. Então, é estabelecida, automaticamente, uma prioridade pela ordem de consulta. Esta técnica é muito simples, mas tem a desvantagem de ter um tempo de resposta degradado.
2. Controlador de Prioridades de Interrupções (*PIC*): se estiverem presentes duas, ou mais, interrupções será atendida a que tiver maior prioridade. Esta técnica será estudada com mais detalhe no capítulo 3.7 ao se estudar o PIC 8259.
3. Corrente *Daisy*: cada periférico tem uma entrada de activação de interrupção (IEI) e uma saída de activação de interrupção (IEO). Um pedido de interrupção só pode ser feito se IEI for 1. Atendendo à Fig. 3.2 vê-se que o periférico 2 está a requerer uma interrupção fazendo com que a sua saída seja 0. Tal vai fazer com que os periféricos 3 e 4 fiquem desactivados. Contudo o periférico 1 vai ficar com a possibilidade de efectuar um pedido de interrupção pois tem uma maior prioridade.

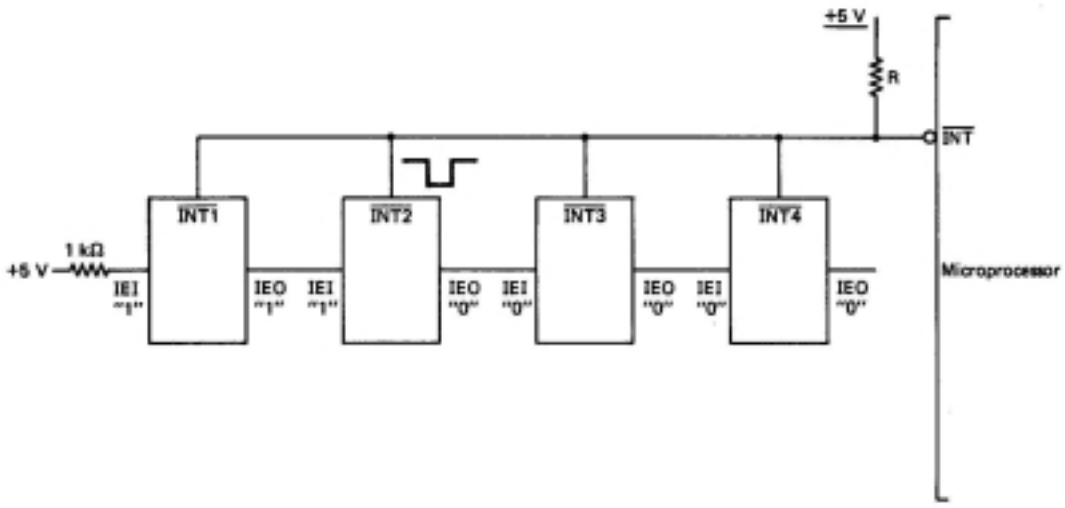



**Fig. 3.2**

### *3.5 Interrupções por falha de energia*

Em muitos sistemas de microprocessadores a interrupção de maior prioridade é usada para detectar e actuar convenientemente uma falha de alimentação.

As fontes de alimentação usadas pelos microprocessadores são normalmente +5 e/ou +12V, as quais são derivadas da alimentação de 220V. A falha de alimentação pode ser detectada, por exemplo, quando a tensão desce abaixo dos 180V. Contudo poderão ocorrer alguns milisegundos até que as fontes de 5 ou 12V apresentem valores suficientemente baixos para impedir o correcto funcionamento do microcomputador. Durante estes milisegundos podem se executar algumas centenas de instruções para se preparar para a falha de alimentação de uma maneira ordenada. Quando a alimentação regressar o programa interrompido pode recomeçar sem perda de informação. É claro que se supõe a existência de memória não volátil (por exemplo RAM com alimentação a pilhas) para guardar o conteúdo dos registos durante a falha de energia.

### *3.6 Temporizador Programável 8254*

Existem problemas relacionados com a precisão de intervalos de tempo para gerar sinais do tipo onda quadrada e pulsos. Inclusive em alguns casos os microprocessadores são chamados a contar eventos e a tomar medidas após um determinado número de pulsos. Como exemplo temos o caso de um relógio em tempo real. É para aplicações deste género que o Temporizador Programável 8254 foi concebido.

O 8254 fornece três registos temporais separados de 16 bits cada, podendo cada um dos registos ser programado como contador ou como temporizador. Por exemplo, se se carregar o contador 0 com o valor $868_{10}$ e especificando-se o modo 3 gera-se uma onda quadrada com 50% de *duty-cycle* do tipo divisão-por-868.

A programação do 8254 consiste em escrever um byte na porta de controlo seleccionando um dos seis modos de operação possíveis.

Cada contador suporta um número de 16 bits em contagem decrescente até 0 a uma taxa determinada pela entrada do registo. Quando ocorre a contagem terminal pode se gerar uma interrupção, um impulso estroboscópico (*strobe*), terminar um impulso *one-shot* ou inverter o valor lógico de uma onda quadrada.

Existe um dispositivo, o 8253, que é uma versão "desactualizada" do 8254, sendo as principais diferenças o facto de o 8253 não ter modo de *read-back* além de estar limitado a uma frequência máxima de 2 MHz.

### *3.6.1 Interface do 8254*

O diagrama de blocos do 8254 está representado na Fig. 3.3.

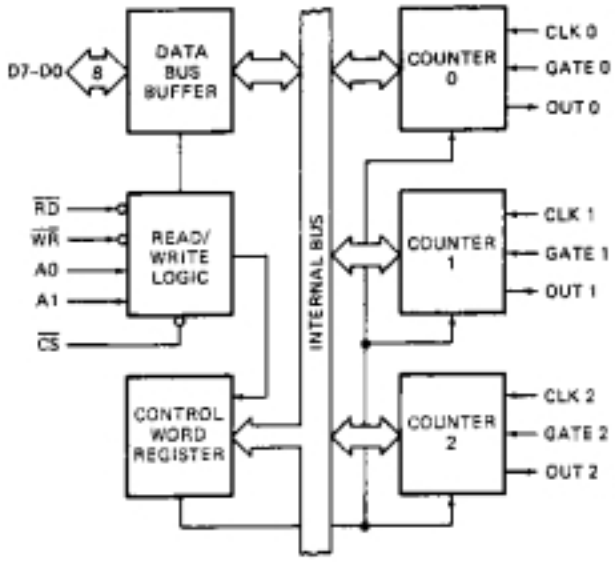



**Fig. 3.3**

O 8254 é constituído por 3 contadores de 16 bits cada. No lado esquerdo do diagrama de blocos do 8254 (fig. 3.3 da sebenta) pode ver-se que existem 8 linhas de dados para comunicação com o exterior (D7 - D0). Estas linhas permitem escrever ou ler comandos e números no 8254; isto é determinado pelas entradas WR e RD. A entrada CS é necessária para que o dispositivo seja seleccionado. As entradas A0 e A1 servem para determinar qual o contador seleccionado. Do lado direito do diagrama de blocos vê-se os três contadores com as suas entradas de CLK e GATE, bem como com a sua saída OUT. A entrada CLK permite aplicar um sinal com qualquer frequência desde 0 (sinal dc) até 8 MHz. A entrada GATE permite activar ou interromper esse contador através de sinais externos. A saída do contador aparece no seu pino OUT respectivo.

Como se pode verificar, para o processador existem quatro portas de Entrada/Saída seleccionadas pelas linhas de endereço A0 e A1. As entradas RD e WR determinam se os dados são para serem lidos ou escritos usando o barramento bidireccional de dados (Data Bus - D0 a D7). O pino CS tem de ser 0 para que se possa efectuar qualquer operação. O conjunto de operações possíveis pode ser visto na Tabela 3.1.

| CS | RD | WR | A1 | A0 | |
|---|---|---|---|---|---|
| 0 | 1 | 0 | 0 | 0 | Escreve no Contador 0 |
| 0 | 1 | 0 | 0 | 1 | Escreve no Contador 1 |
| 0 | 1 | 0 | 1 | 0 | Escreve no Contador 2 |
| 0 | 1 | 0 | 1 | 1 | Escreve Palavra de Controlo |
| 0 | 0 | 1 | 0 | 0 | Lê do Contador 0 |
| 0 | 0 | 1 | 0 | 1 | Lê do Contador 1 |
| 0 | 0 | 1 | 1 | 0 | Lê do Contador 2 |
| 0 | 0 | 1 | 1 | 1 | Nenhuma Operação |
| 1 | X | X | X | X | Nenhuma Operação |
| 0 | 1 | 1 | X | X | Nenhuma Operação |

**Tabela 3.1**

O 8254 é programado escrevendo-se um só byte para cada contador na porta de controlo seguido de um ou dois bytes para especificar o valor da contagem inicial.

A palavra de controlo do 8254 tem o seguinte formato:

| D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
|---|---|---|---|---|---|---|---|
| SC1 | SC0 | RW1 | RW0 | M2 | M1 | M0 | BCD |

Em que os bits têm o seguinte significado:

| SC1 | SC0 | |
|---|---|---|
| 0 | 0 | selecção do contador 0 |
| 0 | 1 | selecção do contador 1 |
| 1 | 0 | selecção do contador 2 |
| 1 | 1 | comando *read-back* |

| RW1 | RW0 | |
|---|---|---|
| 0 | 0 | Comando *latch* do contador |
| 0 | 1 | R/W só do LSB |
| 1 | 0 | R/W só do MSB |
| 1 | 1 | R/W do LSB primeiro e do MSB depois |

| M2 | M1 | M0 | |
|---|---|---|---|
| 0 | 0 | 0 | modo 0 |
| 0 | 0 | 1 | modo 1 |
| X | 1 | 0 | modo 2 |
| X | 1 | 1 | modo 3 |
| 1 | 0 | 0 | modo 4 |
| 1 | 0 | 1 | modo 5 |

A palavra de controlo pode dividir-se em três tipos:

1. Standard: especifica o modo de operação e o contador seleccionado. Cada contador pode ter um modo de operação diferente e ser programado para operar como contador BCD ou binário.

2. *Latch*: fixa o valor do contador especificado por D6 e D7 (importante por causa dos erros). Neste caso a contagem continua, sendo o valor fixado até ser lido.
3. *Read-back*: permite fazer a fixação (*latch*) de qualquer um ou de todos os contadores ou então fixar uma palavra de estado espacial.

### *3.6.2 Modos de operação do 8254*

**Para todos os modos:**

Inicialmente escreve-se a palavra de controlo (*control word - CW*) que é feito quando a linha WR vai primeiramente a 0. De seguida a linha WR vai novamente a 0 uma ou duas vezes, consoante se esteja a inicializar com um valor de 8 ou 16 bits, respectivamente, para que seja escrito o valor inicial da contagem (1 ou 2 bytes); esse valor só passa para o registo da palavra de controlo após WR ser 1. Passado um ciclo de relógio é iniciada a contagem em ordem decrescente aquando da transição descendente do sinal do relógio (*clock*).

**Modo 0: contador de eventos**

Neste modo a contagem só é efectuada com a GATE a 1. Quando é escrita a CW a saída (OUT) vai a 0. De seguida, sempre que ocorrer uma transição descendente do sinal de relógio o valor da contagem é decrementado uma unidade. Quando a contagem atingir o valo numérico 0, a saída será 1. Se o valor inicial da contagem for N, a saída só voltará a 1 quando decorrerem N+1 impulsos de relógio. Isto pode ser observado na parte superior da fig. 3.4.

Na parte do meio da fig. 3.4, pode ver-se que GATE foi a 0 durante a contagem; neste caso o valor da contagem é retido. Quando GATE voltar a 1 a contagem continua normalmente.

Na parte inferior da fig. 3.4 mostra que se for escrito um novo valor par um contador, ele será carregado no respectivo contador um ciclo de relógio depois. A nova contagem irá então iniciar-se.

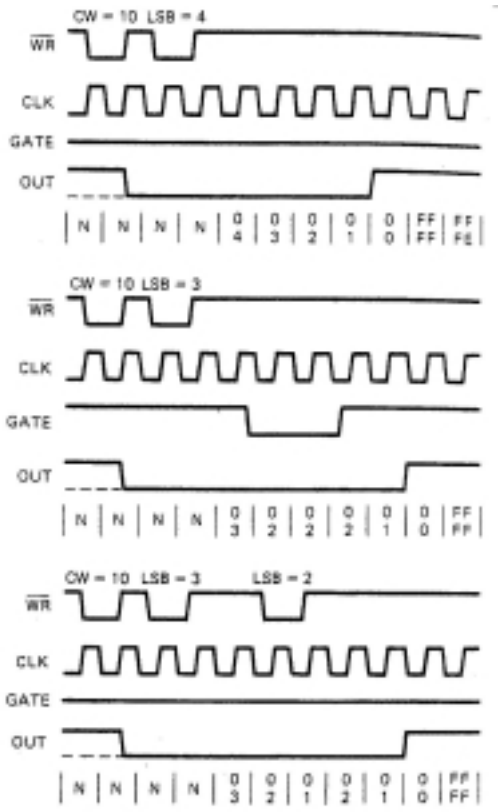



**Fig. 3-4**

**Modo 1: one-shot**

O princípio básico deste modo é que quando for aplicado um sinal à entrada *trigger* de um dispositivo, a sua saída irá ser de acordo com um valor pré-estabelecido (e definido). Após um determinado intervalo de tempo, a sua saída voltará a um estado não definido.

Vejamos alguns exemplos de funcionamento neste modo. Na parte superior da fig. 3.5 podemos ver que após ser escrito o valor inicial de contagem (2ª vez em que WR está a 0), e como o sinal de GATE está a 0, a contagem não começa imediatamente como sucedia no modo 0. Neste modo, a entrada GATE funciona como um *trigger*. Quando esta entrada é 1, o valor de contagem será transferido do registo para o contador respectivo no impulso de relógio seguinte. Só então é que é iniciada a contagem decrementando o valor da contagem em 1 em cada ciclo de relógio. Quando a contagem atingir o valor numérico 0, saída OUT, durante a contagem a 0, será 1. Por outras palavras, se for carregado o valor N no contador e se for feito o *trigger* do dispositivo fazendo GATE igual a 0, a saída OUT será 0 durante N períodos de relógio. O comprimento deste sinal é de N vezes o período do sinal de relógio.

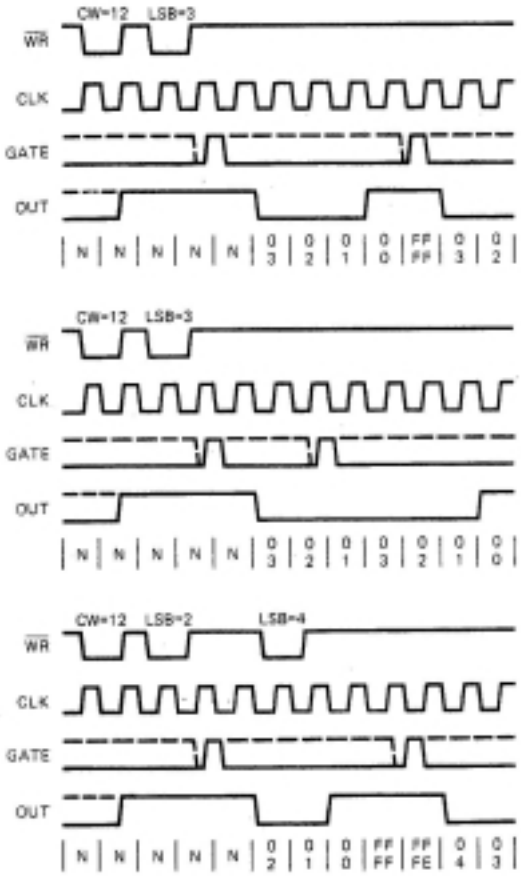



**Fig. 3.5**

A parte do meio da fig. 3.5 demonstra o significado de *retriggerável*. Se ocorrer outro impulso de *trigger* durante a contagem, então esta será reiniciada.

A parte inferior da fig. 3.5 pode ser observado o efeito de se carregar um novo valor de contagem antes de ocorrer a contagem terminal; neste caso esse valor não será carregado no contador nem decrementado até que ocorra outro impulso de *trigger*.

**Modo 2: contador de divisão-por-N (ou gerador de interrupção temporizado)**

Observando a parte superior da fig. 3.6, com GATE sempre igual a 1, pode ser observado que após começar a contagem no contador, a saída OUT do contador será 1 durante a contagem até se atingir o valor numérico 1; então, a saída será 0 durante um período do relógio. No período seguinte a contagem recomeça e a saída OUT volta a 1. Se o contador for carregado com o valor N, a saída OUT do contador irá a 0 durante um período do relógio de N em N períodos do relógio. A frequência do sinal de saída será, então, igual à frequência do sinal de relógio de entrada a dividir por N.

Na parte do meio da fig. 3.6, pode se ver que se GATE for a 0 durante a contagem, então a contagem parará. Quando GATE voltar a 1 o valor inicial da contagem é recolocado no contador.

A última parte da fig. 3.6, mostra o que ocorre quando se escreve um novo valor enquanto está a ser feita a contagem; neste caso, esse novo valor não será transferido enquanto a contagem não atingir o valor numérico 1.

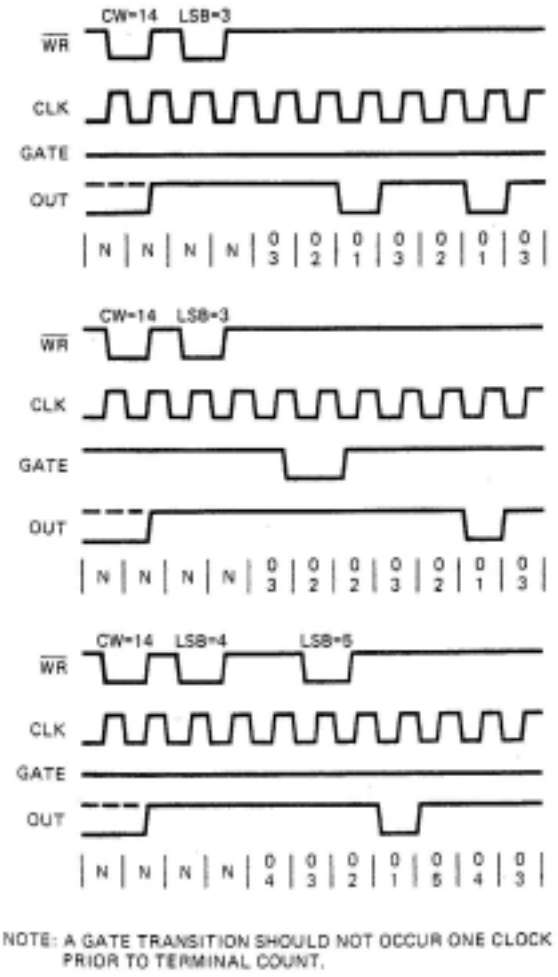



**Fig. 3.6**

**Modo 3: gerador de onda quadrada**

Se o 8254 for programado neste modo e se for carregado um valor inicial par, a onda na saída OUT será uma onda quadrada. A frequência desta onda quadrada será igual à frequência do sinal de relógio a dividir pelo número escrito no contador. Caso o número escrito seja ímpar, e o 8254 for programado neste modo, a onda de saída será 1 durante mais um período do relógio do que 0, resultando, assim uma onda que não será simétrica.

Olhando para a parte superior da fig. 3.7, vê-se que um período após se escrever um valor no contador, o contador irá decrementar duas unidades ao valor da contagem. Neste caso quando a contagem atinge o valor numérico 2, a saída OUT vai a 0 e o valor inicial é recarregado. A saída mantém-se 0 enquanto a contagem é novamente decrementada em duas unidades, até se atingir o valor numérico 2, em que a saída volta a 1 e o valor inicial é recolocado no contador e a contagem é reiniciada. Então, o ciclo repete-se.

A parte do meio da fig. 3.7, mostra o que sucede se for escrito um número ímpar no contador. Como se pode ver o número de períodos de relógio de cada onda mantém-se igual ao número colocado no contador. Contudo, como anteriormente mencionado, a onda de saída está a 1 mais um período de relógio do que está a 0.

A parte inferior da fig. 3.7 mostra que a contagem é interrompida se GATE for 0. Após voltar a 1 a contagem continua.

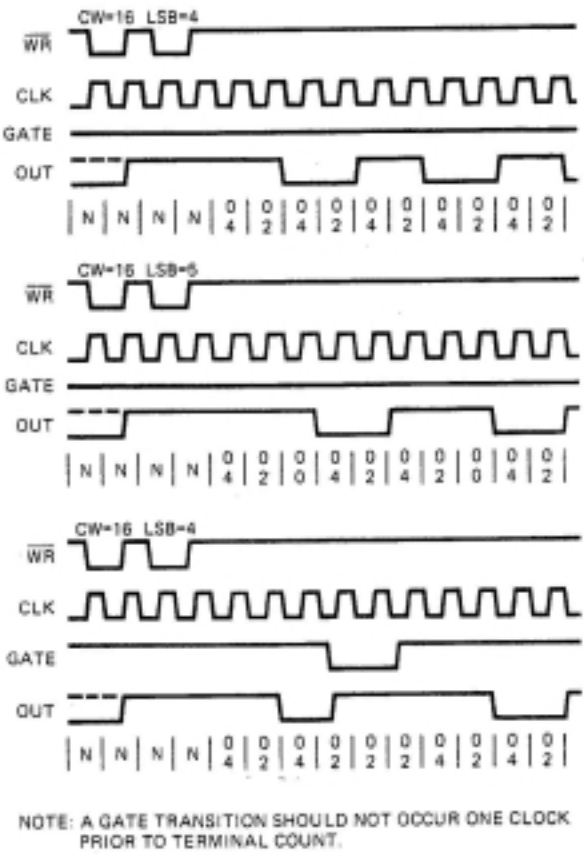



**Fig. 3.7**

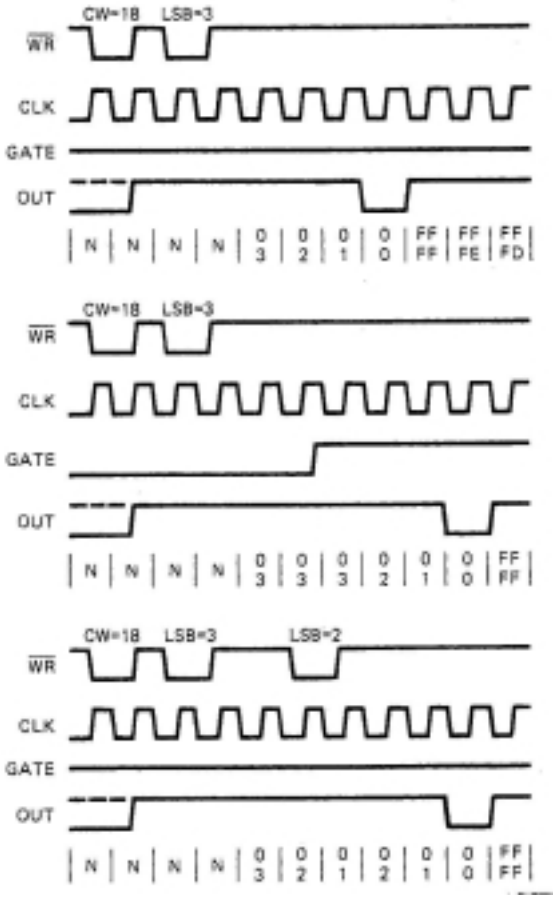



**Fig. 3.8**

### Modo 4: Software-triggered strobe

Este modo e o modo 5, são muitas vezes confundidos com o modo 1, mas há uma diferença óbvia: o modo 1 é usado para produzir um pulso com valor lógico 0 com comprimento de N períodos de relógio. Olhando para a parte superior da fig. 3.8, vê-se que o modo 4 produz um pulso com valor lógico 0 após N+1 períodos de relógio. Neste modo a saída é 0 durante um período de relógio e depois volta a 1. Por outras palavras, o modo 4 produz um impulso de um período de relógio N+1 períodos após o valor de contagem ser escrito no registo. Este modo é denominado de *sotware-triggered*, porque é a escrita do valor da contagem no registo que inicia o processo. É de notar que após terminada a contagem, o contador começa a decrementar desde o valor FFFFh.

Este modo pode ser usado quando se pretende enviar dados paralelos para uma porta e depois de um determinado tempo de atraso enviar um sinal estroboscópico para que o sistema receptor saiba que os dados estão disponíveis.

**Modo 5: Hardware-triggered strobe**

Este modo é utilizado quando se pretende produzir um sinal estroboscópico com valor lógico 0 um intervalo de tempo, programável, depois de um sinal ascendente ser aplicado à entrada GATE. Este modo é muito útil quando se pretende atrasar a subida de um sinal um determinado intervalo de tempo.

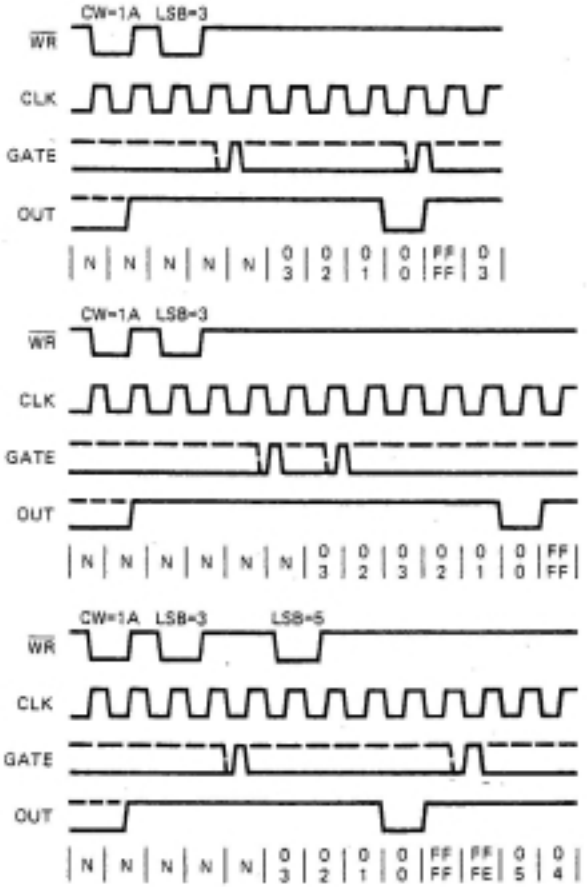



**Fig. 3.9**

A fig. 3.9 mostra exemplos de ondas quando se está a operar no modo 6. Na parte superior desta figura, pode observar-se que o valor da contagem não é transferido para o contador até que GATE vá a 1. Quando isso ocorre o valor de contagem será transferido para o contador um período de relógio depois, seguindo-se um decremento do valor da contagem nos períodos de relógio seguintes. Quando a contagem atinge o valor numérico 0, a saída OUT será 0 durante um período de relógio. A saída OUT irá a 0 N+1 períodos de relógio depois de a entrada GATE ir a 1.

A parte do meio da fig. 3.9 mostra que se ocorrer outro sinal de *trigger* ocorrer durante a contagem, o valor da contagem será recarregado e a contagem recomeçará. Se outro *trigger* ocorrer durante esta nova contagem OUT manter-se-á a 1, podendo, assim, utilizar-se este modo para produzir um sinal da falha de energia.

Por fim, na parte inferior da fig. 3.9 pode ver-se que se for escrito um novo valor de contagem para um contador, este não será carregado até se atingir o fim da contagem anterior.

## *3.7 Controlador de Interrupções Programável 8259A*

Consideremos os processadores 8080, 8085 e Z-80. No primeiro caso temos uma só linha de interrupções (INTR), enquanto que o 8085 contem cinco linhas de interrupção e duas no caso do Z-80. Contudo, na generalidade dos casos o processador lida com vários periféricos sendo necessário a cada processador lidar com vários periféricos geradores de interrupções. A solução é expandir o conjunto do *chip* de modo a incluir um PIC (Controlador de Interrupções Programável). Este é o caso do próximo objecto de estudo: o 8259A.

O 8259A aceita interrupções vectoriais, podendo no modo 8080/8085:

- aceitar oito pedidos de interrupções separados (podendo atingir um máximo de 64 se em cascata);

- determinar e colocar os três bytes necessários para uma instrução CALL no Data Bus do sistema em resposta ao INTA;
- determinar prioridades e arbitrar interrupções simultâneas;
- programar as interrupções para um *trigger* de nível ou de flanco:
- programar o vector de endereço e um intervalo de endereço de 4 ou 8 bytes.
-

### *3.7.1 Descrição do 8259A*

O diagrama de blocos do 8259A encontra-se representado na Fig. 3.10.

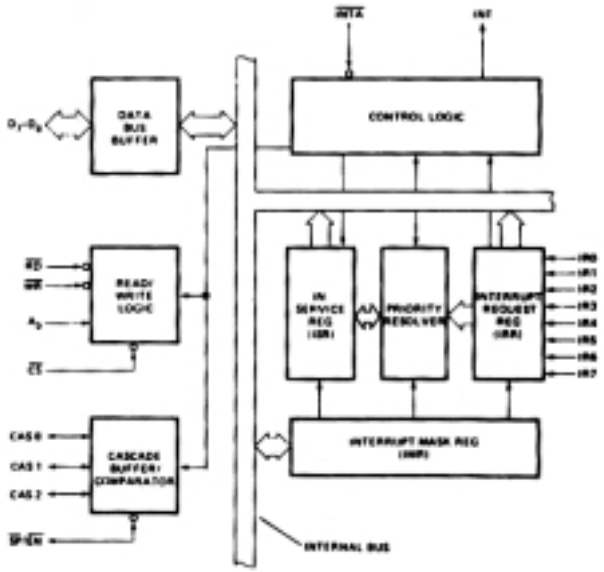



**Fig. 3.10**

O 8259A permite expandir o número de dispositivos requerentes de interrupções que por defeito é baixo nos processadores (1, 2 ou 5) para 8 no modo simples até 64 se em cascata.

Observando o diagrama de blocos do 8259A podemos observar o seguinte:

- as linhas bi-direccionais D7-D0 servem para escrever palavras de controlo no 8259A ou para ler uma palavra de estado. Isto é feito através de WR e RD quando CS está activo
- IR7-IR0 são as entradas que permitem activar o pino INT do 8259A quando uma interrupção é pedida (uma, ou mais, das entradas IR0 a IR7 é activada). A saída INT serve para avisar o processador que existe um pedido de interrupção a ser feito
- a entrada INTA é ligada à saída INTA do processador, serve para o processador notificar o 8259A que já tem conhecimento do pedido de interrupção
- o IMR (registo das máscaras de interrupções) é usado para activar (mascarar - *mask*) ou desactivar (desmascarar - *unmask*) entradas individuais de interrupções. Cada bit deste registo corresponde à entrada de interrupção com o mesmo número. Activa-se uma entrada de interrupção enviando uma

palavra de comando com um 0 na posição do bit correspondente a essa entrada

- o IRR (registo dos pedidos de interrupção) regista quais as interrupções que estão a pedir atendimento. Se uma entrada tiver um sinal activo, então o bit correspondente neste registo será activado. É de notar que o sinal de pedido de interrupção terá de estar activo até ao flanco descendente de INTA
- o ISR (registo em-serviço) regista quais as interrupções que estão a ser atendidas. Cada interrupção em atendimento tem o respectivo bit activado.
- O Priority Resolver ("solucionador" de prioridades) actua como um juiz que determina se e quando um pedido de interrupção nas entradas IR deve ser atendido. Só entra em acção se ocorrer um pedido de interrupção com prioridade maior à que estiver presentemente a ser atendida; caso contrário não actua.

### *3.7.2 Operação em modo cascata*

Ligações:

O INTA vai do processador para o Mestre e para todos os Escravos; o INT do Escravo vai ser ligado a uma das entradas IR do Mestre; o INT deste é que liga à entrada INT do processador.

As linhas CAS0, CAS1 e CAS2 são ligadas entre o Mestre e o(s) Escravo(s) funcionando como saídas no Mestre e como entradas no(s) Escravo(s). Estas linhas têm a ver com a identificação de cada Escravo pelo Mestre.

SP/EN é alto (1) no Mestre e baixo (0) no(s) Escravo(s).

Quando um Escravo tem um pedido de interrupção numa das suas entradas IR, então envia um pedido de interrupção através de INT ao Mestre; este, por seu turno, ao detectar um pedido de interrupção numa das suas entradas envia um pedido de interrupção ao processador através da sua linha INT. Quando o processador envia o seu 1º INTA, todos os Escravos vão ignorá-lo; só o Mestre é que o vai atender. Após esta recepção, o Mestre envia uma identificação (através de CAS0, CAS1 e CAS2) de acordo com a inicialização, que permitirá activar o respectivo Escravo. Se for ligada uma interrupção directamente ao Mestre, este trata-se como se estivesse a operar no modo simples.

### *3.7.3 Interface do 8259A*

Para o programador o PIC aparenta ser duas portas de I/O (ou localizações de memória) especificadas por A0. Contudo podem escrever-se em sete registos separados. O endereço específico do PIC é controlado por um descodificador de endereços ligados à entrada CS.

Assumindo que a interrupção activa é não-mascarável, o 8259A pede uma interrupção ao microprocessador colocando INTR em 1. O processador, após completar a sua instrução corrente, emite INTA que é recebida pelo 8259A como INTAB. Em seguida são colocados três bytes no Data Bus do sistema, sendo o primeiro 11001101 que corresponde ao código operativo para uma instrução CALL. Após recepção pelo microprocessador, são colocados dois pulsos de INTA adicionais que são usados para indicar os 16 bits de endereço correspondentes à interrupção activa. Note-se que o 8259A não permite que cada entrada de interrupção tenha o seu próprio endereço. Em

seu lugar existe um endereço base que pode ser programado bem como um intervalo de 4 ou 8 bytes.

O 8259A foi concebido para lidar com um grande número de pedidos de interrupções. Pode ser atribuída uma prioridade a cada uma das entradas de interrupção para ser usada quando ocorrem pedidos de interrupção múltiplos.

### *3.7.4 Modos de operação do 8259A*

Podem-se programar seis modos diferentes de operação.

1. *Fully nested*: este é o modo por defeito e identifica IR0 como sendo o de mais alta prioridade e IR7 como o de mais baixa. Neste modo quando uma interrupção está a ser servida todas as outras de igual ou menor prioridade são lembradas (no IRR – Registo de Pedido de Interrupções) mas não servidas. Um bit do ISR – Registo In-Service - será activado de acordo com a interrupção activa. Quando a rotina de serviço da interrupção terminar é necessário que seja dada uma instrução de fim de interrupção (EOI) não específica. Isto irá fazer o *reset* (desactivação) do bit in-service mais alto. A instrução EOI é dada escrevendo uma palavra de controlo de operação ao registo de comandos. Ao escrever-se uma palavra de controlo de inicialização especial é também possível programar o 8259 para inserir automaticamente o comando de fim de interrupção não específico. Neste caso o último bit in-service irá ser desactivado no último pico do último pulso INTA.
2. Prioridade Rotativa Igualitária: Este modo é usado quando todos os dispositivos têm a mesma prioridade. Neste caso é dada a prioridade menor à interrupção que está a ser executada, após se completar o seu tratamento.
3. Prioridade Rotativa Específica: Se for colocada uma palavra de controlo específica na EOI, pode determinar-se a estrutura de prioridades, após se tratar a interrupção actual, na rotina de serviço. Isto é conseguido especificando a entrada de mais baixa prioridade como parte da palavra de controlo. A vantagem deste modo é que a rotina de serviço pode determinar as prioridades dependendo na execução do programa em vez de um esquema de rotação fixo, o que confere maior flexibilidade ao programador.
4. Máscara Básica: Permite ignorar qualquer interrupção colocando o bit respectivo em IMR (Registo de Máscaras de Interrupções) em 1. Esta interrupção será então ignorada em todos os modos.
5. Máscara Especial: Se estiver escolhido este modo e se a rotina de serviço actual escrever um 1 no IMR (mascarar-se a si própria) é permitido atender uma interrupção de prioridade mais baixa (o que não acontece normalmente) bem como uma interrupção de mais alta prioridade.
6. *Polled*: não é usada a saída INT. Vai dar prioridade às entradas IR0 a IR7 e fornece uma porta de estado que permite ser consultada (*polled*). O byte de estado é da forma 1 X X X X W2 W1 W0 (em que o 1 inicial significa uma entrada activa e os três últimos bits - W2, W1 e W0 – representam o código binário para o pedido em serviço com prioridade mais alta).

### *3.7.5 Programação do 8259A*

Para se programar o 8259A são necessárias duas a quatro palavras de inicialização (ICWs). Estes bytes especificam o vector de endereço, seleccionam um

intervalo entre os endereços de 4 ou 8 bytes, identificam o modo simples ou em cascata, seleccionam o modo normal ou automático de EOI e programam o tipo de *trigger* (disparo).

**Palavras de controlo de inicialização.**

Os significados de cada um dos bits de ICW1 a ICW4 estão representados no Apêndice I – a). Em todos os casos ICW1 e ICW2 têm de ser escritas. Estes bytes programam o endereço base que juntamente com o intervalo irá ser ligado ao Data Bus durante INTA. ICW3 deve ser escrita só quando se pretendem usar um ou mais PIC's Escravos. ICW4 é necessária para especificar um processador 8086 ou 8088, para activar o modo EOI automático e o modo *fully nested* especial (para PIC's em cascata) e seleccionar o modo de *buffer* para o escravo e mestre. Esta última selecção é necessária quando são usados *buffers* de Data Bus entre o PIC e o Data Bus do sistema.

Igualmente no Apêndice I – b) pode ser observado o diagrama de fluxos com os passos requeridos para a inicialização do 8259A.

**Palavras de controlo de operação.**

Os significados de cada um dos bits de OCW1 a OCW3 estão representados no Apêndice II. As palavras de controlo de operação só são necessárias quando as condições por defeito (obtidas após a inicialização) têm de ser alteradas e para dar o comando EOI não automático no fim de cada rotina de serviço. OCW1 controla as máscaras das interrupções (1's lógicos escritos em OCW1 fará com que as interrupções seleccionadas sejam mascaradas e ignoradas pelo 8259A). Os modos de operação são seleccionados escrevendo em OCW2. Podem ser dados oito comandos – geralmente no fim de cada rotina de serviço de interrupção. Esse modos são:

1. EOI não específico – este comando deve ser usado no fim da rotina de serviço quando se está a operar no modo *fully nested*. Desactiva o bit in-service da interrupção mais alta. Enquanto este bit está activo não são permitidas interrupções de prioridade igual ou menor. Este comando é inserido automaticamente pelo 8259A se o bit D1 de ICW4 for 1.
2. EOI específico – usa-se este comando para forçar um bit in-service específico a ser desactivado. Este comando só deverá ser utilizado quando o modo *fully nested* não estiver implementado. Os bits 0-2 seleccionam o bit in-service.
3. Rotação em EOI não específico – este comando deve ser dado quando todos os periféricos têm a mesma prioridade. Causa uma rotação automática das prioridades, como explicado anteriormente.
4. Rotação no modo EOI automático (set) – este comando activa o modo Rotação em EOI não específico.
5. Rotação no modo EOI automático (clear) – este comando desactiva o modo Rotação em EOI não específico.
6. Rotação em EOI específico – este comando permite ao programador definir qual a prioridade mais baixa no modo de rotação. Por exemplo, se programarmos IR4 como o dispositivo com prioridade mais baixa então IR5 terá a prioridade mais alta.
7. Definição de prioridades – este comando altera a prioridade definindo o dispositivo com prioridade mais baixa.
8. Nenhuma operação.

A OCW3 é usada para activar o IRR ou o IRS para uma subsequente operação de leitura.

# 4. Periféricos

## *4.1 Interface de Portas Programável 8255*

O 8255 é um dispositivo de interface para Entradas/Saídas (I/O) de um modo geral, com 24 linhas organizadas como 3 portas I/O de 8 bits denominadas A, B e C. Estes bits não podem ser programados como Saídas ou Entradas; em vez disso os bits das portas A e B funcionam como um byte, enquanto que na porta C pode-se programar os 4 bits mais altos e os 4 mais baixos (em dois *nibbles* – 4 bits) separadamente.

O 8255 pode ser operar em três modos:

**Modo 0**: três portas I/O simples

**Modo 1**: duas portas I/O *handshaking*

**Modo 2**: porta I/O bidireccional com cinco sinais de *handshaking*

Os modos podem ser misturados. Por exemplo, a porta A pode ser programada para funcionar no modo 2, enquanto que a porta B pode ser programada para funcionar no modo 0. Existe, também, um modo de bit set/reset que permite aos bits individuais da porta C serem activados (set) ou desactivados (reset) para fins de controlo.

### *4.1.1 Interface do 8255*

O diagrama de blocos do 8255 encontra-se representado na Fig. 4.1.

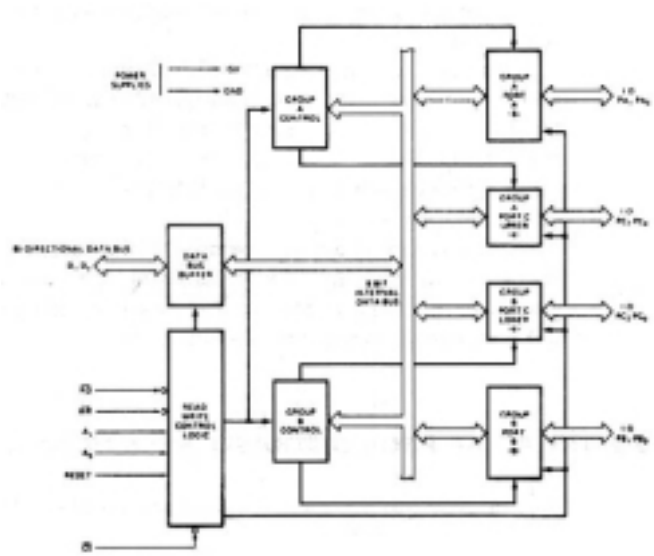


**Fig. 4.1**

Observando o diagrama de blocos do 8255 é de salientar o barramento de dados bidireccional. Todas as comunicações com a PPI (Interface de Portas Programável) ocorrem através destas oito linhas. De facto para o microprocessador o 8255 aparenta ser quatro localizações de I/O correspondentes aos endereços das linhas de entrada A0 e A1. O endereço específico é controlado pela entrada CS. Só quando esta entrada estiver a 0 é que o 8255 se encontra operacional. Na Tabela 4.1 encontram-se as operações de leitura e escrita possíveis com o 8255. Quando RD for 0 pode se ler qualquer das portas de informação, aplicando a combinação apropriada de A0 e A1. Quando A0 e A1 forem

ambas 1 tem-se acesso à porta de controlo. Este é um registo especial do 8255 que controla o modo de operação do dispositivo. É de notar que este registo só pode ser escrito. Quando a PPI não está acessível as ligações do barramento de dados estão num estado de alta impedância e o processador está livre para comunicar com qualquer outro dispositivo do sistema.

| A1 | A0 | RD | WR | CS | |
|---|---|---|---|---|---|
| | | | | | Operação de Entrada (LEITURA) |
| 0 | 0 | 0 | 1 | 0 | Porta A → Barramento de Dados |
| 0 | 1 | 0 | 1 | 0 | Porta B → Barramento de Dados |
| 1 | 0 | 0 | 1 | 0 | Porta C → Barramento de Dados |
| | | | | | Operação de Saída (ESCRITA) |
| 0 | 0 | 1 | 0 | 0 | Barramento de Dados → Porta A |
| 0 | 1 | 1 | 0 | 0 | Barramento de Dados → Porta B |
| 1 | 0 | 1 | 0 | 0 | Barramento de Dados → Porta C |
| 1 | 1 | 1 | 0 | 0 | Barramento de Dados → Controlo |
| | | | | | Função de Desactivação |
| X | X | X | X | 1 | Barramento de Dados → Alta Imp. |
| 1 | 1 | 0 | 1 | 0 | Condição Ilegal |
| X | X | 1 | 1 | 0 | Barramento de Dados → Alta Imp. |

**Tabela 4.1**

### *4.1.2 Modos de operação*

Quando a PPI já estiver a fazer de interface ao CPU é necessário seleccionar um modo de operação. Tal como já foi dito é possível operar em três modos mais o modo de bit set/reset. Vai-se em seguida ver o funcionamento de cada um desses modos.

Basta escrever única uma palavra de controlo na porta de controlo para determinar o modo de operação. A Fig. 4.2 mostra os dois tipos de palavras de controlo possíveis. Quando o bit 7 da palavra de controlo é 0 selecciona-se o modo bit set/reset – a); se for 1 pode se escolher qualquer um dos três modos das portas – b).

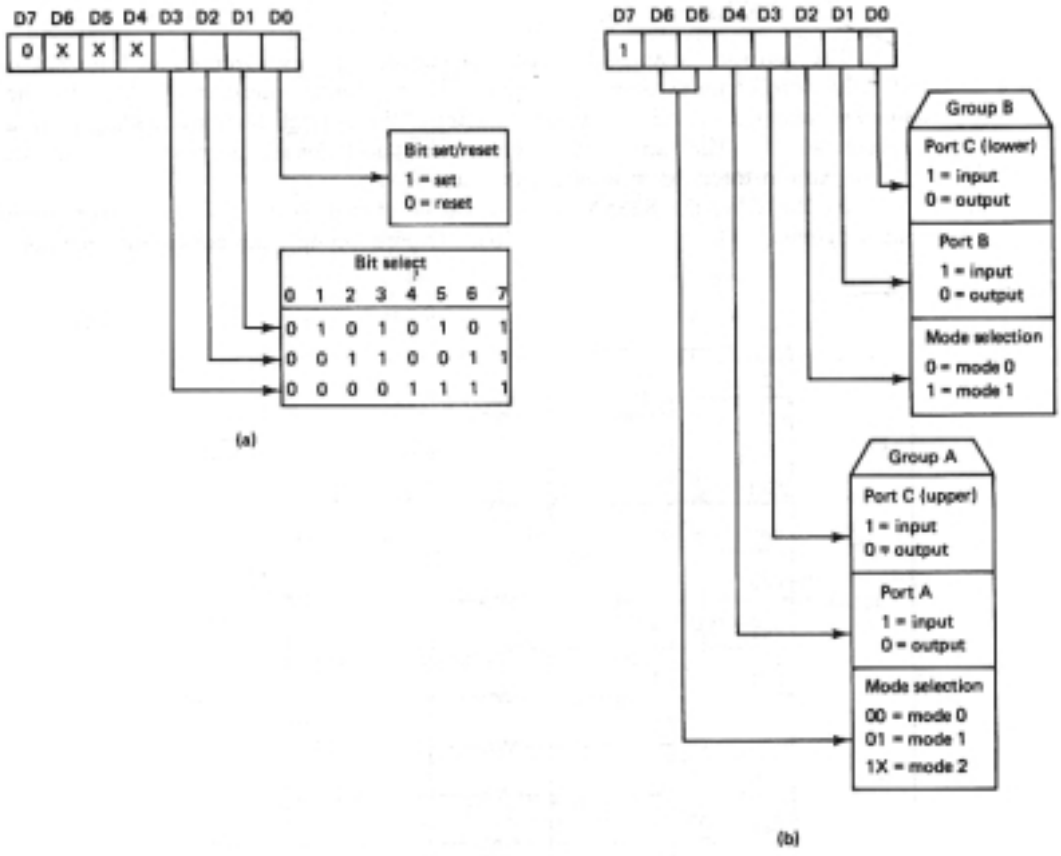



**Fig. 4.2**

Estudando a Fig. 4.2 pode se verificar que as três portas estão divididas em dois grupos para a selecção do modo. A porta A e os bits mais altos da porta C podem ser programados para qualquer um dos modos 0 a 2. A porta B e os bits mais baixos da porta C só podem ser programados para operar nos modos 0 ou 1.

**Modo 0: I/O Básicas**

Este modo deve ser seleccionado quando se pretende operar com I/O incondicional e é útil quando se pode assumir que o dispositivo de I/O está sempre pronto e quando não são necessários sinais de *handshaking*.

**Modo 1: I/O estroboscópicas**

Usa-se este modo quando se lida com interfaces I/O de interrupções e de *handshaking*. Neste modo as portas A e B são programadas como portas de informação e a porta C é programada para transportar sinais de estado. Uma das características únicas deste modo é que se podem efectuar transferências de informação sem intervenção directa do CPU. Existem quatro configurações possíveis quando se está a operar no modo 1 e correspondem às quatro combinações das portas A e B como entradas e saídas.

*Timing da porta de entrada*

Quando a porta A ou B está programada como porta de entrada, existem três sinais de controlo que são dedicados a suportar transferências de informação nesta porta. Esses sinais são IBF, STB e INTR. O significado destes sinais está na Tabela 4.2 e a Fig. 4.3 mostra o *timing*.

| Sinal | Direcção | Descrição |
|---|---|---|
| IBF | FORA | Um 1 nesta saída indica que a informação foi carregada para o *latch* de entrada; basicamente um sinal de reconhecimento IBF é activado pelo flanco descendente de STB e é reactivado pelo flanco ascendente da entrada RD. |
| STB | DENTRO | Um 0 nesta entrada carrega a informação para a entrada de *latch*. |
| INTR | FORA | Um 1 nesta saída pode ser usado para interromper o CPU quando um dispositivo de entrada pede serviço. INTR é activado pelo flanco ascendente de STB se IBF for 1 e INTE for 1. Este procedimento permite a um dispositivo de entrada pedir serviço do CPU simplesmente fazendo um *strobe* da sua informação na porta. INTE A é controlado pelo bit set/reset PC4 e INTE B é controlado pelo bit set/reset PC2. |

**Tabela 4.2**

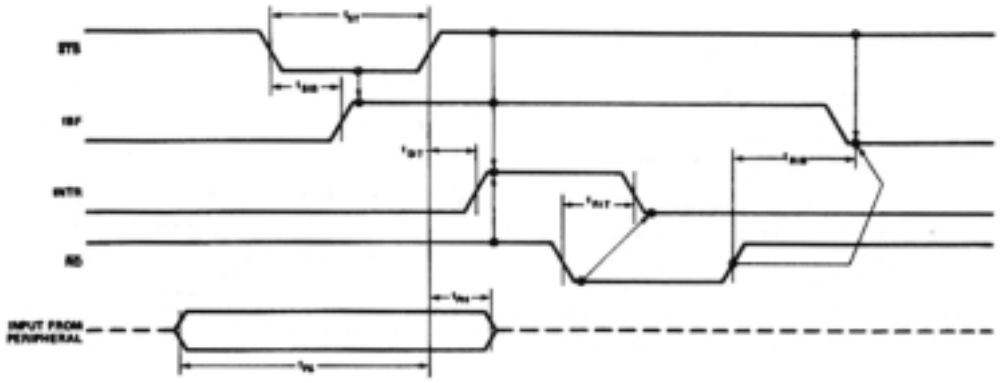



**Fig. 4.3**

Quando o periférico tem informação para o microprocessador coloca-a nas linhas de entrada da porta A ou B e envia um impulso para o 8255 na entrada STB. A PPI responde guardando esta informação e pondo a sua saída IBF a 1. Isto é um sinal para o periférico indicando que a informação foi guardada mas ainda não foi lida pelo processador.

Se o bit de INTE (*INTerrupts Enabled*) foi activado, IBF irá igualmente fazer com que a saída INTR fique 1. O processador agora tem a possibilidade de eleger a linha IBF lendo a sua palavra de estado no modo 1 ou deixar INTR gerar uma interrupção alertando, deste modo, o processador que o *buffer* de entrada está cheio.

Em qualquer dos casos o processador deve saltar para uma rotina que leia a informação da porta. O flanco descendente de RD faz com que a saída INTR do 8255 seja desactivada (reset) e o flanco ascendente de RD desactiva (reset) IBF. A

transferência de informação está, agora, completa e o periférico (monitorando IBF) pode fazer o *strobe* do próximo byte de informação.

*Timing da porta de saída*

Quando as portas A ou B estão programada como portas de saída no modo 1, existem três linhas da porta C que são dedicadas a suportar esta função. Essas linhas são OBF, ACK e INTR. O seu significado está na Tabela 4.3 e a Fig. 4.4 mostra o *timing*.

| Sinal | Direcção | Descrição |
|---|---|---|
| OBF | FORA | A saída OBF será 1 para indicar que o CPU escreveu informação na porta especificada. O *flip-flop* de OBF será activado (set) pelo flanco ascendente da entrada WR e desactivado (reset) pelo flanco descendente do sinal de entrada ACK. |
| ACK | DENTRO | Um 0 nesta entrada informa o 8255 que a informação da porta A ou da porta B foi aceite. Basicamente é uma resposta do dispositivo periférico indicando que já recebeu a informação de saída do CPU. |
| INTR | FORA | Um 1 nesta saída pode ser usado para interromper o CPU quando um dispositivo de saída recebeu informação transmitida pelo CPU. INTR é activado pelo flanco ascendente de ACK se OBF for 0 e INTE for 1. É desactivado (reset) pelo flanco descendente de WR. INTE A é controlado pelo bit set/reset PC6 e INTE B é controlado pelo bit set/reset PC2. |

**Tabela 4.3**

Assumindo que existe informação escrita previamente numa das portas de informação, o periférico monitoriza OBF (*output buffer full*). Quando esta linha for 0 existe informação pronta para ser lida pelo periférico. Usando OBF como uma entrada STROBE, o periférico guarda o byte de informação e responde com um pulso ACK. O flanco descendente deste pulso desactiva OBF e se INTE for 1 o flanco ascendente faz com que INTR também seja 1.

Também neste caso o CPU tem a oportunidade de eleger a linha OBF ao ler a palavra de controlo no modo 1 ou permitir que INTR alerte o CPU que OBF está em 1 e que o periférico está pronto para ler mais informação. Em qualquer dos casos o processador deve escrever um novo byte de informação na porta de saída. O flanco descendente de WR irá desactivar INTR e o flanco ascendente de WR irá forçar OBF a ser 0. O periférico, monitorando OBF, pode fixar o novo byte de informação e o ciclo repete-se.

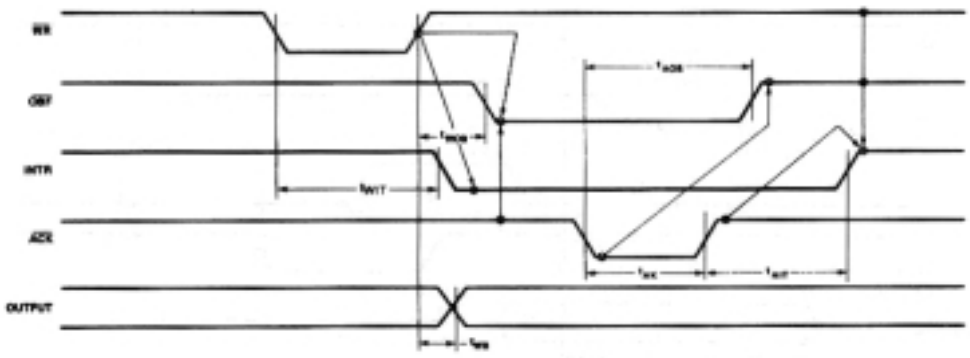



**Fig. 4.4**

**Modo 2: I/O estroboscópica bidireccional**

Quando se opera no modo 2, a porta A do 8255 torna-se uma porta bidireccional de informação suportada pelos cinco sinais de *handshaking*. Os sinais de *handshaking* são idênticos aos fornecidos no modo 1 excepto que agora são referidos unicamente à porta A. Este modo particular de operação é útil quando se transfere informação entre dois computadores.

Quando a porta A é programada para operar no modo 2, a porta B pode operar no modo 0 ou no modo 1. Se programada no modo 0, PC0-PC2 podem ser programados como entradas ou saídas em modo 0. Se a porta B for programada no modo 1, então PC0-PC2 tornam-se sinais de *handshaking* para esta porta.

*Timing da porta de entrada*

A Fig. 4.5 é um diagrama temporal ilustrando a sequência de eventos enquanto um byte de informação é transferido para o 8255 pelo periférico e posteriormente do 8255 para o periférico. Os números no diagrama são de seguida explicados.

1. É fornecida informação pelo periférico.
2. O periférico aplica um pulso STB ao 8255.
3. Quando a informação é guardada, IBF vai a 1.
4. Após STB retornar 1 com IBF ainda activo, INTR vai a 1, pedindo uma interrupção se esta característica for usada.
5. Agora podem-se usar *polling* ou interrupções para servir o periférico. O *buffer* do 8255 é lido quando RD for a 0.
6. O flanco descendente de RD desactiva INTR.
7. O flanco ascendente de RD desactiva IBF.

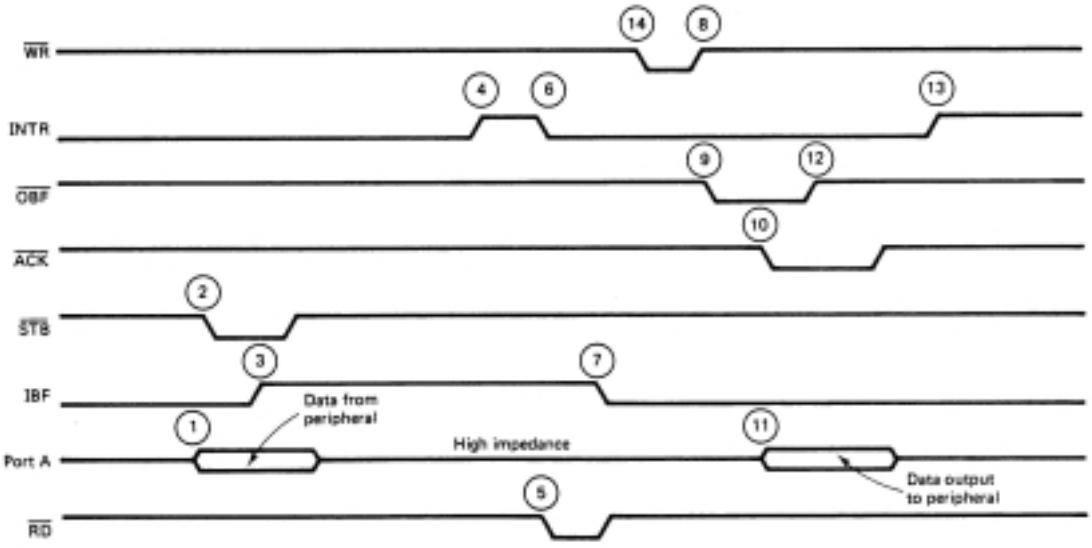



**Fig. 4.5**

*Timing da porta de saída*

A próxima sequência ocorre quando o processador coloca um byte de informação para o periférico através do 8255.

8. A informação é enviada pelo processador e guardada pelo 8255 (é de notar que o periférico nesta altura tem uma impedância de alto-estado).
9. O flanco ascendente de WR faz com que OBF mude para 0 (o *buffer* de saída está cheio).
10. O periférico reconhece OBF fazendo com ACK ir a 0.
11. No flanco descendente de ACK o 8255 liberta a sua informação no barramento.
12. OBF retorna a 1 (o *buffer* de saída está vazio).
13. O flanco ascendente de ACK activa INTR, pedindo uma interrupção se esta característica for usada.
14. Agora podem-se usar *polling* ou interrupções para escrever o próximo byte de informação no 8255.

Um ponto que não deve ser esquecido relativamente à operação no modo 2 é que só está disponível uma saída de INTR. Isto levanta a questão de como é que o processador pode determinar se a interrupção requer que seja lida ou escrita a informação – isto é, quem pediu a interrupção? Há duas soluções possíveis.

O *polling* pode ser usado quando há vários pedidos de interrupção partilhando a mesma linha de entrada. A palavra de estado no modo 2 é mostrada na Fig. 4.6. Com esta técnica a rotina de serviço de interrupção (ISR) pode eleger a porta C testando IBFA e OBFA. De seguida o controlo é transferido para a rotina apropriada.

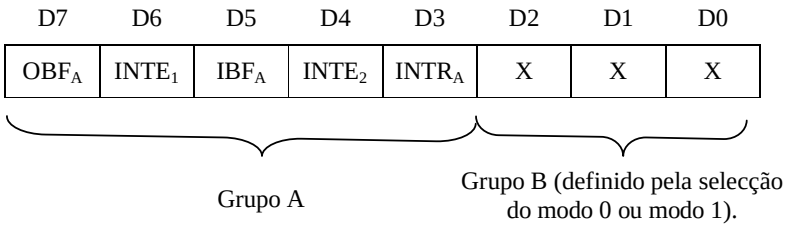



**Fig. 4.6**

Também é possível uma solução de *hardware*. O pedido de interrupção pode ser combinado com OBF ou IBF para gerar duas interrupções separadas. Estas podiam ser ligadas a um PIC (Controlador de Prioridades de Interrupções) que podia ser usado para instruções de *restart* únicas.

É de notar que se podem mascarar interrupções geradas por IBF ou OBF. Isto é feito desactivando INTE1 ou INTE2 com uma operação de desactivação de bit (*bit reset*). PC6 corresponde a INTE1 e PC4 a INTE2. Os mesmos bits devem ser activados que se quiser activar as interrupções.

**Modo bit set/reset**

Quando o bit 7 da palavra de controlo é 0 está activo o modo bit set/reset. Neste modo qualquer um dos bits da porta C pode ser activado (set) num 1 lógico ou desactivado (reset) num 0 lógico. É de notar que só se pode activar ou desactivar um bit de cada vez. Pode se tirar vantagem desta característica para gerar sinais de *strobe*.

## *4.2 UART e USART*

Existem protocolos que definem certas regras que devem ser seguidas para ajudar a fazer com que as técnicas de comunicação se tornem standard. Um exemplo disto é a adopção de um bit 0 de início e um bit 1 de paragem. Também há várias taxas de informação (*baud rates*) que se tornam standard. Quando se prepara uma porta série devem ser especificados vários parâmetros, sendo os mais comuns o número de bits por caracter (geralmente entre cinco e oito), o número de bits de paragem (um ou dois), o bit de paridade usado para a detecção de erros simples (paridade par ou ímpar ou sem paridade) e a taxa de débito.

### *4.2.1 UART*

UART são as iniciais de Receptor - Transmissor Universal Assíncrono.

Apesar da recepção e de transmissão de informação em série poder ser feita através de *software*, os programas que podem adaptar os vários protocolos podem tornar-se muito longos e tediosos. Geralmente empatam o processador em ciclos temporais, misturando bits de informação e geralmente requerem tempo que pode de outro modo ser utilizado mais eficientemente. Por esta razão as companhias criaram o *chip* UART. A Fig. 4.7 representa o diagrama de blocos de uma UART vulgar de segunda geração: a AY-5-1013 da General Instruments.

Este *chip* fornece um transmissor e um receptor independente e separado de informação em série. Os dois relógios de entrada (*16X CLOCK*) determinam a taxa de débito e devem ser escolhidos 16 vezes maiores que a taxa de informação (*data rate*)

pretendida. Por exemplo, se o transmissor operar a 300 baud, o pino do relógio deve receber uma onda quadrada de 4800 Hz.

Ao dividir cada tempo de bit em 16 períodos de tempo, a UART é capaz de localizar o centro de cada bit de informação com mais precisão. Existem algumas UART's que permitem que o relógio seja 32 ou 64 vezes a taxa de informação.

É de notar que por causa dos circuitos separados do receptor e do transmissor, pode ocorrer operação *full-duplex*. Isto seria dificilmente alcançado com a UART em "*software*".

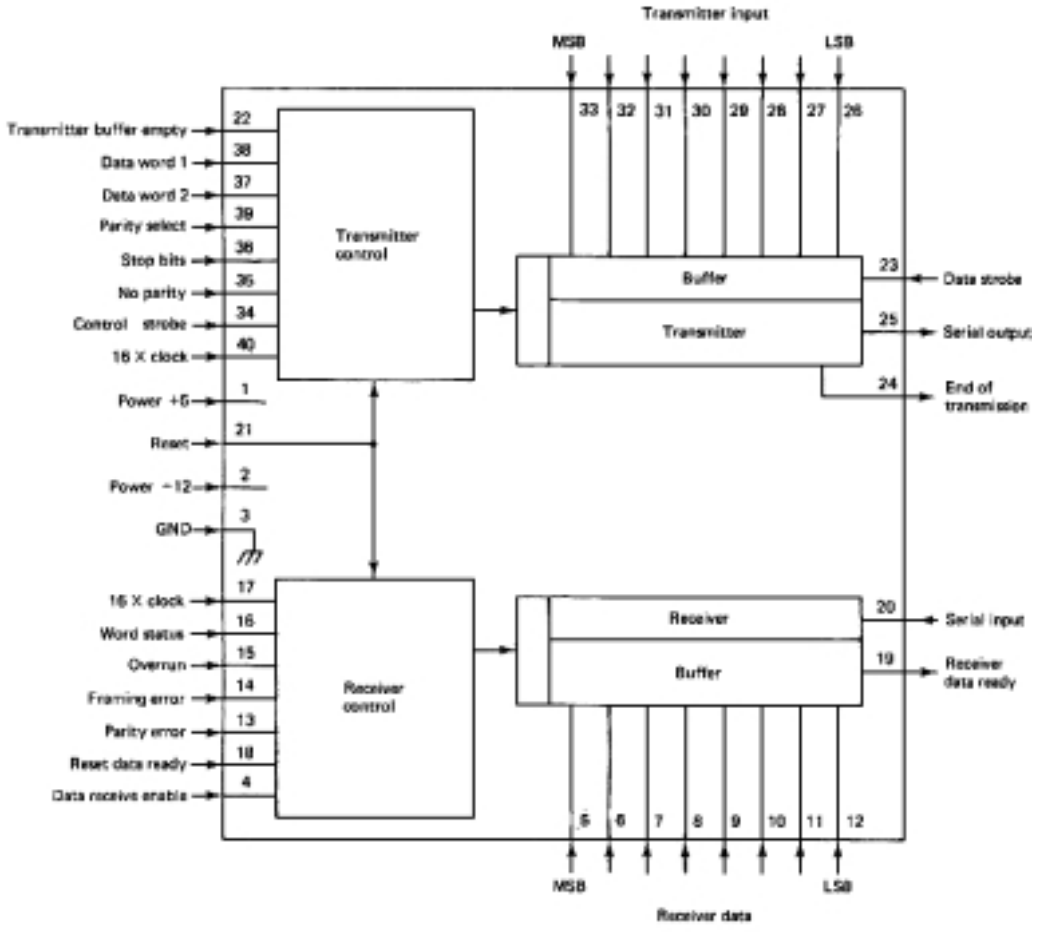



**Fig. 4.7**

Chamou-se ao AY-5-1013 um componente de segunda geração pois combina o receptor e o transmissor num único *chip*. Contudo, ao contrários dos componentes de terceira geração, tem de se ligar electricamente todas as suas funções de controlo, isto é, não podem ser programadas directamente pelo microprocessador. Podem-se seleccionar as seguintes funções:

1. Palavra de informação (*data word*) – DW1, DW2: permitem cinco a oito bits por palavra de informação
2. Selecção de paridade – PS: paridade par ou ímpar
3. Bits de paragem – SB: um ou dois

4. Sem paridade – NP: sem bit de paridade.

Adicionalmente a estas funções de controlo, podem-se monitorar três bits de estado:

1. *Buffer* do transmissor vazio - TBE: o transmissor está pronto para um novo caracter
2. Informação do receptor prontos - RDR: o receptor tem um caracter para ser lido
3. Fim de transmissão – EOT: não está a ser transmitido qualquer caracter.

Há outros três sinais que são fornecidos e que indicam condições de erro:

1. Erro de *frame* - FE: não foi recebido o bit de paragem
2. Erro de paridade – PE: a paridade da palavra recebida está incorrecta
3. Erro de *overrun* – OR: o novo caracter recebido foi escrito por cima do anterior.

Analise-se a Fig. 4.8 (um exemplo de como se pode fazer a interface da AY-5-1013 numa arquitectura de sistemas) e considere-se a sequência de eventos enquanto um caracter é transmitido e recebido para uma melhor compreensão.

1. A palavra de estado (RDR, TBE, PE, FE ou OR) é colocada em alta impedância pela UART e é activada a entrada *STATUS WORD INPUT*. Neste circuito a porta de entrada F1H é a porta de estado.
2. O microprocessador monitoriza a porta F1H e testa o bit 1, TBE. Se for 1, então está a ser escrito um byte de informação no transmissor na porta de saída F0H. O OUT F0H DSP (DATA STROBE) faz com que o byte de informação seja guardado e inicia a transmissão.
3. A UART insere automaticamente o bit de início, os 8 bits de informação, um bit de paridade par e dois bits de paragem. A taxa de informação será 1/16 da taxa do relógio.
4. Como a UART tem um *buffer* duplo, o TBE volta a 1 quando se envia o bit de início, indicando que se pode carregar um novo byte de informação (contudo, não será transmitido até o primeiro estar completo).
5. O microprocessador testa um caracter recebido lendo a sua porta de estado, neste caso F1H. Se RDR for 1 o receptor está a guardar o novo byte de informação.
6. As saídas do receptor não são colocadas em alta impedância pelo AY-5-1013 e assim sendo deve ser construída uma porta de entrada usando um *buffer* 74LS244. É de notar que o IN F0 DSP activa este *buffer* e faz o reset de RDR (isto evita a condição de *overrun*).

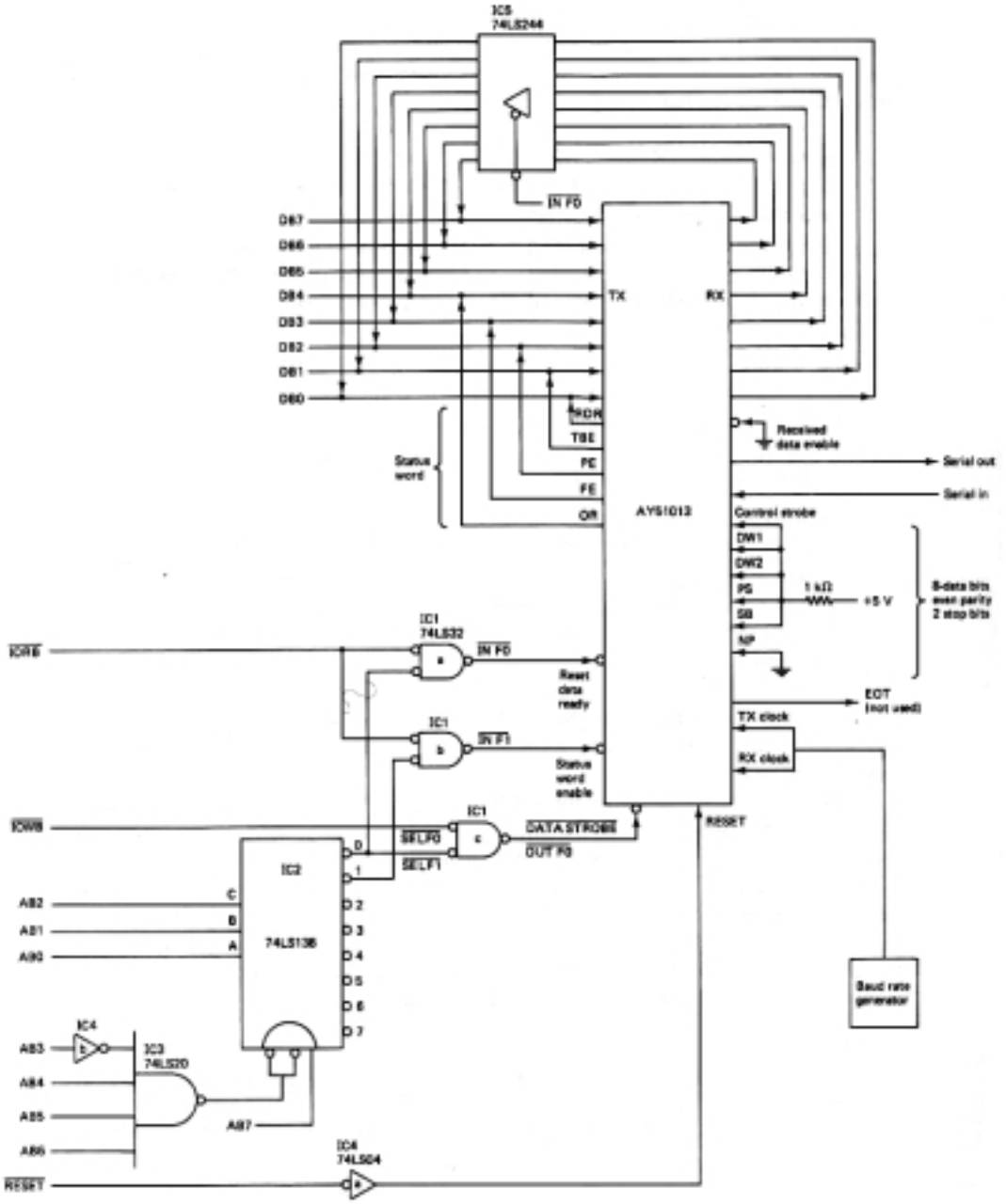



**Fig. 4.8**

O *software* que controla a interface da UART pode igualmente testar as três *flags* de erro enquanto espera que TBE e RDR estejam prontos. Contudo, em muitos casos, estas linhas são ignoradas.

A característica interessante da UART é que, para o microprocessador, a transferência de informação aparenta ser feita em paralelo. Ele escreve simplesmente um byte para a saída do transmissor e lê um byte da porta de entrada do receptor. A UART é que se encarrega de colocar a informação em série, inserir os bits de início, paragem e paridade, bem como de controlar a taxa de informação.

Geralmente controla-se a UART com uma simples rotina de *polling*, mas também se podem usar interrupções e DMA. Estas duas últimas opções podem ser boas escolhas devido ao relativamente longo período de tempo entre sinais prontos.

### *4.2.2 USART*

USART são as iniciais de Receptor - Transmissor Universal Síncrono/Assíncrono.

Como exemplo de uma USART vai-se estudar o 8251A da Intel. Sendo um componente de terceira geração, o 8251 pode ser programado pelo microprocessador para comunicações em série síncronas ou assíncronas. Igualmente sob o controlo de programa estão o número de bits por palavra de informação, o número de bits de paragem e a escolha de paridade.

Quando está a operar no modo síncrono cada *frame* é precedida por um ou dois caracteres de sincronização, conforme especificado. Estes caracteres serão inseridos automaticamente pelo transmissor e procurados pelo receptor quando no modo de "caça" (*hunt*).

Todos os sinais de estado podem monitorados através de uma porta de entrada de 8 bits. São igualmente fornecidas *flags* para erros de paridade, de *framing* e *overrun*.

*Interface do 8251A*

A Fig. 4.9 representa o diagrama de blocos do 8251A.

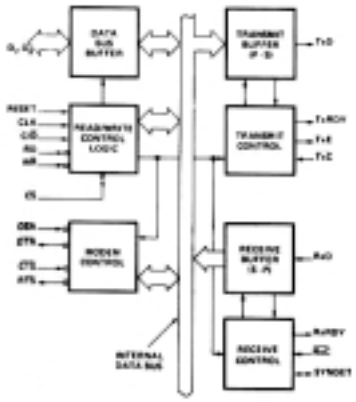


**Fig. 4.9**

A Fig. 4.10 mostra uma interface típica.

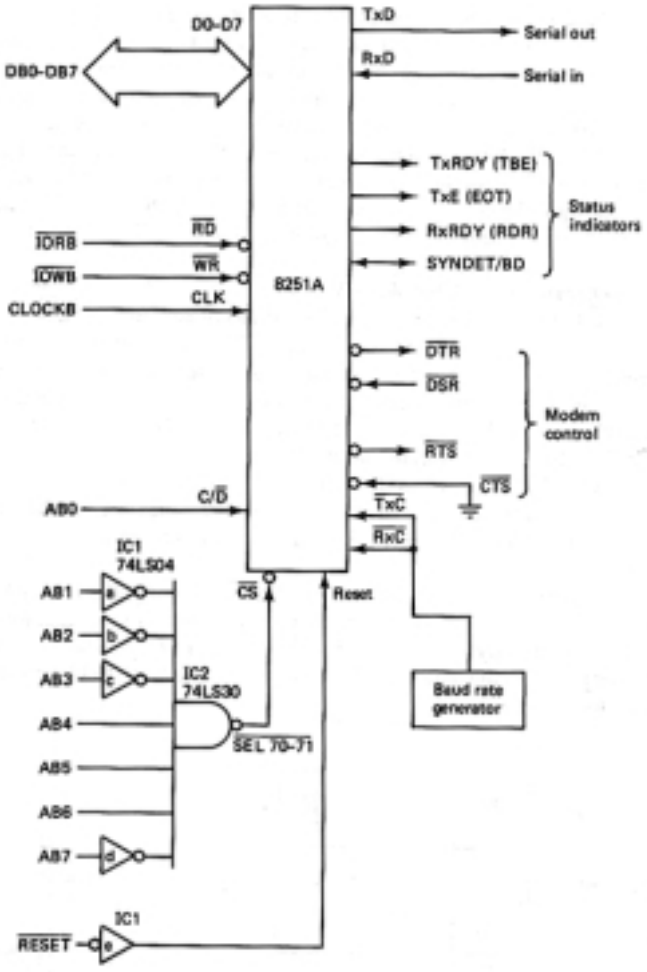



**Fig. 4.10**

Como é costume nos dispositivos de I/O da Intel a entrada CS tem de ser 0 para que o dispositivo seja seleccionado.

A entrada C/D selecciona as funções de controlo ou as funções de informação da USART, tal como indicado na Tabela 4.4.

| C/D | RD | WR | Função |
|---|---|---|---|
| 0 | 0 | 1 | Lê uma palavra de informação |
| 0 | 1 | 0 | Escreve uma palavra de informação |
| 1 | 0 | 1 | Lê a palavra de estado |
| 1 | 1 | 0 | Escreve a palavra de estado |

**Tabela 4.4**

A entrada CLK não está relacionada com relógios de taxa baud mas é precisa para o *timing* interno. Deve ter uma frequência pelo menos 30 vezes maior que a taxa de informação. Para 19,2 baud dá 576 kHz.

A Fig. 4.10 mostra duas entradas de relógios de taxa baud, T×C e R×C ligados em conjunto. Isto fará com que o receptor e o transmissor operem à mesma taxa de informação tal como é normalmente desejado. Sob o controlo de programa os relógios podem ser escolhidos como 1, 16 ou 64 vezes a taxa de informação. Isto permite que a taxa baud variar sem mudar a frequência do relógio.

O bloco assinalado como *baud rate generator* na Fig. 4.10 pode ser implementado como uma das três maneiras:

1. *Oscilador TTL*: este tipo de circuito requer três inversores 74LS01e um cristal. Apesar de ser uma solução muito simples tem como principal desvantagem o facto de não se poder alterar a frequência sem alterar o tempo de base do cristal.
2. *Oscilador PIT*: já foi discutido um PIT: o 8254. Este PIT fornece uma solução agradável para o problema da taxa de baud do relógio – agradável pois podem ser programadas várias taxas de baud alterando a contagem inicial no registo PIT programado como gerador de onda quadrada.
3. *Gerador externo de taxa baud*: existe o Motorola MC14411 que fornece 14 relógios de taxas de baud diferentes que podem ser ligadas às entradas de relógio da USART.

Ainda acerca da Fig. 4.10, pode se observar que o 8251A fornece quatro sinais de estado para controlo externo. Todos estes sinais podem igualmente ser monitorizados internamente através da porta de estado. T×E indica que o *buffer* do transmissor está vazio e que não estão a ser transmitidos quaisquer caracteres.

T×RDY e R×RDY indicam uma condição de prontos para o transmissor e para o receptor. Devem ser usados para pedir uma interrupção ou para iniciar uma transferência DMA. Também podem ser testados fazendo *polling* da porta de estado interna.

SYNDET/BD é um sinal que vai a 1 quando se está a operar no modo síncrono e foi detectado o caracter de sincronização. Repare-se que este pino também pode ser programado como entrada, sendo, neste caso, usado para fornecer o sinal de sincronização externamente. Quando se está a operar no modo assíncrono, SYNDET/BD é uma saída que fica 1 para indicar uma condição de quebra. O caracter de quebra é um 0 lógico constante normalmente enviado pelo receptor ao transmissor para suspender a transmissão (talvez devido a uma condição de erro).

Também são fornecidos quatro sinais de modem: DTR e RTS, ambos saídas, e DSR e CTS, ambos entradas. Estes sinais destinam-se a aplicações com *handshaking*. Repare-se que CTS deve ser 0 para permitir a função de transmissão no 8251A. A entrada DSR é de fim geral e pode ser monitorizada através da palavra de estado.

*Programação do 8251A no modo Assíncrono*

Quando se programa o 8251A no modo assíncrono deve se seguir a seguinte sequência:

1. Fazer o reset (interna ou externamente)
2. Instrução do modo (especificar o modo assíncrono)
3. Instrução de comando.

O Apêndice III descreve a forma das instruções de modo e de comando. Deve ser usado um comando de reset para começar a sequência de inicialização. O próximo comando, apenas, será interpretado como uma instrução do modo. Após a instrução do modo estar escrita, todas as escritas futuras serão interpretadas como instruções de comando. A única maneira de retornar à instrução do modo é aplicar um pulso de reset ou escrever uma palavra de comando com o bit 6 em 1.

Apesar de a maior parte das aplicações usar o 8251A num modo de *polling*, também se podem usar interrupções e DMA. Num ambiente de interrupções T×RDY e R×RDY podem ser usados para iniciar o pedido de interrupção. Se isto for feito, repare-se que o pino de saída T×RDY pode ser mascarado fazendo com que a entrada CTS seja 1 ou dando o comando *transmit enable inactive command*. Se for usada DMA T×RDY é ligada à entrada DREQ do dispositivo de DMA.

Muitas vezes associam-se as interrupções e a DMA com transferências de alta velocidade e aparentemente seriam incompatíveis com as taxas de informação lentas geralmente associadas com técnicas de I/O em série. Contudo, existem boas razões para se usar DMA ou interrupções para controlar uma porta série. A taxa de informação inerentemente baixa da porta série significa que o *polling* será um desperdício muito grande dos recursos do microprocessador. Usando interrupções ou DMA, pode se transferir informação para a porta só quando necessário, dando deste modo tempo considerável ao processador entre transferências de informação.

*Programação do 8251A no modo síncrono*

Quando se programa o 8251A no modo síncrono deve se seguir a seguinte sequência:

1. Fazer o reset (interna ou externamente)
2. Instrução do modo (especificar o modo síncrono e o número de caracteres de sincronização)
3. Um ou dois caracteres de sincronização
4. Instrução de comando.

Tal como no modo assíncrono, a instrução do modo só pode ser escrita imediatamente após um reset. A seguir a esta instrução a USART espera um ou dois caracteres de sincronização. As escritas seguintes serão interpretadas como instruções de comando.

No Apêndice IV encontra-se o formato para a instrução do modo quando a USART é operada no modo síncrono. A instrução de comando não muda relativamente ao modo assíncrono.

Depois de se programar a USART para o modo síncrono, a linha de saída série irá a 1 até CTS ser 0. Nesta altura o conteúdo do *buffer* do transmissor irá ser colocado em série e enviado. Normalmente isto serão um ou dois caracteres de sincronização.

Elegendo T×RDY, o processador pode agora enviar a informação a ser transmitida. Se o *buffer* do transmissor ficar vazio a qualquer altura, os caracteres SYNC serão inseridos automaticamente. Tal é ilustrado na Fig. 4.11.

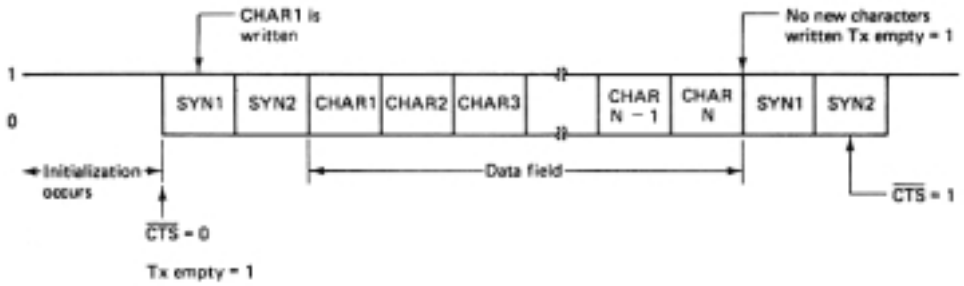



**Fig. 4.11**

Repare-se que é transmitido um bit por cada pulso do relógio de taxa baud (ou seja, no modo síncrono não se aplicam as opções de relógio 16× e 64×).

Quando está a receber informação a instrução de comando deverá especificar "*enter hunt mode*". A USART irá, então, testar a informação vinda após receber cada bit para uma comparação com o caracter de sincronização. Quando a comparação coincide SYNDET/BD irá a 1, indicando sincronização de caracter. Agora pode se ler informação elegendo R×RDY.

## *4.3 Teclado*

### *4.3.1 Tipos de teclado*

Quando se pressiona uma tecla no computador está se a activar um interruptor. Existem várias maneiras de fazer esses interruptores. De seguida daremos uma vista de olhos na construção e no funcionamento de alguns desses interruptores.

**Interruptores de teclas mecânicos**

Neste tipo de interruptores, quando se pressiona a tecla, unem-se duas peças de metal. Os elementos do interruptor são na realidade feitos de uma liga de fósforo e bronze com chapas douradas nas áreas de contacto. O interruptor da tecla normalmente contem uma mola para fazer a tecla voltar à posição de não pressão e talvez um pequeno pedaço de espuma para ajudar a enfraquecer uma saída abrupta. Alguns interruptores de teclas mecânicos actualmente consistem numa cúpula moldada em silicone com uma pequena peça de borracha condutora no lado de baixo. Quando se pressiona uma tecla a espuma da borracha curto-circuita duas pistas na placa de circuito impresso para produzir o sinal de tecla pressionada.

Os interruptores mecânicos são relativamente baratos, mas têm sérias desvantagens. Em primeiro lugar, sofrem de ressalto do contacto. Uma tecla premida pode estabelecer e cortar o contacto várias vezes antes de estabelecer um contacto sólido. Em segundo lugar, os contactos podem oxidar ou ficar gastos ou sujos com a idade e deixar de fazer uma ligação fiável. Os interruptores mecânicos de melhor qualidade têm geralmente uma duração de 1 milhão de golpes. Os de tipo de cúpula de silicone duram normalmente 25 milhões de golpes.

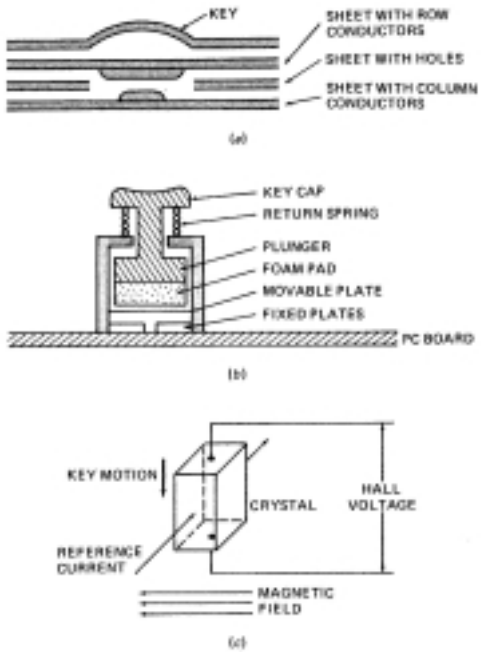



**Fig. 4.12**

**Interruptores de teclas de membrana**

Estes interruptores são na realidade um tipo especial de interruptores mecânicos. Consistem numa "sanduíche" de três camadas de plástico ou borracha, tal como mostrado na Fig. 4.12a). A camada do topo tem uma linha condutora de tinta de prata correndo debaixo de cada linha de teclas. A camada do meio tem um buraco debaixo de posição de tecla. A camada de baixo tem uma linha condutora de tinta de prata correndo debaixo de cada coluna de teclas. Quando se pressiona uma tecla, empurra-se a linha de tinta do topo através do buraco para estabelecer contacto com a linha de tinta de baixo. A vantagem dos teclados de membrana é que podem ser construídas unidades muito finas e seladas. São normalmente usados em caixas registadoras em restaurantes de *fast-food*, em instrumentos médicos e em outras aplicações com confusão. A duração destes teclados varia numa vasta gama de valores.

**Interruptores de teclas capacitivos**

Tal como mostrado na Fig. 4.12b), um interruptor de tecla capacitivo tem duas pequenas chapas de metal no fundo de uma peça de espuma. Quando se pressiona a tecla a parte móvel é encostada à chapa fixa. Isto altera a capacidade entre as duas chapas fixas. Um circuito sensor amplificado detecta esta alteração na capacidade e produz um sinal de nível lógico (0 ou 1) indicando que a tecla está a ser pressionada. A grande vantagem deste tipo de interruptores é que não têm contactos mecânicos capazes de oxidar ou ficarem gastos ou sujos. Uma pequena desvantagem é que é necessário um circuito especializado para detectar a mudança na capacidade. Este tipo de interruptores de teclas normalmente tem a duração de 20 milhões de golpes.

**Interruptores de teclas de efeito de corredor**

Este é outro tipo de interruptor sem contactos mecânicos. Aproveita-se do efeito de deflexão de uma carga móvel por campo magnético. A Fig. 4.12c) mostra o seu funcionamento. Um cristal semicondutor é atravessado por uma corrente de referência entre duas faces opostas. Quando uma tecla é pressionada, o cristal desloca-se por um campo magnético que tem as suas linhas de fluxo perpendiculares à direcção do fluxo de corrente no cristal. O deslocamento do cristal através do campo magnético causa o desenvolvimento de uma pequena diferença de tensão entre duas das outras faces opostas do cristal. Esta diferença de tensão é amplificada e utilizada para indicar que uma tecla foi pressionada. Este tipo de interruptores de teclas são mais caros por causa dos mecanismos de interrupção mais complexos e têm uma duração média de 100 milhões de golpes ou mais.

### *4.3.2 Ligações de circuito e interface de um teclado*

Na maioria dos teclados, os interruptores de teclas são ligados numa matriz de linhas e colunas como mostrado na Fig. 4.13.

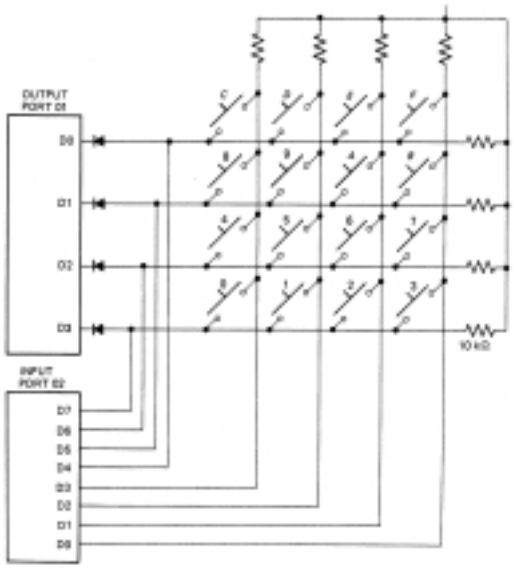



**Fig. 4.13**

Para os exemplos aqui ilustrados só irão ser usados interruptores mecânicos, mas o princípio é o mesmo para os outros tipos de interruptores. Obter dados significativos dum teclado destes são requeridas as seguintes três tarefas:

1. Detectar o pressionar de uma tecla
2. Libertar a tecla pressionada
3. Codificar a tecla pressionada (produzir um código standard para a tecla pressionada)

As três tarefas podem ser efectuadas através de *software*, *hardware* ou uma combinação dos dois tipos, dependendo da aplicação. Vai se começar por mostrar como podem ser feitas através de *software*, como pode ser feito num microprocessador baseado numa escala de mercearia onde pode ocorrer que uma tecla não seja premida

durante muito tempo. Posteriormente serão descritos alguns dispositivos de *hardware* que efectuam essas tarefas.

### *4.3.3 Interface em software do teclado*

A Fig. 4.13 mostra como é que um teclado hexadecimal pode ser ligado a um par de portas do micro-computador de modo a que as três tarefas de interface possam ser feitas como parte dum programa. As linhas da matriz são ligadas a quatro linhas da porta de saída. As linhas das colunas da matriz são ligadas a quatro linhas da porta de entrada. Para tornar o programa mais simples, as linhas das linhas da matriz também são ligadas a quatro linhas de entrada.

Quando não há teclas a ser pressionadas, as linhas das colunas são mantidas a 1 pelas resistências de *pull-up* ligadas a +5 V. Ao pressionar uma tecla liga-se uma linha da matriz a uma coluna. Se for colocado um 0 na saída de uma linha e uma tecla dessa linha for premida, então o 0 irá aparecer na coluna que contem a tecla e pode ser detectado na porta de entrada. Se forem conhecidas a linha e a coluna de tecla premida e então sabe-se qual a tecla que foi premida e pode se converter esta informação para um código qualquer em que se queira representar essa tecla. A Fig. 4.13 representa um fluxograma de um procedimento que detecta, liberta e produz o código hexadecimal para a tecla premida.

Uma maneira fácil de detectar se qualquer tecla foi premida é enviar 0 para todas as linhas e verificar as colunas para ver se uma tecla premida ligou um 0 a uma coluna. No algoritmo da Fig. 4.14, primeiro envia-se 0 para todas as linhas e verifica-se as colunas repetidamente até que todas as colunas sejam 1. Isto é feito para assegurar que uma tecla foi previamente libertada antes de se olhar para a próxima. Quando as colunas forem todas 1, o programa entra noutro ciclo, que espera até que apareça um 0 numa das colunas, indicando que uma tecla foi premida. Este segundo ciclo faz a tarefa de detecção. Um procedimento simples de atraso de 20 ms faz a tarefa de libertação da tecla.

Após o tempo de libertação, é feita uma nova verificação para ver se a tecla continua premida. Se agora as colunas forem todas 1, então não há qualquer tecla a ser premida e a detecção inicial foi causada por  um pulso de ruído. Se qualquer uma das colunas continuar em 0, então assume-se que uma tecla foi premida.

A tarefa final é determinar a linha e a coluna da tecla premida e converter esta informação num código hexadecimal. Para obter a informação da linha e da coluna é enviado um 0 para uma linha e são lidas todas as colunas. Se nenhuma das colunas for 0 então a tecla premida não está naquela linha e o 0 é rodado para a próxima linha e verificam-se novamente as colunas. O processo é repetido até que um 0 numa linha produza um 0 numa coluna.

### *4.3.4 Interface em hardware do teclado*

Em circuitos em que o CPU esteja demasiado ocupado para efectuar as tarefas anteriormente descritas em *software*, pode ser usado um dispositivo externo para as fazer. Um exemplo de um dispositivo MOS que faça estas tarefas é AY5-2376 da General Instruments, que pode ser ligado às linhas e colunas da matriz de interruptores do teclado. O AY5-2376 detecta independentemente o premir de uma tecla, colocando, ciclicamente, as linhas a 0 e verificando as colunas tal como foi feito com *software*. Quando detecta a pressão de uma tecla, espera um tempo de libertação. Se a uma tecla continuar premida após o tempo de libertação, o AY5-2376 produz um código de 8 bits

e envia para, por exemplo, uma porta de oito linhas paralelas do micro-computador. Para fazer com que o micro-computador saiba que há um código ASCII válido nas linhas de dados, o AY5-2376 envia um pulso estroboscópico. O micro-computador pode detectar este pulso e ler o código ASCII através de *polling* ou através de interrupção. Com a interrupção o micro-computador não tem de prestar atenção ao teclado até receber um sinal de interrupção, por isso este método usa muito pouco do tempo do micro-computador.

O AY5-2376 tem uma característica chamada *two-key rollover*, o que significa que se duas teclas forem premidas no tempo muito próximas uma da outra, cada uma delas será detectada, libertada e convertida para ASCII.

Hoje em dia muitos computadores e terminais usam teclados separados com codificadores incorporados. Em vez de usarem um dispositivo de codificação como o AY5-2376, usam um microprocessador dedicado.

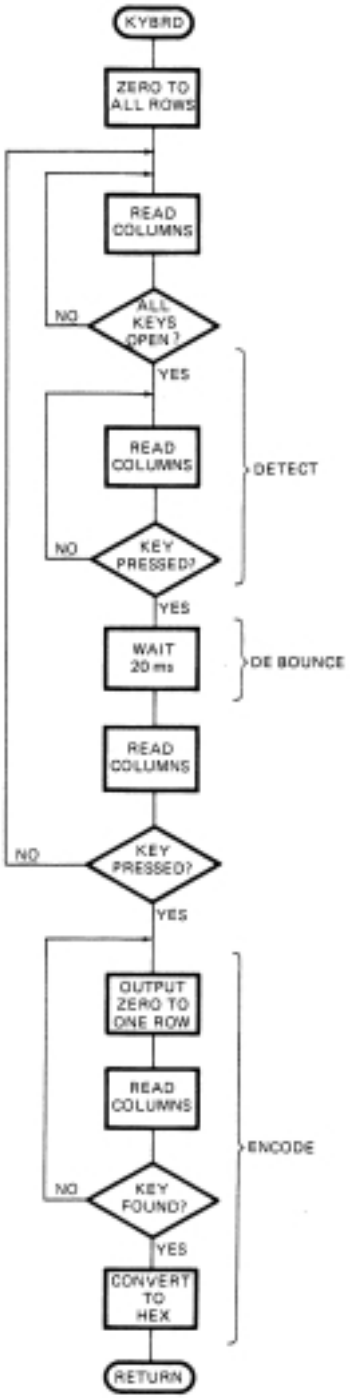
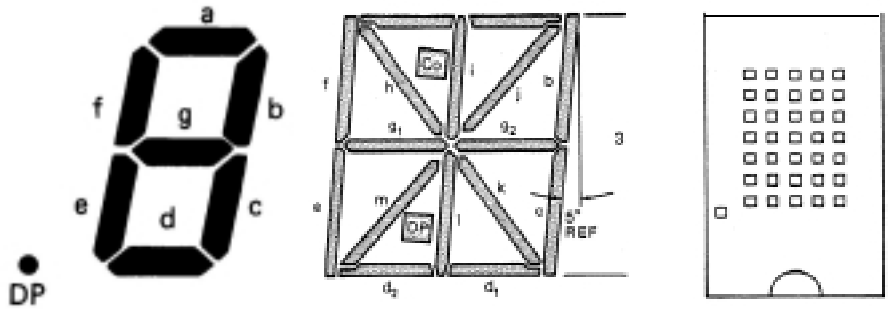



**Fig. 4.14**

### *4.4 Display*

Para dar indicações ou valores de dados aos utilizadores, muitos instrumentos e máquinas controlados por microprocessadores precisam de mostrar letras do alfabeto e números. Em sistemas que necessitam de mostrar grandes quantidades de informação é normalmente usado um CRT (*Cathode Ray-Tube*). Em sistemas onde só é necessário

mostrar pequenas quantidades de informação são usados *displays* do tipo dígito-único (*simple digit-type*). Há várias tecnologias usadas para fazer estes *displays digit-oriented*, mas só se vai estudar as duas mais importantes: LEDs (*Light Emiting Diodes*) e LCDs (*Liquid-Crystal Displays* ). Os LCDs consomem muito pouca energia, por isso são muito usados em aparelhos portáteis alimentados a baterias. Contudo, os LCDs não emitem a sua própria luz: eles simplesmente mudam a reflexão da luz disponível. Assim sendo, para um aparelho que tenha de ser utilizado em condições de pouca luz, tem de se incluir uma fonte de luz para os LCDs ou usar LEDs, que emitem a sua própria luz.




*a)* *b)* *c)*

**Fig. 4.15**

### *4.4.1 Interface de LEDs com micro-computadores*

Existem *displays* alfa-numéricos de LEDs disponíveis em três formatos: para mostrar unicamente números e letras hexadecimais, são usados *displays* de 7 segmentos como o mostrado na Fig. 4.15a); para mostrar números e todo o alfabeto, são usados *displays* de 18 segmentos como o mostrado na Fig. 4.15b) ou então usar *displays* de matriz de 5 por 7 pontos como na Fig. 4.15c). O *display* tipo 7 segmentos é o mais usado, mais barato e mais fácil de fazer interface, por isso vai se concentrar no seu estudo e mais tarde serão indicadas as alterações necessárias para fazer a interface com os outros tipos.

**Ligação directa dos *displays* LEDs**

A Fig. 4.16 mostra um circuito que pode ser ligado a uma porta paralela num micro-computador para guiar um único *display* de 7 segmentos em. Para um *display* em ânodo-comum, um segmento é activado aplicando-lhe um 0 lógico. O 7447 converte um código BCD aplicado nas suas entradas no padrão de 0 requerido para indicar o número representado pelo código BCD. Esta ligação de circuito é chamada *display estático* porque a corrente atravessa sempre o *display*.

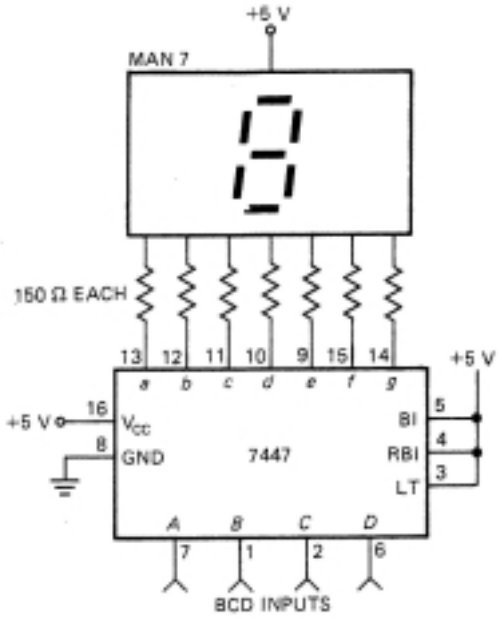



**Fig. 4.16**

***Displays* LEDs multiplexados por *software***

O circuito da Fig. 4.16 funciona bem para conduzir só um ou dois dígitos LED com uma porta paralela de saída. Contudo, este esquema tem vários problemas se se pretender, por exemplo, oito dígitos. O primeiro problema é o consumo de energia. Para cálculos em pior caso, assuma-se que todos os oito dígitos estão a indicar o dígito 8, ou seja, estão acesos todos os segmentos. Em casos típicos cada segmento consome 20 mA, o que dá um total de 140 mA por dígito, perfazendo um total de 1,12 A para os oito dígitos! Um segundo problema desta técnica é que cada *display* para um dígito requer um descodificador 7447 separado, cada um utilizando, talvez outros 13 mA (o consumo típico de um 7447). A corrente requisitada pelos LEDs e pelos descodificadores poderá ser várias vezes maior que o resto dos circuitos no aparelho.

Para resolver os problemas da técnica estática, usa-se o método de multiplexagem. Um circuito como exemplo é a melhor maneira de explicar como funciona a multiplexagem. A Fig. 4.17 mostra um circuito que pode ser acrescentado a um par de portas do micro-computador para conduzir vários *displays* LEDs em ânodo-comum num modo multiplexado. Repare-se que o circuito só tem um 7447 e que as saídas do 7447 estão ligadas em paralelo para as entradas dos segmentos de todos os dígitos. A questão que se pode colocar à primeira vista é: os dígitos não irão mostrar todos o mesmo número? A resposta é que sim caso fossem ligados todos ao mesmo tempo. O truque de multiplexar *displays* está em só activar um de cada vez. Os transístores PNP em série com o ânodo comum de cada dígito actuam como interruptores de *On/Off* para esse dígito.

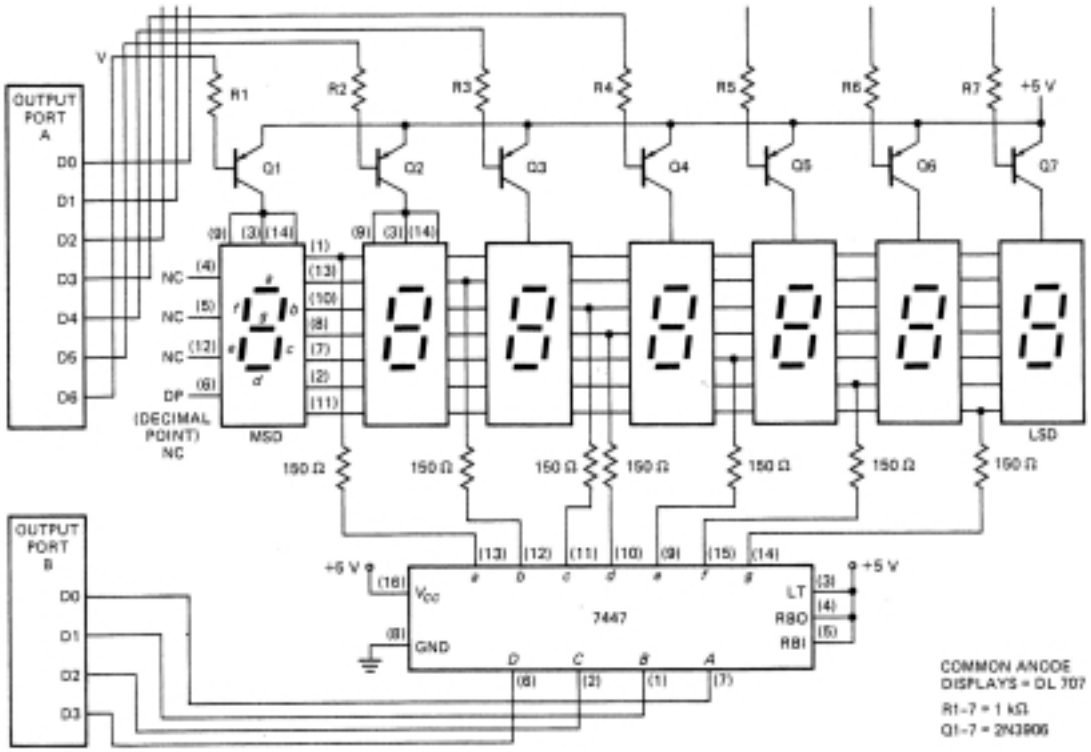



**Fig. 4.17**

A seguir descreve-se o funcionamento deste circuito.

O código BDC para o dígito 1 é enviado da porta B para o 7447. O 7447 envia o código de 7 segmentos correspondente nas linhas do barramento dos segmentos. O transístor ligado ao dígito 1 é ligado através de o envio de 0 no bit apropriado da porta A. Todos os outros bits da porta A são colocados a 1 para se certificar que nenhum outro dígito é activado. Após 1 ou 2 ms, o dígito 1 é desligado colocando todos os bits da porta A a 1. Então, o código BDC para o dígito 2 é enviado da porta B para o 7447 e uma palavra para ligar o dígito 2 é enviada pela porta A. Após 1 ou 2 ms, o dígito 2 é desligado e o processo é repetido para o dígito 3 e até que todos os dígitos tenham a sua vez. Então, o dígito 1 e os dígitos seguintes são acendidos novamente à vez.

Com oito dígitos e 2 ms por dígito, volta-se ao dígito 1 todos os 16 ms, ou aproximadamente 60 vezes por segundo. Esta taxa de refrescamento é suficientemente rápida para que, aos olhos humanos, os dígitos aparentem estar sempre acesos. Taxas de refrescamento de 40 a 200 vezes por segundo são aceitáveis.

A vantagem óbvia da multiplexagem é que só é necessário um 7447 e só está aceso um dígito de cada vez. Normalmente aumenta-se a corrente por segmento para 40 e 60 mA em *displays* multiplexados de modo a que aparentem ser tão brilhantes como se não fossem multiplexados. Mesmo com este aumento de corrente por segmento, a multiplexagem fornece uma grande poupança em energia e em peças.

Esta técnica também pode ser utilizada para conduzir dispositivos com LEDs de 18 segmentos e dispositivos de LEDs de matriz-ponto. Contudo, para estes dispositivos, substitui-se o 7447 da Fig. 4.17 por uma ROM que gere os códigos de segmentos necessários quando o código ASCII for aplicado nos endereços de entrada da ROM.

### *4.4.2 Interface e operações de LCDs com micro-computadores*

Os *displays* de cristais líquidos são feitos fazendo uma "sanduíche" de uma fina camada de um cristal líquido entre duas chapas de vidro. Um película ou plano de fundo é colocada na parte de trás da camada de vidro. Quando se aplica um diferença de tensão entre um segmento e o plano de fundo, cria-se uma corrente eléctrica na região por baixo do segmento. Este campo eléctrico muda a transmissão da luz através da região por baixo do segmento de plano de fundo.

Existem dois tipos comuns de LCDs: dispersão dinâmica e efeito de campo. O primeiro tipo mistura as moléculas onde o campo está presente. Isto produz em caracter de luz como que esboçado no vidro num fundo escuro. O segundo tipo usa a polarização para absorver luz onde existe campo eléctrico. Isto produz caracteres escuros num fundo cinza-prateado.

A maioria dos LCDs requer uma tensão de 2 ou 3 V entre o plano de fundo e um segmento para activar o segmento. Contudo, não se pode simplesmente ligar o plano de fundo à massa e conduzir os segmentos com as saídas de um descodificador TTL, como se fez no caso dos LEDs na Fig. 4.16. A razão para isto é que os LCDs irão deteriorar-se rápida e irreversivelmente se for aplicada uma tensão contínua (DC) superior a 50 mV entre o segmento e o plano de fundo. Para evitar uma construção de DC nos segmentos, os sinais de condução dos segmentos devem ser ondas quadradas com uma frequência entre 30 e 150 Hz. Mesmo que usem pulsos no descodificador TTL, não resultaria porque a tensão de saída baixa (0) dos dispositivos TTL é maior que 50 mV. Normalmente são usadas *gates* CMOS para conduzir LCDs. A Fig. 4.18a) mostra como é que duas *gates* CMOS podem ser ligadas para conduzir um segmento e um plano de fundo de um LCD. A Fig. 4.18b) mostra forma de ondas típicas de condução para o plano de fundo e para os segmentos *on* e *off*. O segmento *off* (que neste caso não é utilizado) recebe o mesmo sinal de condução que o plano de fundo. Nunca há qualquer tensão entre eles, portanto não é produzido qualquer campo eléctrico. A forma de onda para o segmento *on* está desfasada 180º do sinal de plano de fundo, de modo a que a tensão entre este segmento e o plano de fundo seja sempre +V. a lógica para isto é que só se tem de produzir dois sinais: uma onda quadrada a sua complementar. Para as *gates* de condução, a "sanduíche" segmento/plano de fundo aparenta ser algo parecido com um condensador de fugas. As *gates* CMOS podem, facilmente, fornecer a corrente necessária para carregar e descarregar esta pequena capacidade.

Os *displays* LCD mais antigos e/ou baratos ligam-se e desligam-se muito devagar para serem multiplexados do mesmo modo que os *displays* LED. A 0º C alguns LCDs requerem até 0,5 s para ligar ou desligar. Para fazer a interface com estes tipos de *displays* usa-se um dispositivo condutor não-multiplexado. LCDs mais novos e mais caros podem ligar-se e desligar-se mais rapidamente, sendo, por isso, multiplexados usando-se uma variedade de técnicas. A seguir vai-se mostrar como se faz a interface dum *display* LCD não-multiplexado com um microprocessador tal como o SDK-86.

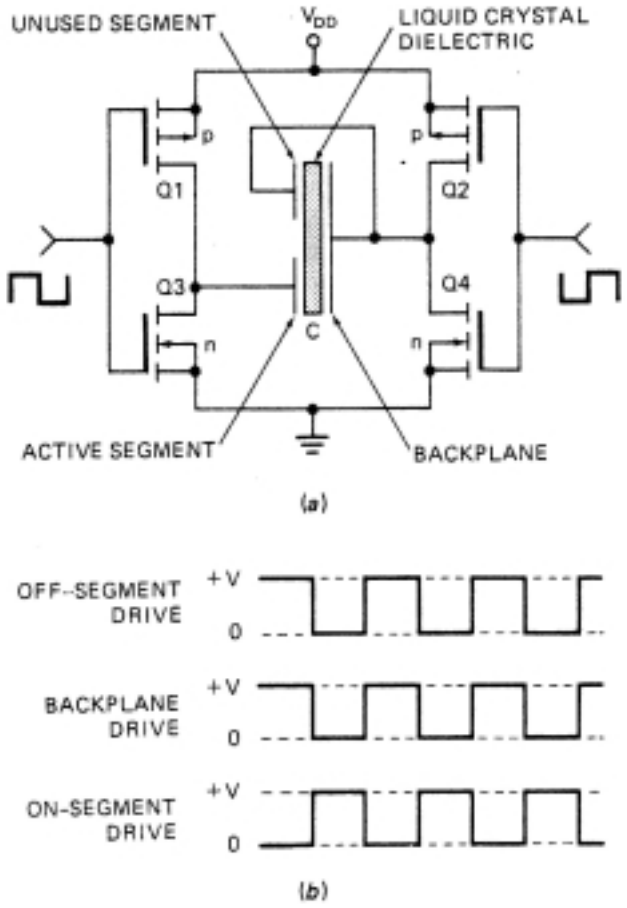



**Fig. 4.18**

**Interface entre um micro-computador e um *display* LCD não-multiplexado**

A Fig. 4.19 mostra como é que um Intersil ICM7211M pode ser ligado para conduzir um *display* LCD não-multiplexado de 7 segmentos tal como pode ser comprado de um loja qualquer de componentes electrónicos. As entradas do 7211M podem ser ligadas aos pinos de uma porta ou directamente aos barramentos do micro-computador tal como mostrado. Neste exemplo ligaram-se as entradas CS à saída Y2 da porta de descodificador do 74LS138. De acordo com a tabela de verdade deste dispositivo os dispositivos são endereçados como portas com endereço base FF10H. A linha de endereço do sistema SDK-86 A2 é ligada à entrada de selector de dígito (DS2) e a linha de sistema A1 é ligada à entrada DS1. Isto dá ao dígito 4 um endereço de sistema em FF10H. O dígito 3 será endereçado em FF12H e o dígito 1 em FF14H. As entradas de dados são ligadas às quatro linhas mais baixas do barramento de dados do SDK-86. A entrada do oscilador é deixada aberta.

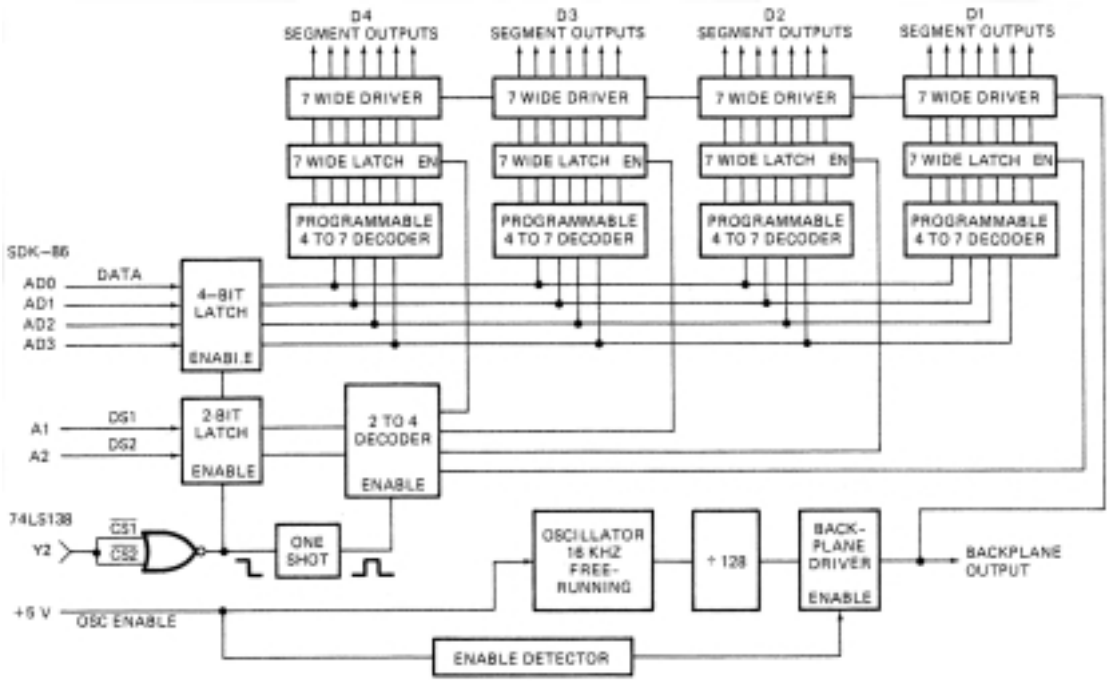



**Fig. 4.19**

Para mostrar um caracter num dos dígitos, basta simplesmente por o código hexadecimal de 4 bits para esse dígito nos 4 bits mais baixos do registo AL do SDK-86 e enviá-lo para o endereço de sistema desse dígito. O ICM7211M converte o código hexadecimal de 4 bits no código de 7 segmentos necessário para ser fixado nas saídas *latch* do dígito endereçado. Um oscilador interno gera automaticamente as formas de onda de condução mostradas na Fig. 4.18b).

Para se fazer a interface com *displays* LCD que possam ser multiplexados pode ser utilizado o Intel ICM7233.

# 5. Acesso directo à memória (DMA)

## *5.1 Transferência de informação por DMA*

Já foi visto um processo de transferir informação entre um periférico e o microprocessador: as interrupções. Outro processo já referido é a entrada/saída programada (*polling*). O segundo processo sincroniza o microprocessador ao periférico à custa da eficiência do processamento. Já o primeiro é mais natural – o processador só responde quando é chamado a intervir. Como resultado temos um *overhead* de *software* maior, resultando, geralmente, numa resposta mais rápida que o *polling* mas uma taxa de informação menor.

Um terceiro processo de transferir informação é chamado Acesso Directo à Memória (DMA). Neste caso o periférico comunica directamente com a memória. sem perturbar o conteúdo dos registos do processador. Um *polling* ou uma interrupção pode demorar vários micro-segundos a processada. Se houver várias transferências a executar, por exemplo cada 4 ou5 ms em média, a execução do programa principal será muito afectada. Por outro lado, há periféricos que trabalham a muito alta velocidade (Ex. discos, *floppy disks*) para os quais a entrada/saída programada ou por interrupção se revela um processo muito lento. Para estes casos a transferência por DMA revela-se a mais eficiente. Em contrapartida requer uma interface muito mais complicada.

A maior parte dos microprocessadores com capacidade de executar DMA tem uma entrada (HOLD, WAIT ou PAUSE) que é activada por um periférico que pretende utilizar a transferência por DMA. Quando activada coloca o processador num estado de "descanso", depois de este realizar a instrução que estava a executar, e faz a desactivação dos barramentos de endereço e de informação (por vezes também de algumas linhas de controlo) colocando-os no terceiro estado (alta impedância). Outros sinais de saída do microprocessador como HOLD ACKNOWLEDGE, informam do reconhecimento do pedido, permitindo aos periféricos iniciarem a transferência de informação. Este é um processo de realizar DMA por paragem completa do microprocessador durante a transferência e é o mais simples de executar, embora seja pouco eficiente, pois durante a troca de informação o processador não está a realizar qualquer actividade útil.

Outro processo de realizar DMA é designado por *cycle-stealling*. Este método baseia-se no seguinte: há normalmente dois ciclos por cada instrução: um de recolha (*fetch*) e outro de execução. Quando a linha de *cycle-steal* (ou de *hold*, etc.) é activada, o processador completa o ciclo de execução da presente instrução, permitindo de seguida um intervalo de um ciclo antes de iniciar a instrução seguinte. Durante esse intervalo a informação pode ser trocada directamente entre a memória e os periféricos.

Os passos normalmente necessários para executar uma transferência por DMA são:

1. Um periférico requisita um pedido de DMA
2. O processador informa que recebeu o pedido e desactiva os barramentos de endereço e informação
3. Lógica exterior endereça a memória
4. Troca de informação directamente entre os periféricos e a memória (Leitura ou Escrita)
5. Fim do DMA.

No passo 1 o periférico requisita uma transferência por DMA activando a entrada correspondente do processador. Este avisa a sua recepção depois de acabar o ciclo de instrução que estava a executar, desactivando de seguida os barramentos de endereço e informação. A maior parte dos processadores com facilidade de DMA usam *buffers* de saída com três estados para facilitar a sua desactivação. Durante este estado o processador permanece numa situação em que não realiza qualquer operação. Lógica exterior (controlador de DMA) toma conta dos barramentos de endereço e informação e fornece um endereço para o sistema de memória (passo 3). Dá-se, então, a transferência de informação entre a memória e o periférico seleccionado. No caso da transferência envolver um bloco de informação, o controlador deve fornecer o endereço inicial, incrementar os endereços de memória após cada transferência e manter um contador que conte o número de palavras transferidas. Após a transferência, o controlador termina o DMA desactivando a sinal aplicado ao microprocessador (HOLD, etc.) o que completa o passo 5. A Fig. 5.1 contém um fluxograma de uma transferência de DMA típica.

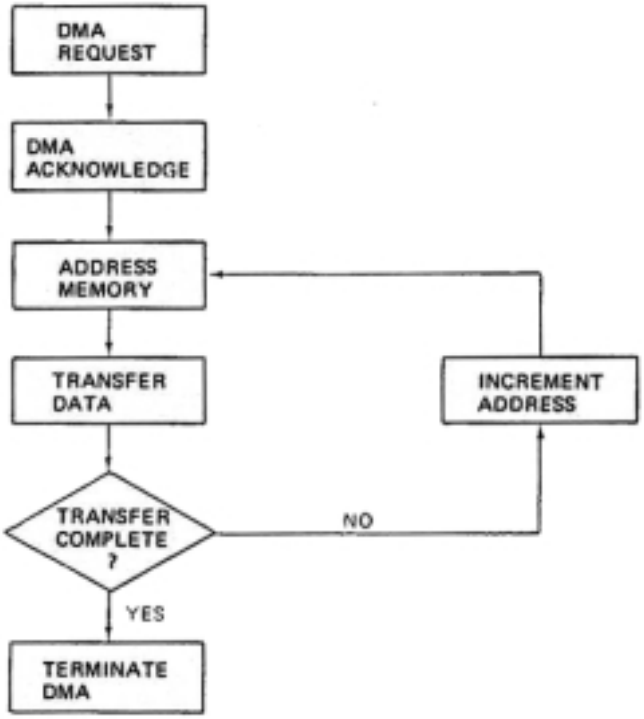



**Fig. 5.1**

Na Fig. 5.2 encontra-se um diagrama de blocos de um sistema de DMA.

É fornecido à lógica de DMA a informação necessária para especificar se se trata de uma operação de leitura ou escrita na memória. O registo de endereço é carregado com o endereço inicial, dando-se de seguida uma transferência entre o periférico e a memória. Após cada transferência o registo de endereço é incrementado para seleccionar o endereço seguinte. Quando a transferência tiver terminado é feito o reset do pedido de DMA, o que autoriza o processador a retornar as suas funções normais.

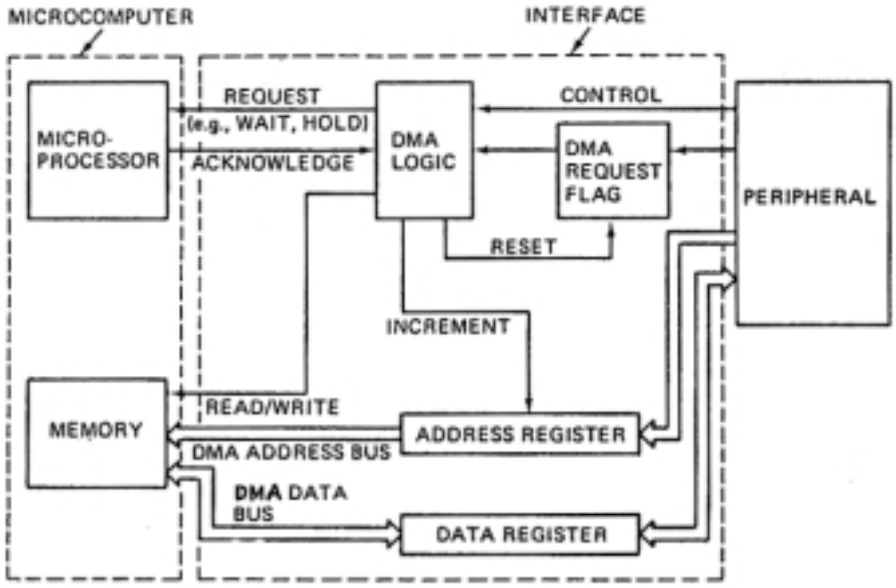



**Fig. 5.2**

A Fig. 5.3 pormenoriza a lógica de DMA necessária para efectuar as operações descritas.

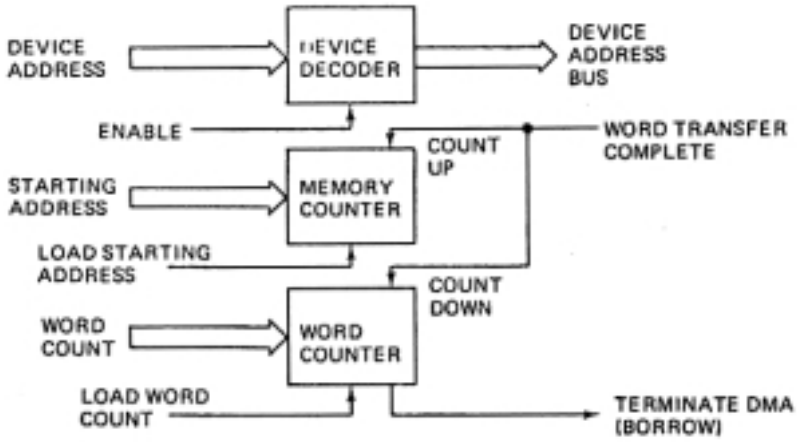



**Fig. 5.3**

O DEVICE ADDRESS activa o periférico apropriado através do descodificador a que liga. O endereço inicial (*starting address*) e o número de palavras a transferir (*word count*) são carregados nos contadores de memória e de palavra. A primeira palavra do bloco é então transferida, sendo o contador de memória incrementado pelo sinal COUNT UP e o contador de palavra decrementado pela entrada de COUNT DOWN. A transferência prossegue decrementando-se o contador de palavras até atingir zero, altura em que é gerado um sinal para terminar a operação de DMA. Nas transferências que envolvem mais que um bloco deve acrescentar-se a esta lógica um contador de bloco.

Existem duas classes de DMA. No DMA sequencial o controlador de DMA (DMAC) efectua uma operação de leitura mandando vir o byte de informação do

DMAC. De seguida é efectuada uma operação de escrita transferindo o byte de informação para a porta de entrada/saída. Também é possível a sequência oposta – ler um byte da porta I/O, escrever um byte na memória. Normalmente são necessários dois a quatro períodos de relógio para cada operação de leitura ou escrita.

O DMA simultâneo fornece as transferências mais rápidas. Com esta técnica as operações de leitura e de escrita são feitas ao mesmo tempo. Isto requer que MEMR e IOW (ou IOR e MEMW) estejam activas simultaneamente. Deste modo a informação não passa pelo DMAC mas vai directamente da memória para a porta I/O ou vice-versa. Como resultado temos uma melhoria de duas vezes na velocidade comparado com o DMA sequencial.

Em qualquer dos casos a transferência de informação é feita completamente em *hardware* envolvendo só o DMAC, o periférico e a memória principal. Como o CPU não está envolvido não há *overhead* de *software*.

Os pedidos de DMA tomam precedência sobre todas as outras actividades do barramento incluindo interrupções. De facto, não será reconhecida qualquer interrupção – mascarável ou não-mascarável – durante um pedido de DMA.

Existem quatro tipos de transferência por DMA:

1. Memória para periférico
2. Periférico para memória
3. Memória para memória
4. Periférico para periférico.

Normalmente é feita a interface do DMAC para o microprocessador como uma porta de I/O. Antes de ocorrer qualquer transferência de informação, o CPU deve programar o DMAC para o tipo de transferência que vai ocorrer, os endereços de origem e de destino e o número de bytes que vai ser transferido.

Considerando que o DMA sequencial e simultâneo são “classes” separadas, existem três modos de DMA existentes em cada classe. A Fig. 5.4 mostra estes três modos.

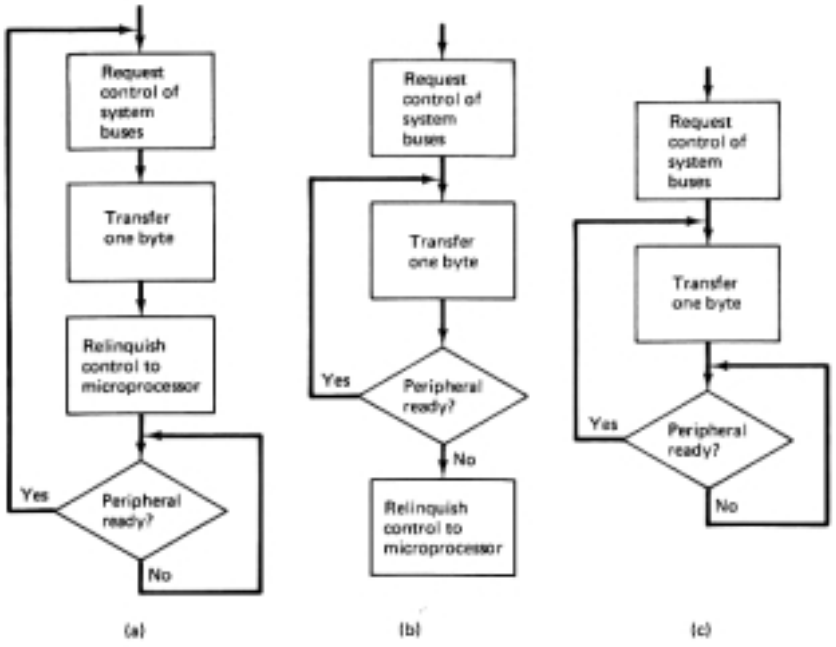



**Fig. 5.4**

No modo simples ou de único byte - a) - o DMAC após ter o controlo dos barramentos do sistema, transfere um único byte de informação. O controlo dos barramentos é depois abandonado até a *flag* READY do periférico estar novamente activa.

O modo de pedido ou *burst* – b) – é usado quando os periféricos têm *buffers* de alta velocidade. Após ter ganho o controlo dos barramentos, é transferida a informação até que a *flag* READY do periférico deixe de estar activa. O controlo dos barramentos é então abandonado para o CPU. Quando READY estiver de novamente activa, ocorre outra explosão de DMA. A vantagem desta técnica é que o *buffer* pode ser preenchido rapidamente pelo DMAC e depois esvaziado ao prazer do periférico.

Um terceiro modo de DMA é chamado modo contínuo ou de bloco – c). Este é semelhante ao modo de pedido excepto que neste caso não é abandonado o controlo dos barramentos até que seja transferido todo o bloco de informação. Esta técnica é muito eficaz com um periférico de alta velocidade que consiga acompanhar o DMAC. Periféricos lentos fazem com que haja longos períodos de inactividade dos barramentos enquanto o DMAC espera pela *flag* READY.

Como o pedido de DMA é amostrado no fim de cada ciclo de máquina (não ciclo de instrução), o tempo de resposta do DMAC não será maior que um ciclo de máquina mais um estado T.

O projecto de um controlador de DMA não é uma tarefa fácil e é provavelmente a maior desvantagem desta técnica. De facto o DMAC é um processador de transferência altamente especializado, competindo com a própria complexidade do microprocessador.

### *5.2 Controlador DMA*

O 8237 é um controlador de DMA que permite transferir informação entre a memória e uma porta de entrada/saída com velocidades da ordem de 1,6 MB/s.

Nem todas as aplicações de DMA envolvem periféricos de alta velocidade. Também se pode usar impressoras com *buffer* com interface de DMA. Sempre que a *flag* BUSY/READY da impressora indicar READY, o *buffer* da impressora pode ser preenchido usando DMA a alta velocidade, a transferência por DMA fica completa e o microprocessador retorna o processamento normal quando a impressora esvaziar o *buffer*.

A vantagem em usar um controlador de DMA (um DMAC) para esta aplicação é que o controlador pode ser programado para manter o registo de endereço inicial e o número de palavras sem intervenção do CPU. Se forem usadas interrupções, seria necessária uma rotina de serviço da interrupção e seria gasto tempo considerável de processamento no armazenamento e na recuperação de variáveis.

A desvantagem do DMA é que o processador tem que parar enquanto ocorre uma transferência por DMA. Isto não é uma verdadeira desvantagem quando a transferência ocorre em explosões (*bursts*) tal como no exemplo da impressora. Por outro lado, é oferecida uma melhor solução por parte das interrupções se a impressora não tiver um *buffer* e o DMA tiver de inserir estados de espera (WAIT) para se sincronizar com taxa de informação da impressora.

Outra desvantagem do DMA é que não é possível fazer processamento *a priori* nem *a posteriori* pelo DMAC. Se tal for necessário, terá de ser feito antes ou após a informação ser transferida.

No passado a maior limitação no uso de DMA foi a grande complexidade de concepção do DMAC. É aqui que entra o 8237. O 8237 fornece todas as características requeridas para transferências por DMA num só *chip* de 40 pinos e incorpora-se directamente na arquitectura de sistema de três estados. Entre as características do 8237 encontram-se as seguintes:

1. Existem quatro canais de DMA
2. Cada canal é capaz de transferir até 64K bytes.
3. Pode se escolher entre prioridades fixas ou rotativas
4. O *hardware* ou o *software* podem requerer transferência por DMA
5. Podem ser especificadas transferências de blocos de bytes ou de bytes individuais
6. São possíveis transferências de informação entre I/O e a memória e de memória para memória.

### *5.2.1 Interface do 8237*

A Fig. 5.5 representa o diagrama de blocos do 8237.

Existe um problema em grandes sistemas onde todas as linhas de dados no módulo do CPU têm *buffer*. É que apesar de os *buffers* "verem" altas impedâncias nas suas entradas (do microprocessador de três estados), normalmente eles irão converter isto em níveis de 1 lógico nas suas saídas, originando problemas de contenda do barramento com o DMAC. Por esta razão os módulos de CPU devem incluir um sinal que permita aos *buffers* de barramento ter três estados.

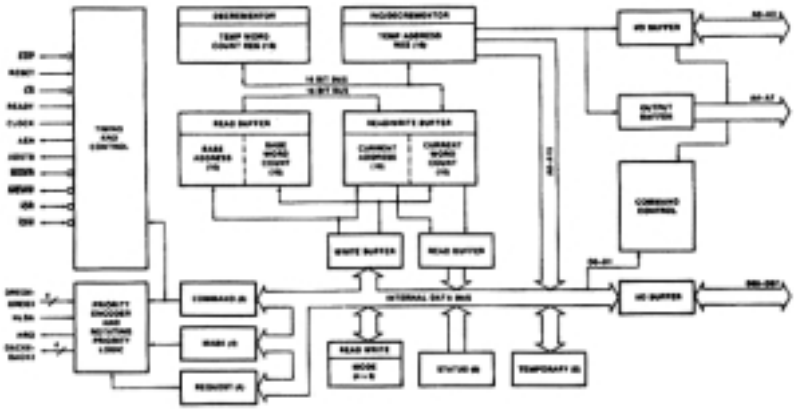


**Fig. 5.5**

A Fig. 5.6 mostra como é que se pode fazer a interface do 8237 com um módulo de CPU.

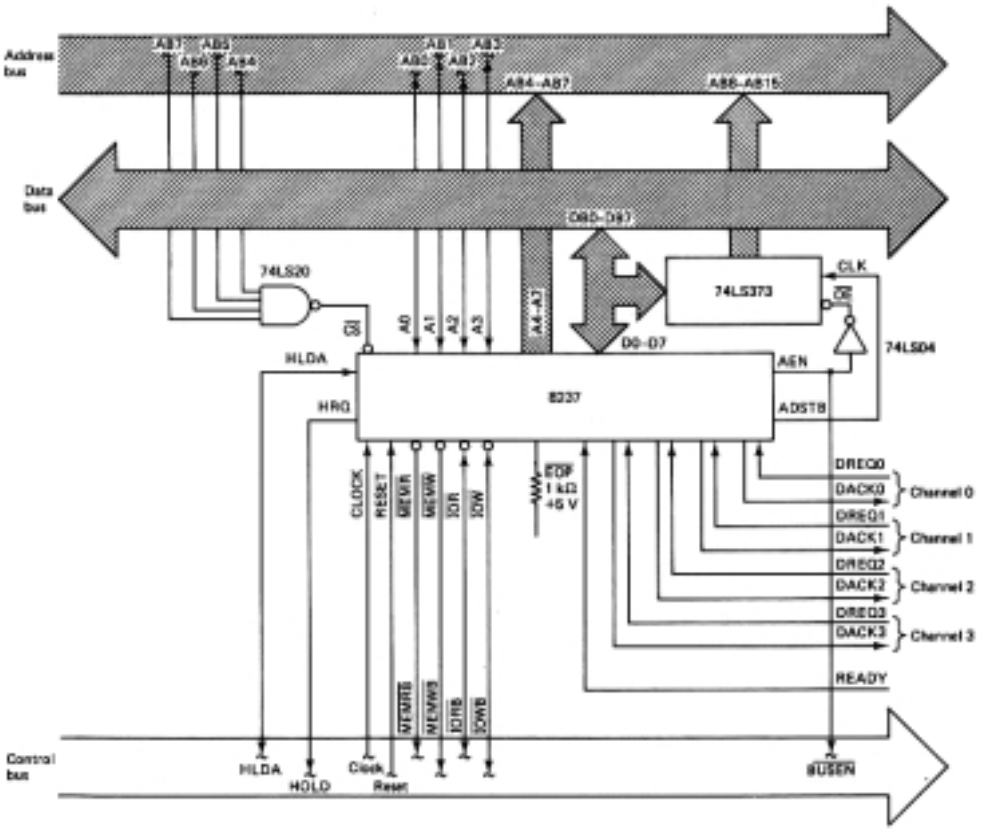



**Fig. 5.6**

É de reparar que é necessário fixar um endereço de alta ordem. Durante o tempo em que o ADSTB está a 1 e quando estiver a ocorrer uma transferência por DMA, D0-D7 vão manter os bits de ordem mais alta do endereço. O flanco descendente de ADSTB fixa este byte no *latch* 74LS373. Para o resto de transferência por DMA, D0-D7 funcionam como barramento bidireccional de dados.

De seguida vão ser enunciados os passos requeridos para efectuar uma transferência de informação com um periférico.

1. Antes da transferência ocorrer, deve se programar o DMAC com o endereço de memória, o número de bytes e o tipo de transferência. No caso do 8237, o DMAC aparenta ser 16 portas de entrada/saída consecutivas seleccionadas com A3-A0 e IOR e IOW. É de notar que o dispositivo não pode mapeado por memória; MEMR e MEMW são somente pinos de saída.
2. Agora pode ser pedida uma transferência por DMA num dos quatro canais por um periférico activando DREQ.
3. Se o 8237 estiver activado e o canal activado não estiver mascarado, o DMAC activa HRQ, pedindo um estado de espera (HOLD) do CPU.
4. O CPU responde completando o ciclo de instrução actual, colocando os seus barramentos em alta impedância e enviando HLDA.
5. O DMAC avisa o periférico deste reconhecimento enviando DACK para o periférico. Este sinal é normalmente usado para o *chip select* (CS) das portas de dados dos periféricos.
6. Agora o 8237 envia o endereço em D7-D0 e coloca AEN e ADSTB a 1. Este endereço representa a origem ou o destino da transferência de informação. O AEN activa as saídas de alta impedância do *latch* e também é usado para desactivar os barramentos do sistema através de BUSEN.
7. O ADSTB vai a 0, fazendo com que endereço de alta ordem seja fixado. D0-D7 mudam para se tornarem o barramento de informação e A0-A7 o barramento de endereço de baixa ordem.
8. Agora o DMAC controla os barramentos e são permitidos os três tipos de transferência de informação, tal como demonstrado na Fig. 5.7. Nesta figura, a carregado, está a transferência de um byte de informação da memória para um dispositivo de entrada/saída. Repare-se na condição não usual do controlo de barramento – estão activos dois sinais em simultâneo: MEMR e IOW. Apesar de isto nunca poder acontecer quando o CPU tem o controlo, é normal quando quem tem o controlo é o DMAC. A informação lida da RAM é directamente enviada para o dispositivo de entrada/saída; ela não vai pelo DMAC.
9. Só é necessário um canal do 8237 para a transferência. No caso especial de uma transferência de memória para memória, são necessários dois canais: um é programado com o endereço origem (o 8237 requer o canal 0) e o outro com o endereço de destino (o 8237 requer o canal 1). Neste caso não são necessários DREQ nem DACK.
10. A informação é transferida entre a RAM e o dispositivo de entrada/saída até que o contador de bytes seja zero (chamada contagem terminal). Também é possível parar a transferência com uma entrada externa EOP (*End Of Process*) no DMAC. Em qualquer caso, HRQ e AEN são removidos, desistindo do controlo do barramento para o processador.

Na Fig. 5.8 está um diagrama temporal indicando estas mesmas actividades.

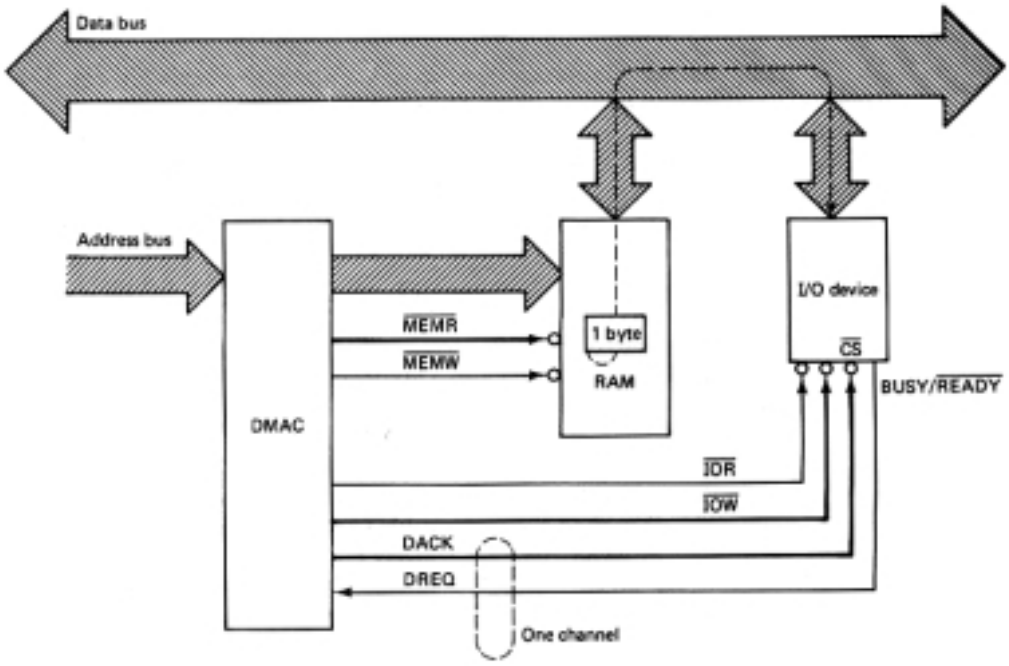



**Fig. 5.7**

### *5.2.2 Tempo de resposta e taxa de transferência*

Tal como mencionado anteriormente o CPU responde dentro de um ciclo de máquina dum pedido de HOLD. Isto significa que podem decorrer um máximo de seis T estados entre o pedido de DMA e o começo da transferência por DMA. Como termo de comparação temos que para responder a um pedido de interrupção são necessários 12 a 31 T estados.

Quando estiver para ocorrer uma transferência por DMA, normalmente o tempo de resposta não é o problema principal. Isto porque o processador de DMA está feito para transferir grandes blocos de informação, ao contrário das interrupções ou do *polling* em que o processador tem que monitorar uma *flag* de BUSY/READY e depois responder rapidamente com um byte de informação antes de ocorrer a próxima transacção.

O 8237 tem dois modos básicos de operação: em repouso (*idle*) ou activo (*active*). No estado de repouso pode ser programado pelo processador para um próximo estado activo durante o qual irá ocorrer a transferência por DMA. A Fig. 5.9 mostra as actividades que ocorrem durante cada um dos seis estados de relógio possíveis.

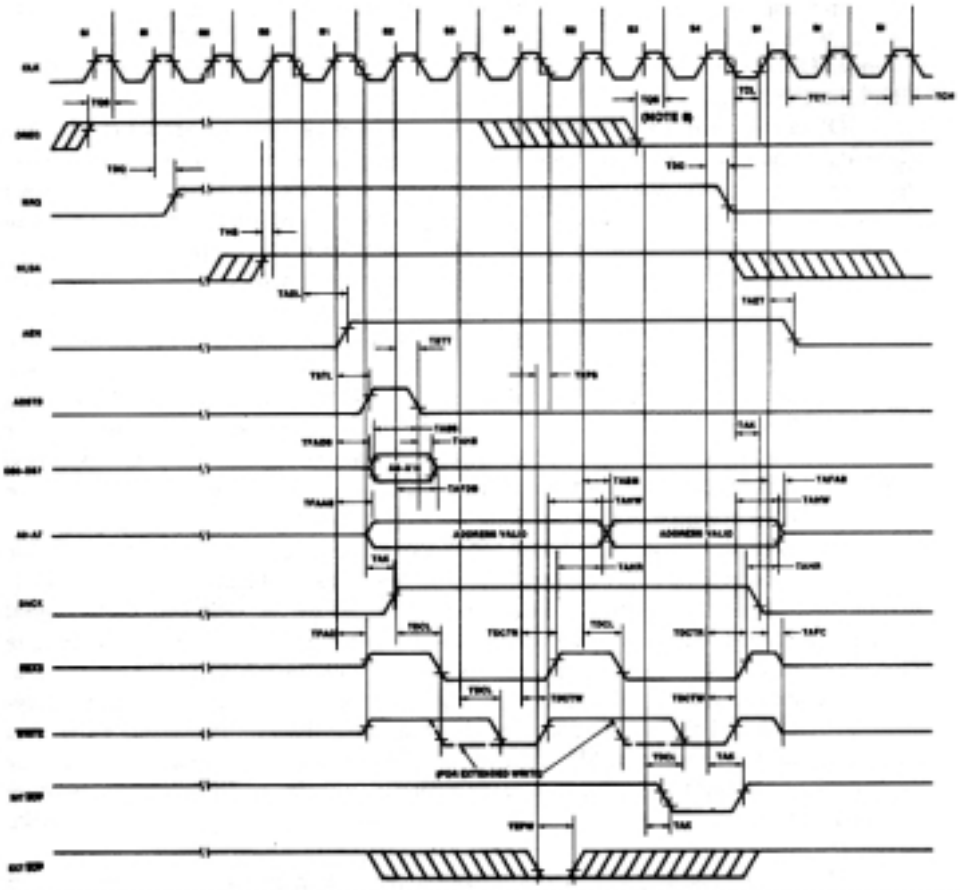


**Fig. 5.8**

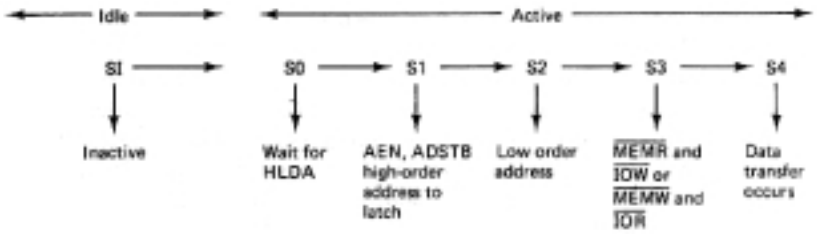



**Fig. 5.9**

Após ocorrer um DREQ, o DMAC passa do estado S1 para o estado S0, esperando HLDA do processador. No estado S1 é enviado o endereço de alta ordem. Note-se que este endereço só muda uma vez em cada 256 transferências de informação. Por esta razão, o 8237 apaga automaticamente o estado S1 excepto quando tem de mudar o endereço de alta ordem.

Durante S2 é enviado o endereço de baixa ordem e durante S3 aparecem os sinais de controlo. A transferência de informação propriamente dita ocorre durante o estado S4. Assumindo que o endereço de alta ordem não muda, serão necessários três estados de relógio (S2, S3 e S4) para uma transferência de informação.

A taxa de transferência para o 8237A é de 1,6 MB/s. Claro que quer o dispositivo de I/O quer a memória devem ser capazes de manter esta alta taxa de

transferência. Para memórias de semicondutores isto não constitui um problema. Contudo, o dispositivo de I/O pode ser outro assunto. Uma das aplicações mais comuns do DMA é fazer de interface com um *drive* de disco em série – um disco duro ou um *floppy disk*.

Considere-se uma *floppy disk* de dupla densidade de 5 ¼ polegadas que requer um novo byte de informação cada 32 μs. Para fazer interface com este tipo de periféricos deve ser usada a entrada READY do 8237. Esta linha é posta a 0 pelo controlador do periférico enquanto ele recebe (ou escreve) um novo byte de informação. A entrada READY do DMAC é semelhante à entrada READY do microprocessador e é amostrada durante S3. Se esta entrada for 0, os três barramentos "marcam tempo", segurando informação válida até READY ser novamente 1.

Apesar de a interface de *floppy disk* não o requerer, o 8237 é capaz de uma taxa de informação ainda maior se for usado tempo comprimido. Neste modo é suprimido o estado S3; só é usado para expandir o comprimento do pulso dos sinais de controlo IOR e IOW, permitindo transferências de informação só com os estados S2 e S4. Isto iria requerer dispositivos de memória com tempos de acesso inferiores a 400 ns para o 8237A - 5.

As transferências de memória para memória requerem o dobro do tempo das transferências entre memória e um dispositivo entrada/saída. Isto porque é necessário um conjunto de S estados para ler um byte de informação e outro para escrevê-lo, pois MEMR e MEMW não podem ser permitidas ser 0 simultaneamente como acontece com os sinais de controlo de memória e de entrada/saída. Além disso é necessário um registo temporário dentro do 8237 para guardar a informação antes de ocorrer o ciclo de escrita.

As transferências entre memórias são úteis quando se pretende transferir um bloco de informação grande ou quando um bloco de memória é para ser preenchido com um determinado caracter.

### *5.2.3 Programação do 8237*

Cada canal do 8237 tem um registo de 16 bits do endereço actual e um registo de 16 bits da palavra actual. O registo do endereço actual guarda o endereço de memória para a próxima transferência por DMA. O registo da palavra actual age como um contador em modo decrescente de 16 bits e é programado com o número total de bytes a ser transferido menos um. A contagem terminal ocorre quando este registo passa de 0000H para FFFFH.

Cada um destes registos é suportado por registo de endereço base e por um registo de palavra base que contêm os valores iniciais quando programados. Isto permite que ocorra uma inicialização automática de uma sequência (se programada) quando ocorrer a contagem terminal (ou EOP).

A Tabela 5.1 indica os códigos dos sinais de controlo necessários para aceder aos registos de cada canal. Repare-se que os registos do endereço actual e da contagem de palavra só podem ser lidos, não podem ser escritos. Como o endereço base e a contagem de palavras provavelmente requerem 16 bits, é usado um *flip-flop* interno para encaminhar estes dois bytes para o DMAC. Deste modo a primeira escrita para os registos dos endereços base e actual é interpretada como o endereço de baixa ordem e a segunda escrita como o endereço de alta ordem. É usado um esquema semelhante com o registo de contagem de palavra.

| Canal | Registo | Operação | Sinais | | | | | | | *Flip-flop* interno | Barramento de dados DB0-DB7 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | CS | IOR | IOW | A3 | A2 | A1 | A0 | | |
| 0 | Endereço base e actual | E | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | A0-A7 |
| | | | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | A8-A15 |
| | Endereço actual | L | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | A0-A7 |
| | | | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | A8-A15 |
| | Contagem de palavra base e actual | E | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | W0-W7 |
| | | | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | W8-W15 |
| | Contagem de palavra actual | L | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | W0-W7 |
| | | | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 1 | W8-W15 |
| 1 | Endereço base e actual | E | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | A0-A7 |
| | | | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 1 | A8-A15 |
| | Endereço actual | L | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | A0-A7 |
| | | | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | A8-A15 |
| | Contagem de palavra base e actual | E | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 0 | W0-W7 |
| | | | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | W8-W15 |
| | Contagem de palavra actual | L | 0 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | W0-W7 |
| | | | 0 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | W8-W15 |
| 2 | Endereço base e actual | E | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | A0-A7 |
| | | | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | A8-A15 |
| | Endereço actual | L | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | A0-A7 |
| | | | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | A8-A15 |
| | Contagem de palavra base e actual | E | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | W0-W7 |
| | | | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 1 | W8-W15 |
| | Contagem de palavra actual | L | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | W0-W7 |
| | | | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 1 | W8-W15 |
| 3 | Endereço base e actual | E | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | A0-A7 |
| | | | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 1 | A8-A15 |
| | Endereço actual | L | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | A0-A7 |
| | | | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | A8-A15 |
| | Contagem de palavra base e actual | E | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | W0-W7 |
| | | | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | W8-W15 |
| | Contagem de palavra actual | L | 0 | 0 | 1 | 0 | 1 | 1 | 1 | 0 | W0-W7 |
| | | | 0 | 0 | 1 | 0 | 1 | 1 | 1 | 1 | W8-W15 |

**Tabela 5.1**

No 8237 também deve ser programado o modo de transferência por DMA. Existem quatro escolhas (os modos 1 a 3 já foram ilustrados na Fig. 5.4):

1. *Transferência simples*: é transferido um único byte, a contagem de palavras é decrementada e o registo de endereço actual é incrementado ou decrementado conforme programado. DREQ deve ser mantido activo até se receber DACK. Se DREQ estiver activo durante a transferência, o controlo será devolvido ao microprocessador por um ciclo de máquina antes de iniciar a próxima transferência.

2. *Transferência de bloco*: a informação é transferida até que ocorra TC (contagem terminal) ou EOP. DREQ deve ser mantido activo até que ocorra DACK.
3. *Transferência por pedido*: a informação é transferida enquanto DRQE estiver activo. Quando DREQ ficar inactivo, o endereço actual e a contagem de palavra não mudam, permitindo ao dispositivo de I/O retomar a transferência onde a deixou. TC ou EOP terminarão o processo.
4. *Modo em cascata*: neste modo um 8237 age como Mestre e outro como Escravo. Cada um dos escravos é programado para um dos três modos de transferência descritos acima. Cada escravo fornece o endereço de memória para a transferência, mas o mestre dá prioridade aos pedidos de HOLD dos escravos.

Adicionalmente aos quatro modos listados, pode ser especificada uma transferência de memória para memória. Este modo é limitado a transferências de blocos e o canal 0 deve ser usado para a origem e o canal 1 para o destino.

São usados quatro registos para programar o modo de transferência e para seleccionar várias características de operação do 8237. A descrição dos seus bits está no Apêndice V. Cada um destes registos só permite ser escrito.

1. *Registo de comando*: este registo controla a operação do 8237. Repare-se que o *chip* pode ser desactivado fazendo com que o bit D2 seja 1. Se for seleccionada a rotação de prioridades, o último DREQ atendido irá ter a prioridade mais baixa.
2. *Registo de modo*: a porção de acesso à memória da transferência por DMA pode ser programado para uma operação de leitura, escrita ou verificação. Quando programação para verificar transferências, o 8237 envia endereços mas não os sinais de controlo.
3. *Registo de pedidos*: este registo permite ao *software* iniciar a transferência por DMA em vez do DREQ. Quando programado para transferências de memória para memória, activa o bit de pedido para o canal 0 começa a transferência. Os pedidos de *software* não são mascaráveis.
4. *Registo de máscaras*: este registo é usado para mascarar o DREQ para cada canal. Limpar o bit de máscara activa aquele canal para pedidos de DMA. Os bits de máscara podem ser limpos individualmente ou colectivamente como mostrado na alínea d) do Apêndice V.

Há dois registos que só podem ser lidos: são o registo de estados e o registo temporário, mostrados na Fig. 5.10 (a) e b), respectivamente).

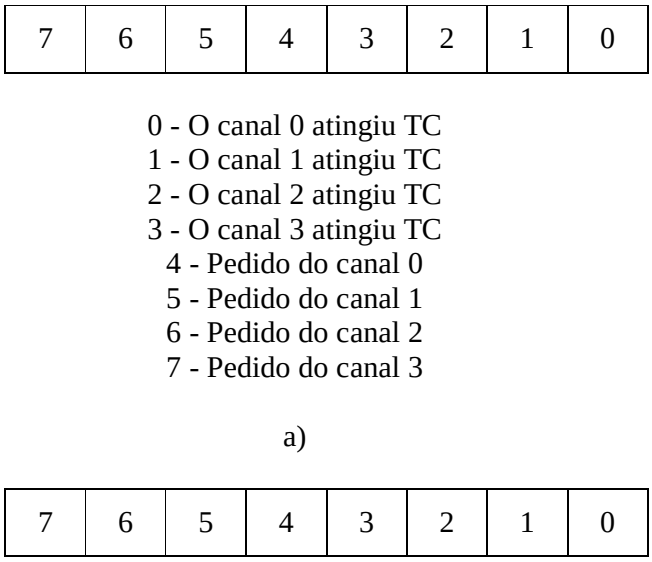



**Fig. 5.10**

Três portas do 8237 não são realmente portas mas usam o pulso de escolha de dispositivo criado por uma instrução de saída (OUT).

1. *Limpar primeiro/último flip-flop*: isto faz o reset do *flip-flop* interno para as portas 0-7 (ver Tabela 5.2). O próximo endereço de memória irá deste modo ser o byte de baixa ordem para o endereço de memória ou para a contagem de palavra.
2. *Limpar mestre*: o 8237 entra no estado de repouso com os registos de comando, estado, pedidos e temporários e o *flip-flop* interno todos em reset e com todos os bits do registo de máscaras em set.
3. *Limpar registo de máscaras*: este comando activa todos os quatro canais a aceitar pedidos de DMA nas suas entradas DREQ limpando todos os bits no registo de máscaras.

A Tabela 5.2 lista a funções de leitura e escrita associadas com cada um dos 16 endereços (base) de porta do DMAC. Utilizando o descodificador da Fig. 5.6, o 8237 requer as portas entre F0H e FFH (as portas FC, FD e FE são controladas pelos pulsos de selecção de dispositivo e não transferem informação).

| A3-A0 | IOR | IOW | Função |
|---|---|---|---|
| 0 | 0 | 1 | Ler endereço actual – Canal 0 |
| 0 | 1 | 0 | Escrever endereço actual – Canal 0 |
| 1 | 0 | 1 | Ler contagem de palavra actual – Canal 0 |
| 1 | 1 | 0 | Escrever contagem de palavra actual – Canal 0 |
| 2 | 0 | 1 | Ler endereço actual – Canal 1 |
| 2 | 1 | 0 | Escrever endereço actual – Canal 1 |
| 3 | 0 | 1 | Ler contagem de palavra actual – Canal 1 |
| 3 | 1 | 0 | Escrever contagem de palavra actual – Canal 1 |
| 4 | 0 | 1 | Ler endereço actual – Canal 2 |
| 4 | 1 | 0 | Escrever endereço actual – Canal 2 |
| 5 | 0 | 1 | Ler contagem de palavra actual – Canal 2 |
| 5 | 1 | 0 | Escrever contagem de palavra actual – Canal 2 |
| 6 | 0 | 1 | Ler endereço actual – Canal 3 |
| 6 | 1 | 0 | Escrever endereço actual – Canal 3 |
| 7 | 0 | 1 | Ler contagem de palavra actual – Canal 3 |
| 7 | 1 | 0 | Escrever contagem de palavra actual – Canal 3 |
| 8 | 0 | 1 | Ler registo de estado |
| 8 | 1 | 0 | Escrever registo de comando |
| 9 | 0 | 1 | Ilegal |
| 9 | 1 | 0 | Escrever registo de pedidos |
| A | 0 | 1 | Ilegal |
| A | 1 | 0 | Escrever um único bit do registo de máscaras |
| B | 0 | 1 | Ilegal |
| B | 1 | 0 | Escrever registo de modo |
| C | 0 | 1 | Ilegal |
| C | 1 | 0 | Limpar primeiro/último *flip-flop* (DSP) |
| D | 0 | 1 | Ler registo temporário |
| D | 1 | 0 | Limpar mestre (DSP) |
| E | 0 | 1 | Ilegal |
| E | 1 | 0 | Limpar registo de máscaras (DSP) |
| F | 0 | 1 | Ilegal |
| F | 1 | 0 | Escrever todos os bits do registo de máscaras |

**Tabela 5.2**

A Fig. 5.11 é o fluxograma dos passos necessários para inicializar o 8237 e mostra as acções tomadas pelo DMAC uma vez programado. Repare-se que é aconselhável desactivar o DMAC até se completar a inicialização se um DREQ for recebido durante este tempo.

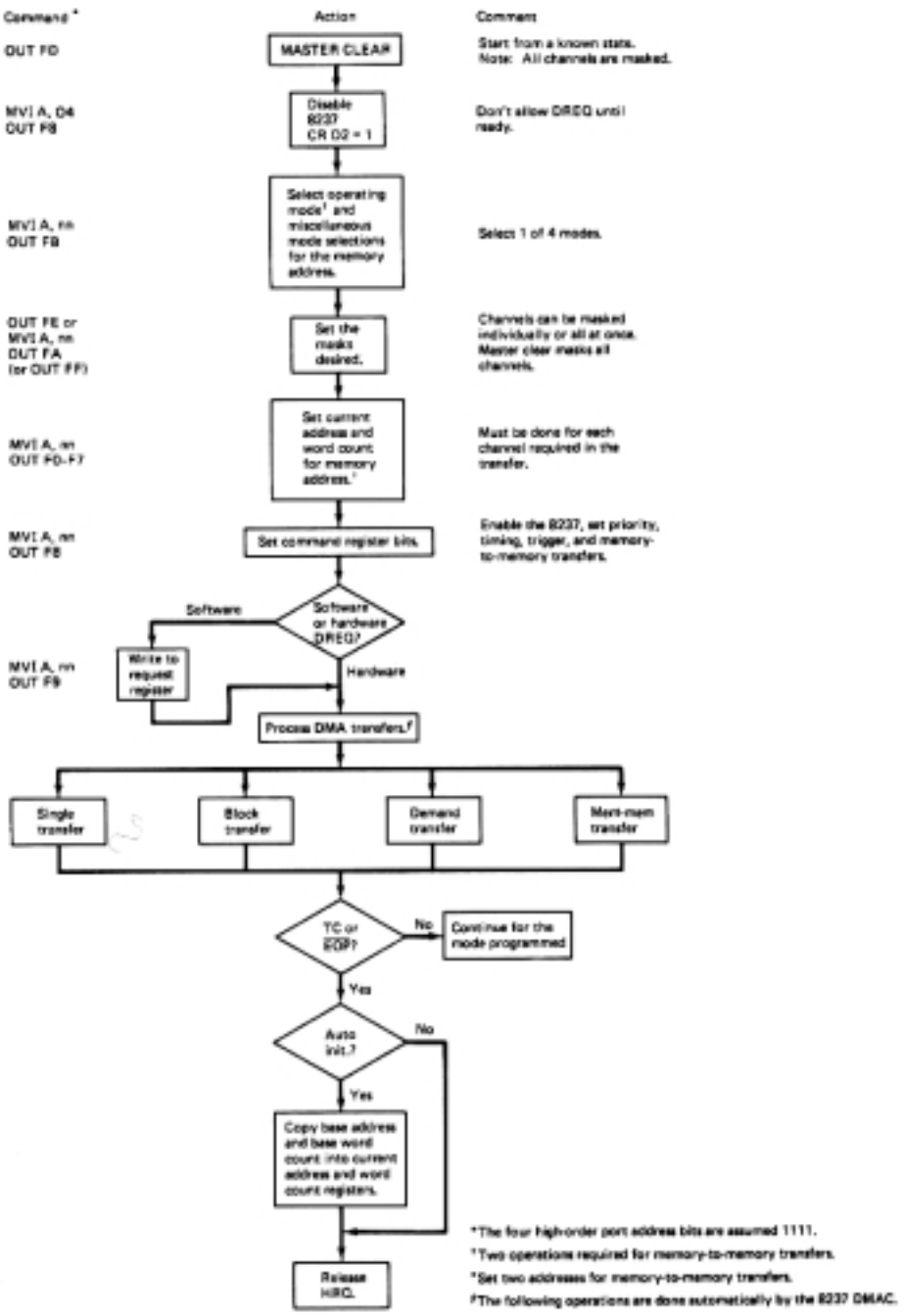

Command *
Action
Comment
OUT FD
MASTER CLEAR
Start from a known state.
Note: All channels are masked.
MVI A, D4
OUT F8
Disable 8237 CR D2 = 1
Don't allow DREQ until ready.
MVI A, nn
OUT F8
Select operating mode† and miscellaneous mode selections for the memory address.
Select 1 of 4 modes.
OUT FE or
MVI A, nn
OUT FA
(or OUT FF)
Set the masks desired.
Channels can be masked individually or all at once. Master clear masks all channels.
MVI A, nn
OUT F0-F7
Set current address and word count for memory address.‡
Must be done for each channel required in the transfer.
MVI A, nn
OUT F8
Set command register bits.
Enable the 8237, set priority, timing, trigger, and memory-to-memory transfers.
Software
Software or hardware DREQ?
MVI A, nn
OUT F9
Write to request register
Hardware
Process DMA transfers.§
Single transfer
Block transfer
Demand transfer
Mem-mem transfer
TC or EOP?
No
Continue for the mode programmed
Yes
Auto init.?
No
Yes
Copy base address and base word count into current address and word count registers.
Release HRQ.
*The four high-order port address bits are assumed 1111.
†Two operations required for memory-to-memory transfers.
‡Set two addresses for memory-to-memory transfers.
§The following operations are done automatically by the 8237 DMAC.

**Fig. 5.11**

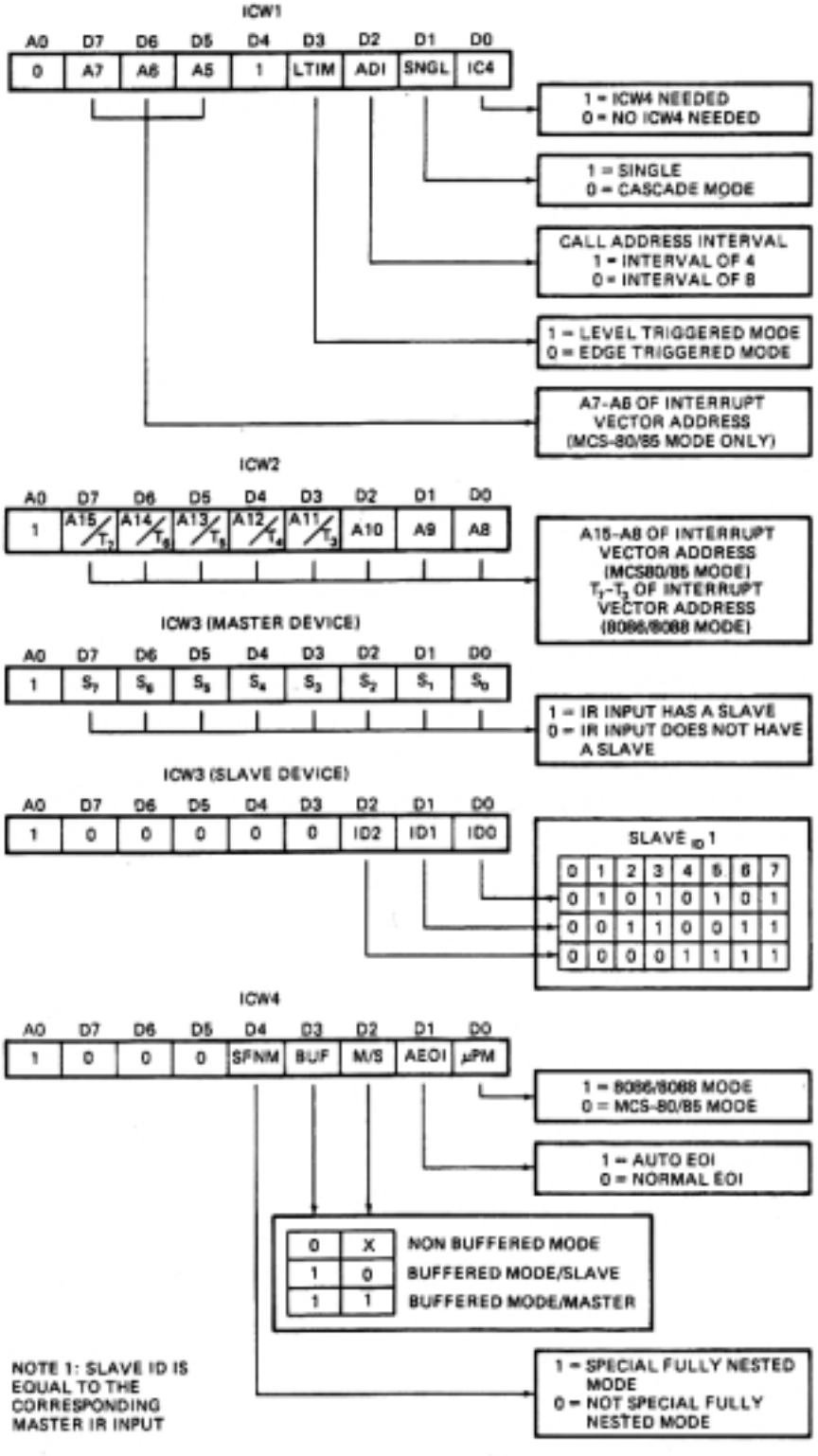
## 6. Bibliografia


- **Computer Organization and Architecture**
  *Designing For Performance*
  William Stallings
  Prentice Hall International Editions

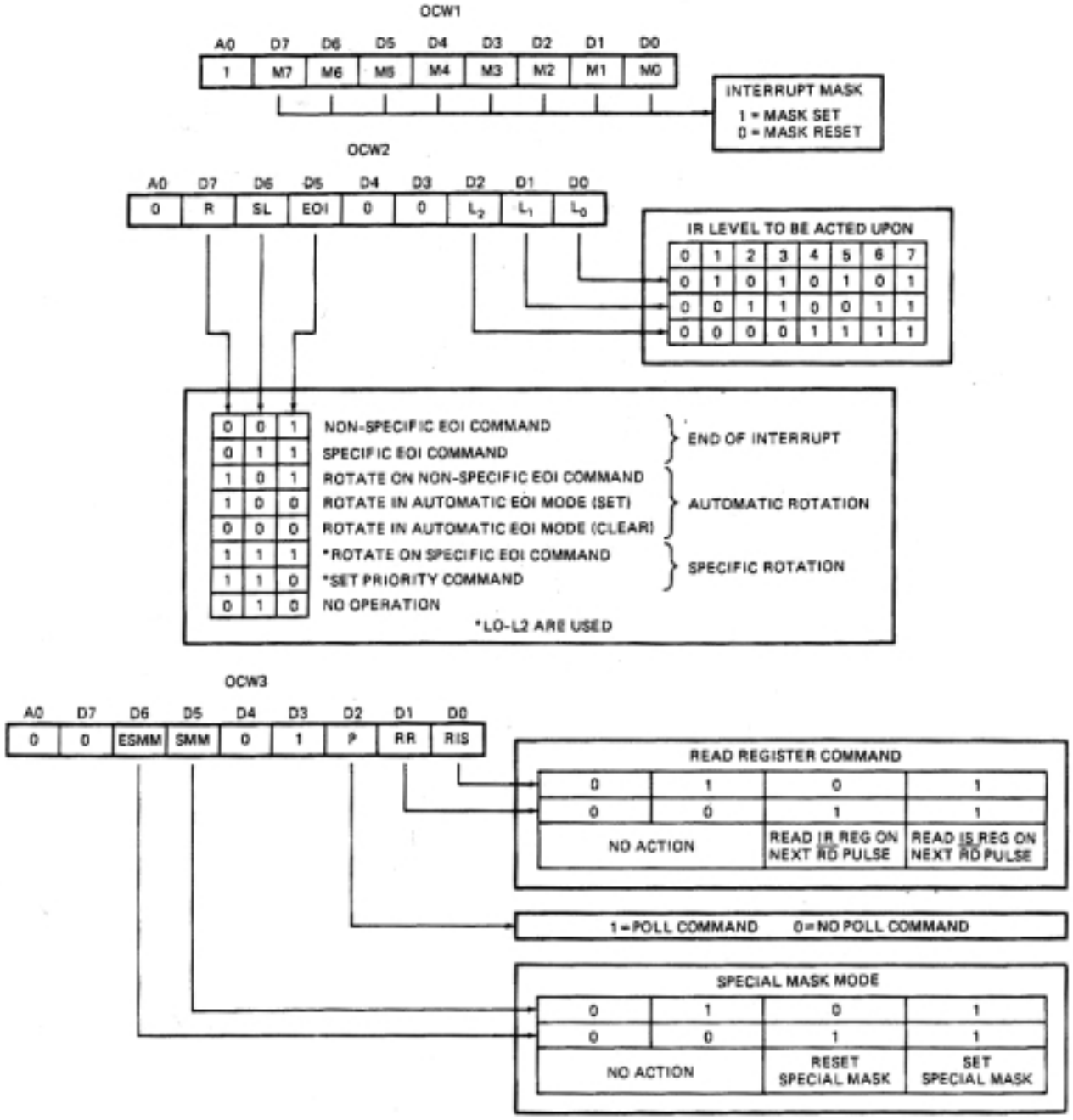
- **Microcomputers and Microprocessors**
  *The 8080, 8085, and Z-80 programming, interfacing and troubleshooting*
  John Uffenbeck
  Prentice Hall International Editions

- **Microprocessors and Interfacing**
  *Programming and Hardware*
  Douglas V. Hall
  McGraw-Hill International Editions

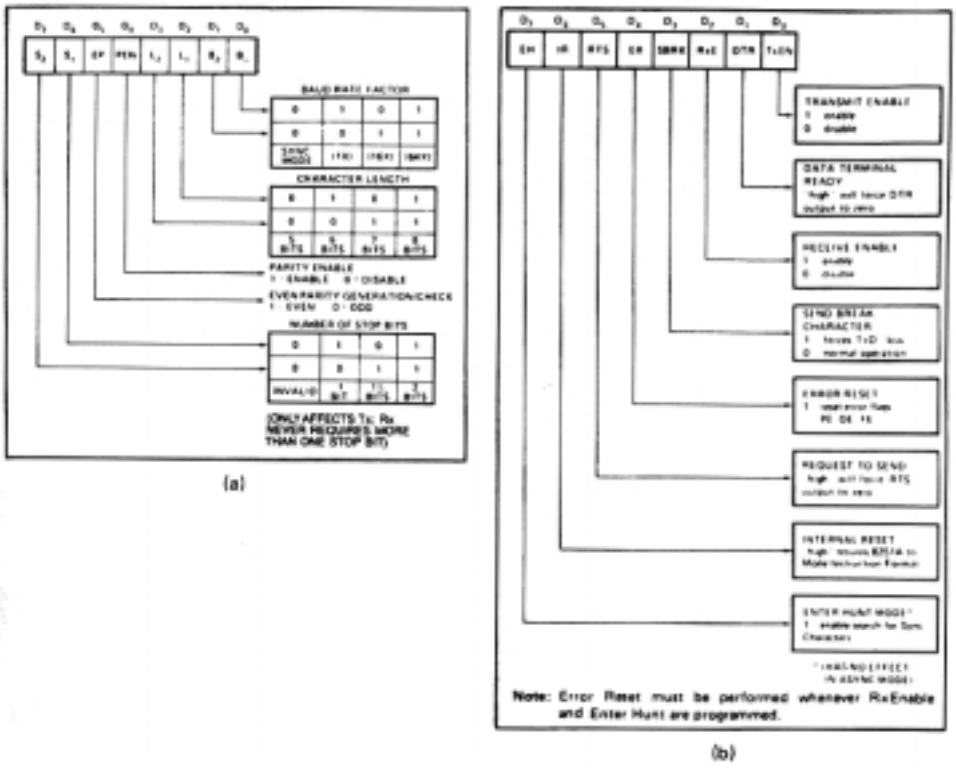
## Apêndice I


ICW1
A0 D7 D6 D5 D4 D3 D2 D1 D0
0 A7 A6 A5 1 LTIM ADI SNGL IC4
1 = ICW4 NEEDED
0 = NO ICW4 NEEDED
1 = SINGLE
0 = CASCADE MODE
CALL ADDRESS INTERVAL
1 = INTERVAL OF 4
0 = INTERVAL OF 8
1 = LEVEL TRIGGERED MODE
0 = EDGE TRIGGERED MODE
A7-A6 OF INTERRUPT VECTOR ADDRESS (MCS-80/85 MODE ONLY)
ICW2
A0 D7 D6 D5 D4 D3 D2 D1 D0
1 A15/T7 A14/T6 A13/T5 A12/T4 A11/T3 A10 A9 A8
A15-A8 OF INTERRUPT VECTOR ADDRESS (MCS80/85 MODE)
T7-T3 OF INTERRUPT VECTOR ADDRESS (8086/8088 MODE)
ICW3 (MASTER DEVICE)
A0 D7 D6 D5 D4 D3 D2 D1 D0
1 S7 S6 S5 S4 S3 S2 S1 S0
1 = IR INPUT HAS A SLAVE
0 = IR INPUT DOES NOT HAVE A SLAVE
ICW3 (SLAVE DEVICE)
A0 D7 D6 D5 D4 D3 D2 D1 D0
1 0 0 0 0 0 ID2 ID1 ID0
SLAVE ID 1
0 1 2 3 4 5 6 7
0 1 0 1 0 1 0 1
0 0 1 1 0 0 1 1
0 0 0 0 1 1 1 1
ICW4
A0 D7 D6 D5 D4 D3 D2 D1 D0
1 0 0 0 SFNM BUF M/S AEOI µPM
1 = 8086/8088 MODE
0 = MCS-80/85 MODE
1 = AUTO EOI
0 = NORMAL EOI
0 X NON BUFFERED MODE
1 0 BUFFERED MODE/SLAVE
1 1 BUFFERED MODE/MASTER
NOTE 1: SLAVE ID IS EQUAL TO THE CORRESPONDING MASTER IR INPUT
1 = SPECIAL FULLY NESTED MODE
0 = NOT SPECIAL FULLY NESTED MODE
(a)
ICW1
ICW2
IN CASCADE MODE
NO (SNGL=1)
YES (SNGL=0)
ICW3
IS ICW4 NEEDED
NO (IC4=0)
YES (IC4=1)
ICW4
READY TO ACCEPT INTERRUPT REQUESTS
(b)

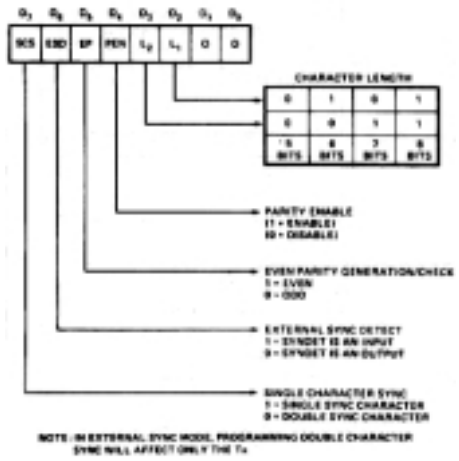
## Apêndice II


OCW1
A0 D7 D6 D5 D4 D3 D2 D1 D0
1 M7 M6 M5 M4 M3 M2 M1 M0
INTERRUPT MASK
1 = MASK SET
0 = MASK RESET
OCW2
A0 D7 D6 D5 D4 D3 D2 D1 D0
0 R SL EOI 0 0 L2 L1 L0
IR LEVEL TO BE ACTED UPON
0 1 2 3 4 5 6 7
0 1 0 1 0 1 0 1
0 0 1 1 0 0 1 1
0 0 0 0 1 1 1 1
0 0 1 NON-SPECIFIC EOI COMMAND
0 1 1 SPECIFIC EOI COMMAND
1 0 1 ROTATE ON NON-SPECIFIC EOI COMMAND
1 0 0 ROTATE IN AUTOMATIC EOI MODE (SET)
0 0 0 ROTATE IN AUTOMATIC EOI MODE (CLEAR)
1 1 1 *ROTATE ON SPECIFIC EOI COMMAND
1 1 0 *SET PRIORITY COMMAND
0 1 0 NO OPERATION
END OF INTERRUPT
AUTOMATIC ROTATION
SPECIFIC ROTATION
*L0-L2 ARE USED
OCW3
A0 D7 D6 D5 D4 D3 D2 D1 D0
0 0 ESMM SMM 0 1 P RR RIS
READ REGISTER COMMAND
0 1 0 1
0 0 1 1
NO ACTION
READ IR REG ON NEXT RD PULSE
READ IS REG ON NEXT RD PULSE
1 = POLL COMMAND 0 = NO POLL COMMAND
SPECIAL MASK MODE
0 1 0 1
0 0 1 1
NO ACTION
RESET SPECIAL MASK
SET SPECIAL MASK

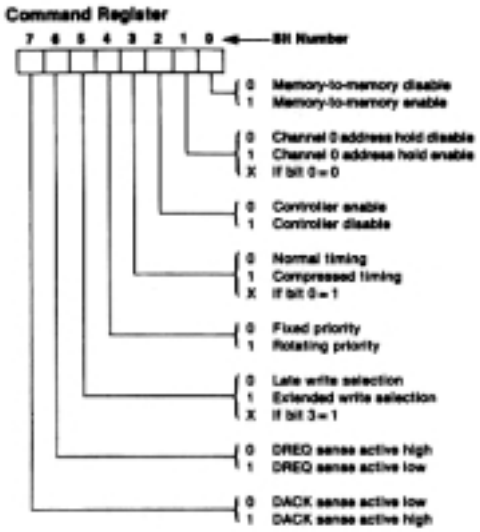
## Apêndice III



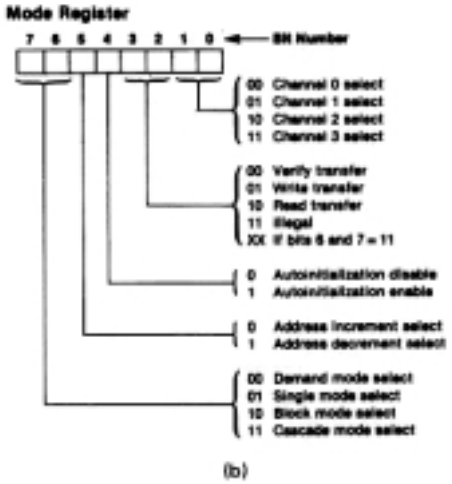
## Apêndice IV



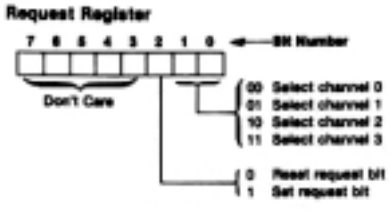
# Apêndice V



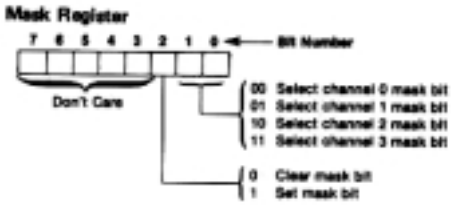

(a)




(b)

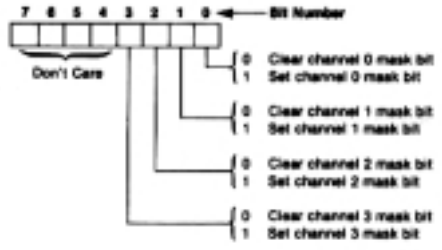



(c)




(d)

All four bits of the Mask register may also be written with a single command.




(e)