

### 3. O MICROPROCESSADOR Z80

O microprocessador Z80 nasceu de uma cisão no seio da Intel, através da qual algumas das pessoas que estiveram ligadas à concepção do 8080 decidiram criar a sua própria empresa e avançar com um projecto que conduziu ao microprocessador Z80, compatível com o 8080, mas representando ao mesmo tempo um enorme salto em frente, tanto na arquitectura como no conjunto de instruções. A empresa que fundaram recebeu o nome de Zilog e em 1976 lançou um dos seus produtos mais bem sucedidos, precisamente o microprocessador Z80. Tão bem sucedido que continua ainda a ser fabricado, quer na versão original quer em variantes que dela descenderam, 20 anos depois de ter sido introduzido.

Para além de possuir um conjunto de instruções largamente mais poderoso do que qualquer dos seus contemporâneos, o Z80 apresenta ainda a vantagem de possuir um modelo de programação e uma arquitectura interna que se aproximam bastante da complexidade ideal para apoiar um curso de introdução aos microprocessadores, razão pela qual continua a ser largamente usado neste contexto e que levou à sua adopção nestas páginas.

#### 3.1 ARQUITECTURA DO MICROPROCESSADOR Z80

A arquitectura do Z80 é frequentemente representada através do diagrama de blocos que se ilustra na figura 3.1 (que é igual ao usado na própria folha de características do componente), embora este diagrama possa com verdade aplicar-se à quase totalidade dos outros microprocessadores de uso genérico que possamos considerar. Refira-se no entanto que a superficialidade desta representação nos serve por enquanto, já que permite um enquadramento simples para a introdução dos aspectos de detalhe que irá tendo lugar nas secções seguintes.

O registo de instrução armazena o código de instrução extraído da memória, enquanto decorre a respectiva execução, proporcionando à unidade de decodificação e controlo a informação necessária para a geração dos sinais de controlo internos. Todos os registos do CPU (unidade de processamento central, *central processing unit*) estão incluídos no bloco com este nome (incluindo o apontador de programa ou *program counter*, PC), realizando-se na ALU todas as

operações lógicas ou aritméticas. Os blocos responsáveis pelo controlo dos barramentos contêm *buffers* com terceiro estado que permitem a colocação das respectivas saídas em alta impedância, quando tal for necessário.

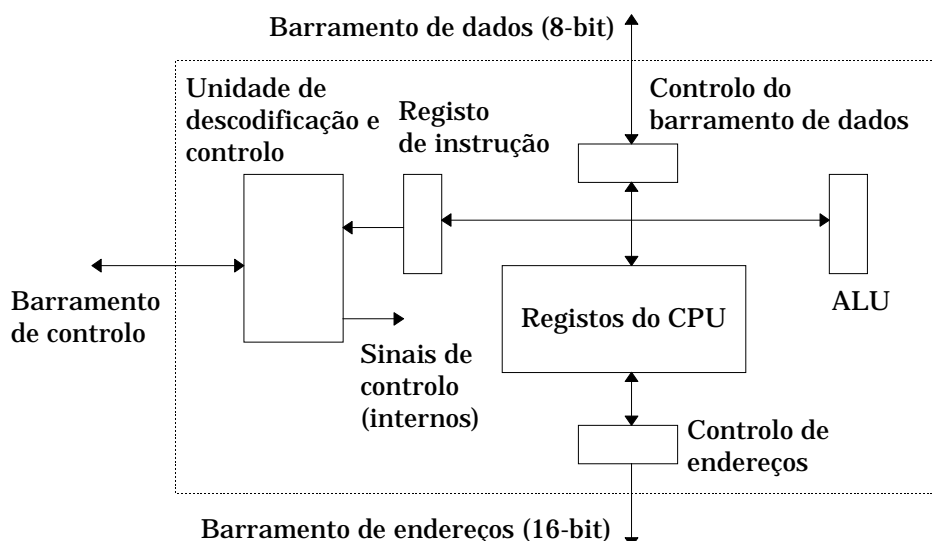


Figura 3.1: Arquitectura simplificada do microprocessador Z80.

## 3.2 INTERFACE COM O EXTERIOR

A apresentação do interface com o exterior passa essencialmente por dois aspectos, sendo o primeiro a configuração dos pinos do componente e a sua organização nos três tipos de barramentos, e o segundo a evolução temporal dos sinais principais destes três barramentos, para os vários tipos de acesso possíveis. Repare-se que este assunto precede a apresentação do modelo de programação do componente, já que alguns dos aspectos a introduzir nessa secção serão melhor compreendidos se existir já algum conhecimento sobre os mecanismos que regem a sua ligação ao exterior.

### 3.2.1 Barramentos e configuração de pinos

Os 40 pinos do Z80, com a excepção dos 2 pinos de alimentação, distribuem-se pelos três barramentos já conhecidos:

1. O *barramento de endereços* dispõe de 16 pinos de saída (A[0:15]), todos com terceiro estado, de forma a permitir que outros dispositivos tomem conta deste barramento (por exemplo, quando se pretende efectuar um acesso directo à memória).

2. O *barramento de dados* inclui 8 pinos bidireccionais, também com terceiro estado, através dos quais se processa toda a troca de informação com os dispositivos de memória e periféricos.
3. O *barramento de controlo* é o que inclui o conjunto de sinais que são específicos deste microprocessador, e que na figura 3.2 se encontram agrupados nos sinais de controlo do sistema, de controlo do CPU e de controlo do barramento do CPU.

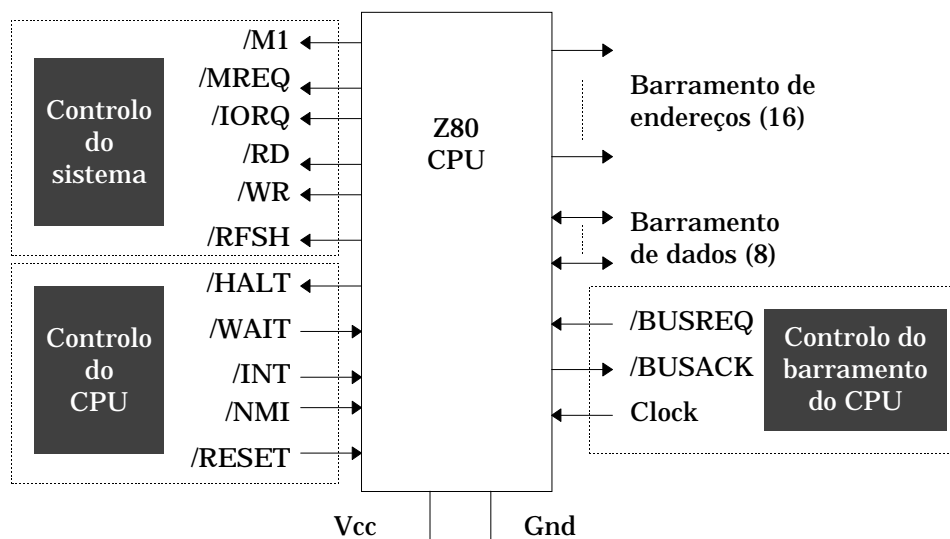


Figura 3.2: Barramentos do microprocessador Z80.

A funcionalidade destes sinais que integram o barramento de controlo do Z80 pode ser sumariada da forma que se apresenta a seguir:

1. Sinais de controlo do sistema:

- /M1: saída que indica os ciclos de leitura da memória que extraem o código da próxima instrução a executar.
- /MREQ: saída que indica que o barramento de endereços contém um endereço válido para aceder à memória.
- /IORQ: saída que indica que a metade menos significativa do barramento de endereços contém um endereço válido para aceder a dispositivos de entrada / saída (o Z80 distingue entre acesso à memória e acesso a dispositivos de entrada / saída, usando para este segundo caso um espaço de endereços com 256 posições). Este sinal também é usado durante os ciclos de atendimento a um pedido de interrupção.
- /RD: saída que indica que o CPU pretende efectuar uma operação de leitura.

- /WR: saída que indica que o CPU pretende efectuar uma operação de escrita.
  - /RFSH: saída que indica que os 7 bits menos significativos do barramento de endereços contêm um endereço de refrescamento para memórias dinâmicas (tem actualmente pouco interesse, dada a evolução tecnológica das memórias dinâmicas).
2. Sinais de controlo do CPU (os sinais /INT e /NMI serão analisados em maior detalhe quando adiante se considerar o processamento dos pedidos de interrupção):
- /HALT: saída que indica que o CPU executou uma instrução de *halt* (que suspende a execução do programa) e que aguarda um pedido de interrupção ou uma ordem de reinicialização (*reset*) para retomar a actividade.
  - /WAIT: entrada que permite estender a duração temporal dos ciclos de acesso a memória ou a dispositivos de entrada / saída, quando estes não forem suficientemente rápidos para acompanharem a rapidez de funcionamento do CPU (actualmente também com pouco interesse, pelas mesmas razões de evolução tecnológica).
  - /INT: entrada que recebe pedidos de interrupção que podem ou não ser atendidos, conforme o programa executado pelo CPU.
  - /NMI: entrada que recebe pedidos de interrupção que são sempre atendidos (*non-maskable interrupt*).
  - /RESET: entrada que provoca a inicialização do CPU, forçando-o a executar a rotina que se inicia no endereço 0000h.
3. Sinais de controlo do barramento do CPU:
- /BUSREQ: entrada para solicitar ao microprocessador que coloque em alta impedância em todos os pinos que suportam esta facilidade, de modo a possibilitar a outro dispositivo o controlo dos barramentos de endereços, dados e controlo.
  - /BUSACK: saída que indica quando o estado de alta impedância está já presente nos pinos que suportam esta facilidade, em resposta a um pedido para controlo externo dos barramentos.

## O microprocessador Z80

- **CLOCK:** entrada que recebe o sinal de relógio que cadencia toda a actividade do CPU.

No encapsulamento DIP (*Dual In-line Package*), o Z80 surge com a configuração de pinos que se ilustra na figura 3.3:

A11	1		40	A10
A12	2		39	A9
A13	3		38	A8
A14	4		37	A7
A15	5		36	A6
CLK	6		35	A5
D4	7		34	A4
D3	8		33	A3
D5	9		32	A2
D6	10		31	A1
+5 V (Vcc)	11		30	A0
D2	12		29	0 V (Gnd)
D7	13		28	/RFSH
D0	14		27	/M1
D1	15		26	/RESET
/INT	16		25	/BUSREQ
/NMI	17		24	/WAIT
/HALT	18		23	/BUSACK
/MREQ	19		22	/WR
/IORQ	20		21	/RD

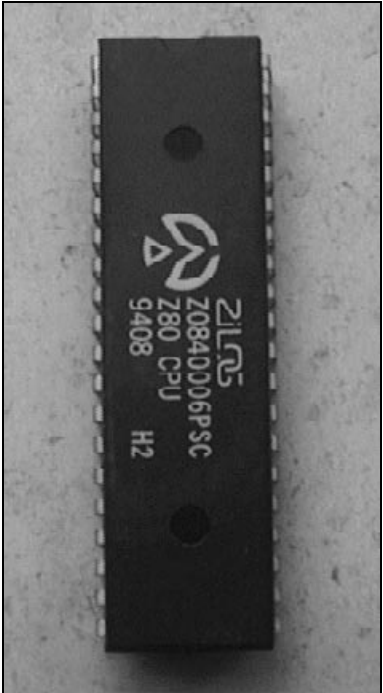


Figura 3.3: Configuração de pinos do Z80 (encapsulamento DIP).

### 3.2.2 Diagramas temporais

A compreensão dos diagramas temporais é frequentemente encarada com alguma relutância, essencialmente por duas razões:

1. Existe a percepção (enganosa) de que se consegue compreender e dominar o processo de desenvolvimento de aplicações sem ser necessário aprofundar o conhecimento acerca dos aspectos temporais dos sinais presentes nos barramentos.
2. O número de parâmetros apresentados nas folhas de características dos microprocessadores, relativamente aos vários tipos de ciclos de acesso ao exterior que podem ter lugar, são em número elevado, o que por vezes confunde e desmotiva mesmo os projectistas mais cuidadosos (normalmente algumas dezenas de parâmetros, com indicação de valores mínimos, típicos ou máximos).

É no entanto fundamental que o leitor desenvolva um conhecimento sólido acerca destas questões, que como se verá não implicam uma análise demasiado demorada e fastidiosa de todos os parâmetros referidos. Os aspectos principais, a compreender em pormenor, são os que dizem respeito à relação entre os sinais do barramento de controlo e o conteúdo dos barramentos de endereços e de dados.

Existem no Z80 três tipos básicos de ciclos temporais, que são o acesso à memória, o acesso a entrada / saída (E/S) e o atendimento de interrupções. Cada um destes tipos de acesso tem a duração de vários ciclos de relógio, dando-se a cada ciclo de relógio a designação de *ciclo T*. Os ciclos T agrupam-se por sua vez em ciclos máquina (*ciclos M*), cada um com três ou mais daqueles ciclos. A inclusão de ciclos de atraso (através da entrada de controlo /WAIT do Z80) aumenta a extensão de um ciclo máquina, o que permitiria ligar a este microprocessador periféricos mais lentos (como já se referiu antes, é actualmente uma facilidade com pouco interesse, dado que serão muito poucos os casos em que os periféricos não consigam acompanhar a velocidade do Z80).

Os vários tipos de ciclos referidos serão agora analisados em algum detalhe, considerando para tal o microprocessador Z80A a 4 MHz, com o objectivo de permitir ao leitor compreender este importante aspecto de utilização de qualquer componente deste tipo.

1. A figura 3.4 ilustra um ciclo de leitura da memória para extracção de um código de operação de uma instrução (*instruction fetch*). Repare-se que o único parâmetro temporal que se representa nesta figura é o que decorre de o Z80 requerer que a memória coloque no barramento de dados os valores contidos na posição endereçada, no mínimo com 35 ns de antecedência relativamente à subida do sinal de relógio que inicia o ciclo T3 (esta transição dará por sua vez origem à desactivação do sinal de leitura). Embora existam neste diagrama temporal muitos outros parâmetros especificados pelo fabricante (que o leitor poderá consultar na folha de características do componente), este é de facto o parâmetro principal sobre o qual a nossa atenção deve incidir, no sentido de garantir que o componente de memória usado respeite esta condição. Como veremos quando no próximo capítulo analisarmos os diagramas temporais dos dispositivos de memória, trata-se de uma condição geralmente simples de satisfazer (repare-se que os sinais /RD e /MREQ estarão activos ainda antes da subida do relógio para iniciar T2).

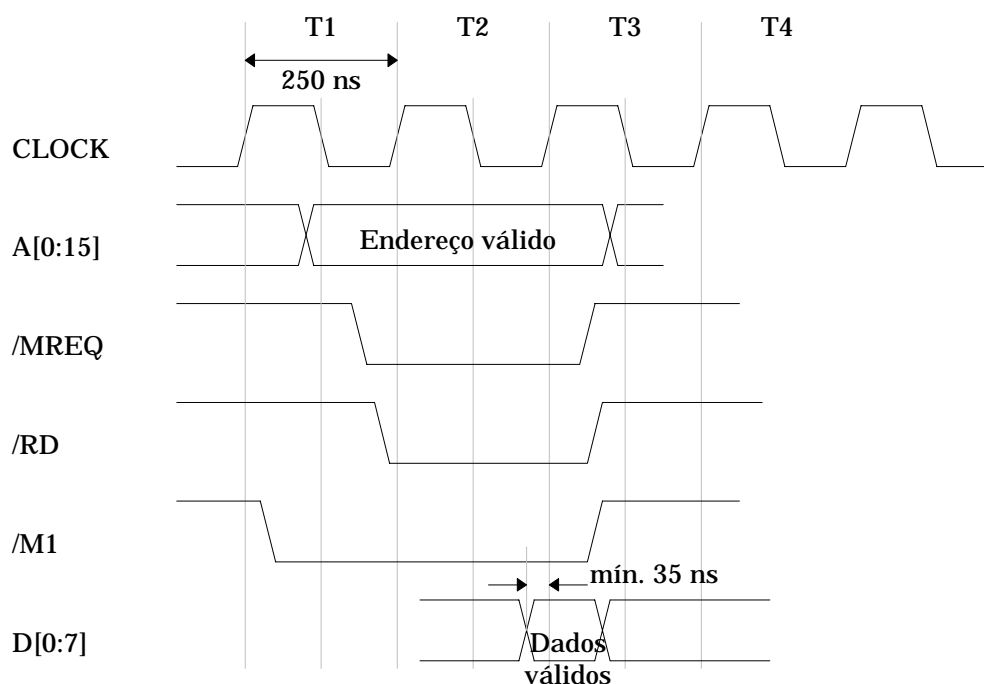


Figura 3.4: Ciclo de acesso a memória para leitura de um código de instrução (Z80A a 4 MHz).

Quando /MREQ abandonar o estado activo (no ciclo de relógio T3) o sinal de selecção da memória será normalmente desactivado (através do bloco de decodificação), levando a que o conteúdo do barramento de dados deixe de estar válido. Repare-se ainda que o sinal /M1 não será habitualmente usado, a menos que se precise de identificar individualmente os ciclos de leitura de códigos de instrução (por exemplo, para se implementar uma execução do programa em modo instrução-a-instrução). Devem ainda referir-se dois aspectos que se omitiram voluntariamente na figura 3.4, já que actualmente têm muito pouca utilidade:

- A inclusão de ciclos de espera, pela colocação em 0 da entrada de /WAIT do Z80, deixou de ter interesse uma vez que os periféricos hoje comuns conseguem acompanhar a rapidez de funcionamento do microprocessador.
- O ciclo de relógio T4, neste ciclo de leitura do código de operação, é usado para gerar os sinais de refrescamento de memórias dinâmicas (não representado na figura 3.4). Atendendo no entanto à evolução tecnológica que se verificou no domínio dos componentes de memória, esta possibilidade deixou de ter interesse.

O ciclo de acesso para leitura de dados tem características temporais idênticas às do que efectua a extracção de um código de instrução, devendo neste caso

garantir-se que os dados estarão válidos pelo menos 50 ns antes da descida do sinal de relógio que inicia o ciclo T3 (em vez dos 35 ns do diagrama anterior). Continua portanto a tratar-se de uma condição simples de satisfazer, como se ilustra pela antecendência com que se estabelecem as condições de descodificação do endereço que se pretende ler, também ilustrada na figura 3.5 (mín. 30 ns + 250 ns + máx. 75 ns).

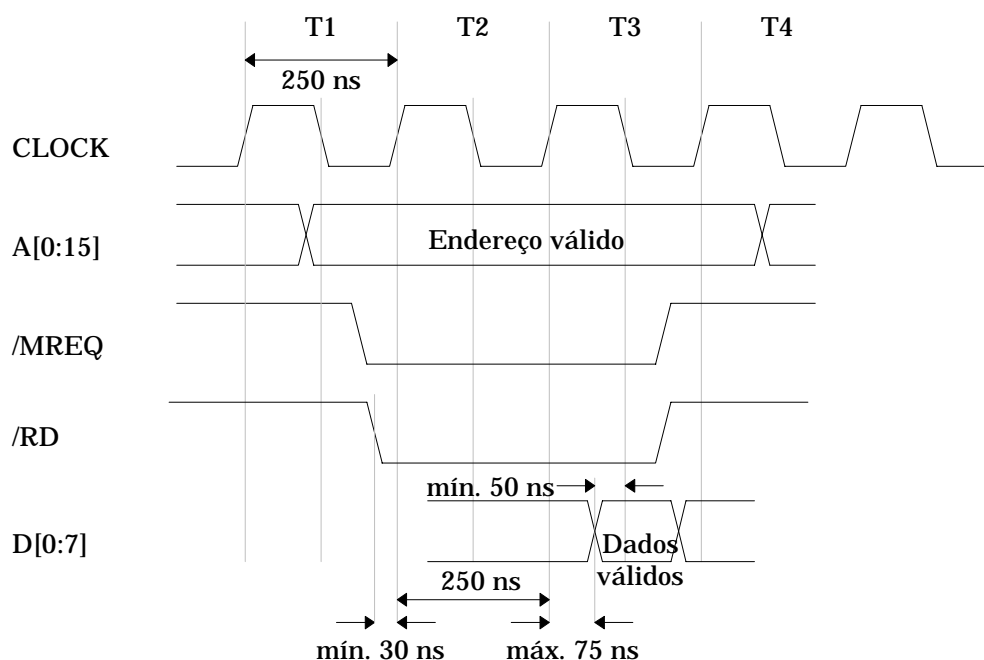


Figura 3.5: Ciclo de acesso a memória para leitura de dados (Z80A a 4 MHz).

Finalmente, e para encerrar os ciclos de acesso à memória, a figura 3.6 ilustra um ciclo de escrita. Neste tipo de ciclo são os componentes de memória que determinam as duas condições temporais que devem ser satisfeitas: garantir uma duração mínima para o intervalo de tempo em que o sinal de escrita (/WR) está activo e garantir que os dados no barramento se manterão estáveis durante um tempo mínimo em torno da passagem deste sinal para o estado inactivo (ambas as condições são plenamente satisfeitas no diagrama temporal ilustrado na figura 3.6).



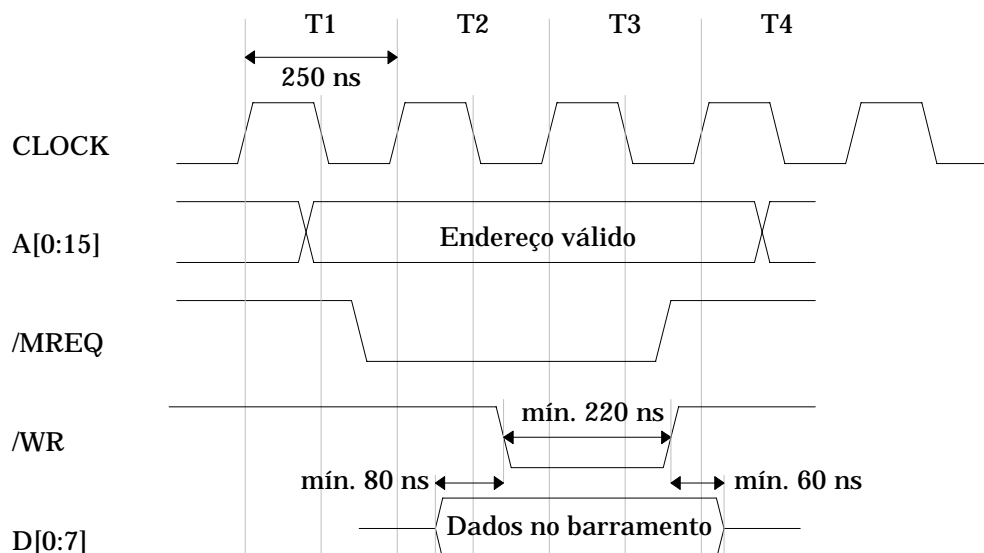


Figura 3.6: Ciclo de acesso a memória para escrita (Z80A a 4 MHz).

2. O outro tipo principal de ciclo de acesso do Z80 é o que diz respeito aos periféricos de E/S, estando ilustrados na figura 3.7 os detalhes mais relevantes da sua evolução temporal. Os sinais de /RD e D[0:7], imediatamente abaixo do sinal de /IORQ, ilustram a evolução temporal de um ciclo de leitura, enquanto os sinais de /WR e D[0:7], por sua vez abaixo dos anteriores, ilustram a evolução de um ciclo de escrita.

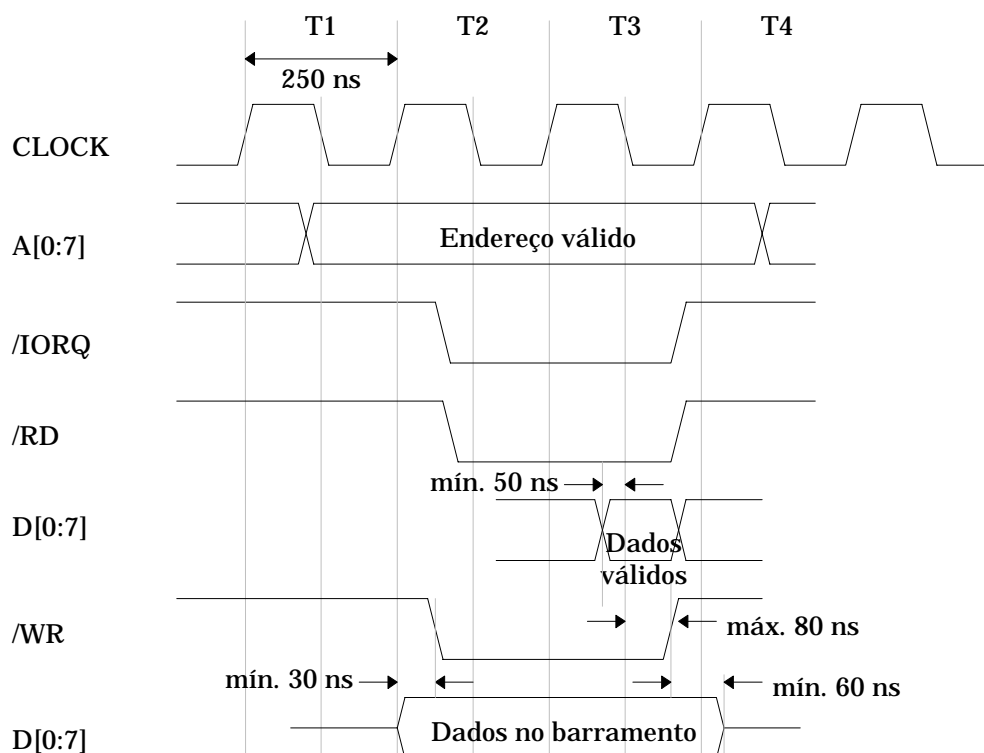


Figura 3.7: Ciclo de acesso a E/S (Z80A a 4 MHz).

Recorde-se que os endereços de acesso a dispositivos de entrada / saída são codificados em apenas 8 bits, pelo que no diagrama temporal da figura 3.7 se representam apenas os bits A[0:7] deste barramento.

3. A folha de características do Z80 ilustra ainda mais 5 ciclos temporais, cuja reprodução omitiremos para não tornar esta secção demasiado extensa:

- Atendimento a uma interrupção em /INT (*interrupt request / acknowledge cycle*): O CPU efectua a amostragem do sinal /INT na transição ascendente do último ciclo de relógio na execução de cada instrução. Se a interrupção for aceite, o CPU executa um ciclo especial com /M1 e /IORQ activos, durante o qual o dispositivo responsável pelo pedido pode colocar um vector de 8 bits no barramento de dados.
- Atendimento a uma interrupção em /NMI (*NMI request cycle*): A amostragem do sinal /NMI é feita ao mesmo tempo que a de /INT. Quando surge um pedido de interrupção neste pino o CPU executa um ciclo idêntico ao de leitura da memória, mas o valor que surge no barramento de dados é ignorado e o CPU vectoriza a execução do programa para o endereço 0066H.
- Pedido / cedência de barramentos (*bus request / acknowledge cycle*): O CPU efectua a amostragem do sinal /BUSREQ na transição ascendente do último período de relógio de qualquer ciclo máquina. Se /BUSREQ estiver activo, serão colocados em alta impedância os barramentos de endereços e de dados, e ainda os sinais /MREQ, /IORQ, /RD e /WR.
- Paragem (*halt acknowledge cycle*): Quando encontra uma instrução de HALT, o CPU coloca a saída /HALT em 0 e fica a aguardar um pedido de interrupção em /INT ou em /NMI.
- Inicialização (*reset cycle*): O sinal de /RESET deve estar activo durante pelo menos 3 ciclos de relógio, durante os quais o CPU mantém em alta impedância os barramentos de endereços, de dados e as linhas de saída do barramento de controlo.

Uma boa compreensão dos 4 tipos principais de diagramas temporais, ilustrados nas figuras 3.4 a 3.7, habilita no entanto o leitor a utilizar o Z80 na grande maioria das situações habituais, recomendando-se para os restantes casos a consulta da folha de características do componente.

### 3.3 MODELO DE PROGRAMAÇÃO

A expressão *modelo de programação* é frequentemente empregue para referir a configuração de registos de um microprocessador, sobre os quais são definidos os modos de endereçamento e o conjunto de instruções suportados pelo componente. Tal como se ilustra na figura 3.8, o Z80 possui 7 registos de uso genérico (A, B, C, D, E, H, L), dos quais 6 podem agrupar-se 2 a 2 para formar 3 pares de registos de 16 bits (B com C, D com E e H com L). Estes pares de registos podem igualmente ser usados de forma genérica, ou então como apontadores para posições de memória, tal como veremos quando mais tarde analisarmos os modos de endereçamento e os tipos de instruções.

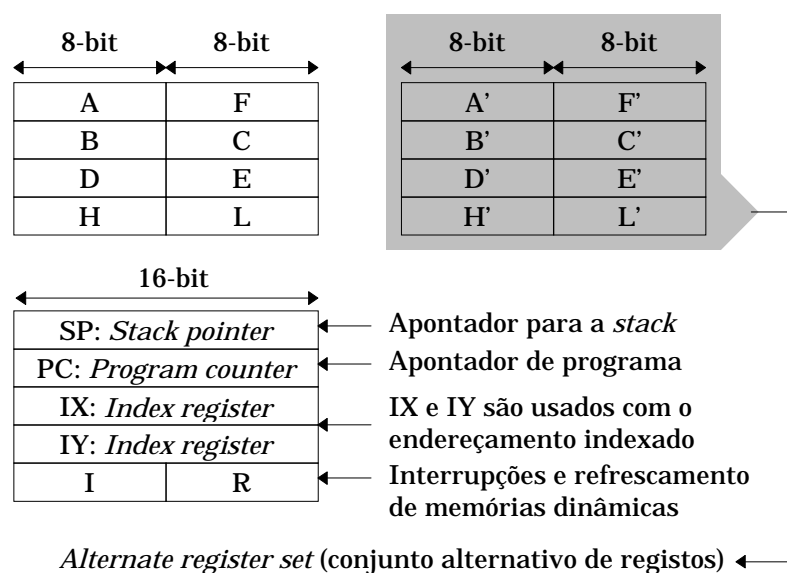


Figura 3.8: Organização dos registos no microprocessador Z80.

Para além destes registos de uso genérico o Z80 dispõe ainda de um conjunto de registos dedicados, nomeadamente 3 de 8 bits (F, I e R) e 4 de 16 bits (SP, PC, IX e IY). A funcionalidade específica destes registos pode ser brevemente apresentada como se segue:

- F: registo que contem as *flags* do microprocessador (dada a sua importância, este registo será descrito em pormenor já a seguir).
- I: o registo I contem o *byte* mais significativo do endereço que define a rotina de atendimento aos pedidos de interrupção no modo 2. O processamento de interrupções e o funcionamento da *stack* serão analisados na secção seguinte.

- R: este registo contém os 7 bits menos significativos que são colocados no barramento de endereços para efeitos de refrescamento de memórias dinâmicas (atendendo à evolução verificada no domínio das memórias, esta facilidade deixou de ter interesse).
- SP: este registo contém o apontador para a *stack*, que constitui uma zona de memória de leitura / escrita usada fundamentalmente para guardar a informação necessária à retoma da execução de um segmento de código, interrompido pela chamada de uma subrotina ou por um pedido de interrupção (como se referiu acima, a análise destas questões será considerada na secção seguinte).
- PC: registo que contém o apontador de programa, usado pelo microprocessador para extrair da memória os códigos de instrução e os respectivos operandos.
- IX e IY: dois registos de 16 bits que são usados como apontadores para posições de memória no modo de endereçamento indexado (que será apresentado mais tarde, quando forem introduzidos os modos de endereçamento).

Finalmente, a figura 3.8 ilustra ainda um conjunto alternativo de registos (*alternate register set*), sobre os quais é apenas possível realizar operações de troca de conteúdo com os correspondentes registos principais (servem para guardar temporariamente o valor dos registos principais, sem que seja necessário armazená-los em memória).

Como foi já anteriormente afirmado, o registo F contém o conjunto de *flags* que permitem implementar saltos condicionais, de acordo com o resultado de instruções anteriormente realizadas. A título de exemplo, é frequente que as rotinas que introduzem um atraso programável se baseiem na estratégia definida pelo seguinte procedimento:

```
inicio  carregar um registo com o valor que define o atraso;
ciclo   decrementar o registo;
        se (o registo atingiu zero)
            então termina a rotina
        senão
            salta para o ciclo
```

Este procedimento inclui um salto programável que se baseia no facto já se ter ou não chegado a zero, pelo que se torna necessário guardar esta informação de cada vez que se decrementar o registo. No Z80 a *flag* z (zero) tem por objectivo precisamente armazenar uma informação deste tipo, existindo então instruções de salto condicional (por exemplo `jp z,ciclo`, que será também apresentada mais tarde) que permitem testar o estado da *flag* pretendida e com base no seu valor determinar qual a instrução seguinte a realizar.

As *flags* existentes no registo F do Z80 estão ilustradas na figura 3.9, podendo o seu significado apresentar-se brevemente como se segue (considera-se que uma *flag* fica activa quando o seu estado passa a 1):

- s (*sign*): activa quando o bit mais significativo do resultado é 1.
- z (*zero*): activa quando o resultado da instrução tem o valor 0.
- h (*half-carry*): activa quando há transporte do bit 3 para o bit 4 (quer na adição quer na subtracção).
- p/v (*parity / overflow*): conforme o tipo de instrução executada, a *flag* afectada poderá ser a P ou a V (que partilham o mesmo bit no registo F). P dá-nos uma indicação de resultado com paridade par, enquanto V nos indica quando o resultado de uma operação excedeu o número de bits disponíveis para a sua representação.
- n: indica se a última operação realizada foi uma adição ou uma subtracção, sendo usada em operações internas do microprocessador (não existem operações de salto condicional baseadas no estado desta *flag*).
- c (*carry*): activa a operação realizada produz transporte para além do bit 7.

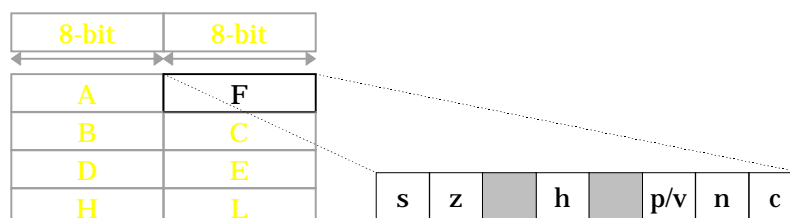


Figura 3.9: O registo das variáveis de estado (*flags*).

Existe alguma inconsistência na forma como cada *flag* é afectada pelas várias instruções, pelo que a forma mais segura de se proceder à respectiva análise é

através das informações que a este respeito estão apresentadas na folha de características do Z80.

### 3.4 PROCESSAMENTO DE INTERRUPÇÕES E FUNCIONAMENTO DA *STACK*

O objectivo de suportar o atendimento de interrupções consiste em permitir que o microprocessador responda a eventos que ocorrem de forma assíncrona relativamente ao conjunto de tarefas realizadas pelo programa principal. O mecanismo de atendimento, que se ilustra na figura 3.10, suspende temporariamente a execução do programa principal e força a execução de uma rotina que permitirá ao microprocessador responder ao evento que deu origem à interrupção. Uma vez concluída esta rotina efectuar-se-à o retorno ao programa principal, que será retomado no ponto onde foi interrompido.

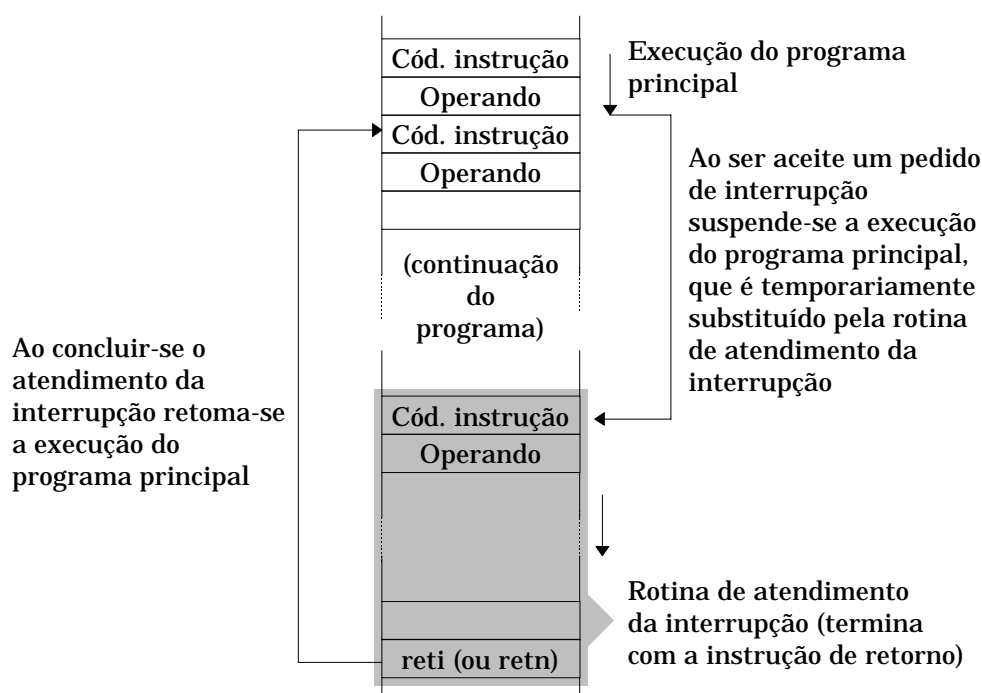


Figura 3.10: Visualização em memória do código principal e do atendimento às interrupções.

Este mecanismo de atendimento assume que o microprocessador guarda temporariamente em memória o endereço da próxima instrução (isto é, guarda o valor do PC) que seria executada se o pedido de interrupção não tivesse surgido, de forma a que seja possível retomar a execução do programa principal exactamente nesse mesmo ponto. A área de memória onde é guardado o valor do PC recebe a designação de *stack* e funciona como uma *pilha* do tipo “o último a entrar é o primeiro a sair” (LIFO, *last in first out*). Esta área de memória é gerida por um apontador a que se dá a designação de apontador para a *stack* (SP,

*stack pointer*), que aponta para a posição onde se encontra o último valor aqui armazenado.

No Z80 a *stack* cresce no sentido descendente, isto é, o SP é decrementado de cada vez que se armazena um novo valor. Este processo está representado na figura 3.11, que ilustra o conteúdo da *stack* e o valor do SP antes e depois de se ter armazenado o valor do PC, que lá estará armazenado enquanto decorrer o atendimento a um pedido de interrupção.

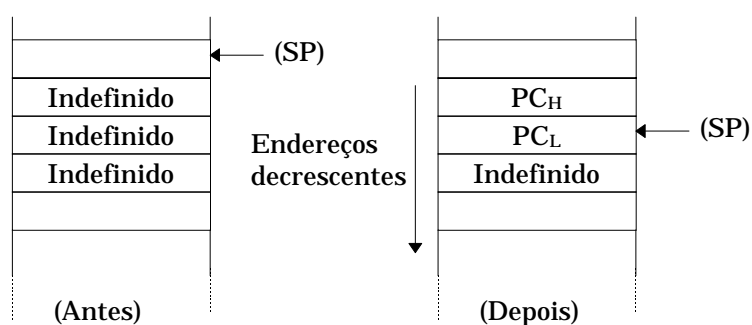


Figura 3.11: Conteúdo da *stack* e alteração do SP durante o atendimento a uma interrupção.

Optou-se por introduzir o funcionamento da *stack* em simultâneo com o processamento das interrupções porque o armazenamento do PC na *stack* é nestes casos automático. Com efeito, o próprio mecanismo de atendimento das interrupções é que se encarrega de suspender a extracção do próximo código de instrução, guardar na *stack* os dois *bytes* do PC e obter o endereço da rotina que serve a interrupção. Se esta rotina terminar com a instrução `reti` (ou `retn`, conforme o pedido de interrupção), a recuperação do valor do PC e a reposição do valor do SP (que voltará a apontar para a posição que se ilustra à esquerda na figura 3.11) ocorrerão também automaticamente.

Para além das instruções que afectam directamente o valor do SP ou o conteúdo da *stack* (veremos mais adiante que existem instruções que usam a *stack* para armazenar ou recuperar o conteúdo dos registos do Z80), também as chamadas a subrotinas e o respectivo retorno (respectivamente as instruções do tipo `call` e `ret`, que serão também introduzidas mais tarde) usam a área da *stack*, neste caso da mesma forma anteriormente ilustrada para o atendimento às interrupções. Estes comentários chamam-nos a atenção para o facto de que a inicialização conveniente do SP deve ser uma das primeiras tarefas a realizar pelo microprocessador, já que a perda de um valor que deveria ter sido guardado na

*stack* impedi-lo-à de reencontrar o ponto onde interrompeu a execução de um programa, conduzindo-o a um comportamento aleatório.

Regressando agora ao processamento dos pedidos de interrupção, deve referir-se que os dois pinos disponíveis no Z80 para este efeito se distinguem entre si pelos seguintes aspectos principais:

- */NMI* (*non-maskable interrupt*) destina-se aos pedidos de interrupção prioritários, já que não é possível evitar o seu atendimento, que terá lugar assim que terminar a execução da instrução actual. Trata-se de um sinal activo à descida, que vectoriza a execução do programa para o endereço 0066h. A rotina que se inicia neste endereço deverá terminar com a instrução *retn* (*return from nmi*), de modo a ser retomado o programa interrompido.
- */INT* destina-se aos pedidos de interrupção de segunda prioridade, cujo atendimento pode ser impedido pelo programa em execução. O Z80 suporta as instruções *ei* e *di* (*enable interrupt* e *disable interrupt*, respectivamente), cujo efeito é o de habilitar ou inibir o atendimento aos pedidos de interrupção neste pino (estas duas instruções actuam sobre um *flip-flop* designado por IFF — *interrupt flip-flop* — que permitirá o atendimento quando estiver em 1, impedindo-o no caso contrário). O modo de atendimento pode ser seleccionado de entre 3 alternativas à escolha, que se descrevem sumariamente a seguir:
  1. No modo 0 o dispositivo responsável pelo pedido de interrupção será também responsável por colocar no barramento de dados o código da instrução que o microprocessador deverá executar. Este modo destinou-se a manter a compatibilidade com o 8080A da Intel, de que o Z80 descende, e a instrução a colocar no barramento de dados será normalmente uma instrução de salto para o endereço que contem a rotina de atendimento. Este modo é seleccionado pela instrução *im 0*.
  2. No modo 1 o atendimento da interrupção é automaticamente vectorizado para o endereço 0038h (idêntico ao que sucede com a resposta a */NMI*, que salta para 0066h). Este modo é seleccionado pela instrução *im 1*.
  3. No modo 2 o dispositivo que efectua o pedido de interrupção será agora responsável por fornecer a metade menos significativa do endereço da posição de uma tabela, que conterà por sua vez o endereço inicial da rotina de atendimento, sendo a metade mais significativa obtida de um registo interno



do Z80 a que se dá a designação de registo I (que deverá ser previamente carregado com um valor adequado). Este modo permite mudar facilmente a rotina que serve cada dispositivo, bastando para tal alterar a tabela com os respectivos endereços. Este modo é seleccionado pela instrução `im 2`.

Nas aplicações simples o mais prático será optarmos pelo atendimento no modo 1, no qual a resposta ao pedido de interrupção é automaticamente vectorizada para o endereço 0038h. Sobretudo nos casos em que os dispositivos que solicitam os pedidos de interrupção não tenham capacidade para forçar nos barramentos os valores necessários ao funcionamento nos outros dois modos, este modo 1 revela-se uma solução fácil e eficaz.

Antes de concluirmos esta secção, resta ainda esclarecer a razão pela qual as rotinas de atendimento a `/NMI` e a `/INT` terminam com diferentes instruções de retorno (`retn` e `reti`, respectivamente). Para este efeito convém referir que existem de facto internamente dois *flip-flops* de interrupção, IFF1 e IFF2, sendo IFF1 aquele que é directamente controlado pelo programador (através das instruções `ei` e `di`). Quando o microprocessador inicia o atendimento a uma interrupção em `/NMI` copia o estado de IFF1 para IFF2 e coloca IFF1 em 0, de modo a que a rotina de atendimento a esta interrupção de alta prioridade não seja eventualmente interrompida por um pedido de interrupção com prioridade inferior (em `/INT`). Quando se efectua o retorno é necessário restabelecer o valor de IFF1, que é automaticamente recuperado (a partir da sua cópia em IFF2) pela instrução `retn`. Uma vez que esta questão não se coloca relativamente ao atendimento dos pedidos de interrupção em `/INT`, a respectiva instrução de retorno (`reti`) não inclui esta operação.

### 3.5 MODOS DE ENDEREÇAMENTO

Tal como veremos adiante relativamente aos tipos de instruções considerados, existe igualmente alguma dispersão entre a literatura disponível no que se refere à classificação dos modos de endereçamento suportados pelo Z80. A estratégia que seguiremos aqui consiste em adoptar directamente a classificação que é usada nas folhas de características do microprocessador, de modo a que o leitor esteja inteiramente familiarizado com a terminologia que encontrará sempre que as consultar.

Uma boa compreensão dos modos de endereçamento suportados por um microprocessador é um factor de enorme importância para permitir o desenvolvimento eficiente de código. Antes de mais, convém começar por esclarecer que a expressão *modos de endereçamento* se refere à forma como, em cada instrução, são especificados os operandos ou indicados os endereços. Como o leitor já terá eventualmente reparado, a mesma instrução (por exemplo, para carregar o registo A com um determinado valor) pode assumir várias formas, às quais correspondem naturalmente diferentes códigos de instrução. A título de exemplo, considerem-se as instruções apresentadas na tabela 3.1, onde se assume que todas elas carregam o registo A com o valor 5Fh. Repare-se que a notação (1000h) é usada para representar o conteúdo da posição de memória cujo endereço é 1000h (o *h* indica a notação hexadecimal).

Mnemónica	Código	#Byte	#M	#T	Assunções
ld a,5fh	3E 5F	2	2	7	
ld a,b	78	1	1	4	b contem 5fh
ld a,(1000h)	3A 00 10	3	4	13	(1000h) contem 5fh
ld a,(hl)	7E	1	2	7	(hl) contem 5fh

Tabela 3.1: Exemplo de diferentes modos de endereçamento empregues com a mesma instrução.

A compreensão dos elementos apresentados nesta tabela é fundamental para o correcto desenvolvimento de aplicações baseadas no Z80, bem como para a compreensão das tabelas que serão apresentadas a seguir, pelo que se chama a atenção do leitor para a análise em pormenor dos seguintes aspectos:

1. A coluna *Mnemónica* contem a representação simbólica que é normalmente usada para apresentar a instrução considerada, de acordo com as regras sintácticas seguidas na folha de características do Z80 e normalmente aceites por *cross-assemblers* como o TASM.<sup>†</sup>
2. A coluna *Código* representa o código objecto que corresponde à instrução considerada (as palavras de 8 bits que serão gravadas em memória). Um vez que o número de instruções do Z80 ultrapassa as 256 combinações que são possíveis com uma palavra de 8 bits, algumas delas possuem códigos com 2

---

<sup>†</sup> Os *cross-assemblers* são programas de computador que lêem o ficheiro que criamos com o código *assembly* (que neste caso constitui o código fonte) e produzem a partir dele o ficheiro com o código objecto (códigos de instrução e operandos) que é executado pelo Z80.

*bytes* (não é este no entanto o caso de nenhuma das instruções representadas na tabela 3.1).

3. A coluna *#Byte* representa o número de *bytes* em memória que serão ocupados pela instrução considerada. Por exemplo, a instrução `ld a,(1000h)` ocupa 3 *bytes* em memória, que correspondem ao código desta instrução (3A) e aos dois *bytes* ocupados pelo endereço (1000h) de 16 bits. Repare-se ainda que as palavras de 16 bits são armazenadas pelo Z80 em memória com o *byte* menos significativo em primeiro lugar (nem todos os microprocessadores seguem este procedimento), como se pode constatar acima pelo código correspondente à instrução `ld a,(1000h)`.
4. A coluna *#M* representa o número de ciclos M (ciclos máquina, correspondentes a três ou mais ciclos de relógio — ciclos T — cada um, como se descreveu quando se apresentaram os diagramas temporais do Z80) necessários para a execução da instrução considerada.
5. A coluna *#T* representa o número total de ciclos de relógio requeridos para a execução da instrução (multiplicando este número pelo período T obtém-se o tempo que cada instrução demora a executar).

Desde que a notação empregue seja bem compreendida, as indicações apresentadas nas tabelas seguintes são suficientemente explícitas para se compreender o resultado da execução das instruções consideradas.

Os exemplos apresentados na tabela 3.1 permitem-nos constatar que o mesmo resultado (A carregado com 5Fh) pode ser obtido com um número de ciclos de relógio (ciclos T) que varia entre 4 e 13, e ocupando 1, 2 ou 3 *bytes* em memória. Estas diferenças fazem com que o modo de endereçamento mais adequado para cada caso varie conforme as circunstâncias. Por exemplo, e embora a instrução `ld a,(hl)` torne necessário que se inicialize anteriormente o par HL, a leitura de uma tabela de dados usando esta instrução poderá ser feita de forma mais rápida e através de um código mais compacto do que se optássemos por instruções do tipo `ld a,(1000h)` (respectivamente 7 ciclos em vez de 13 e 1 *byte* em vez de 3, por cada operação de leitura).

Os modos de endereçamento referidos na folha de características do Z80 são os seguintes:

1. Imediato (*immediate*)
2. Imediato estendido (*immediate extended*)
3. Página zero modificado (*modified page zero*)
4. Relativo (*relative*)
5. Estendido (*extended*)
6. Indexado (*indexed*)
7. Ao registo (*register*)
8. Indirecto por registo (*register indirect*)
9. Implícito (*implied*)
10. Ao bit (*bit*)

Cada um destes modos será agora brevemente apresentado, incluindo alguns exemplos de aplicação. As tabelas que conterão os exemplos a apresentar daqui em diante contêm duas colunas adicionais, cujo significado é o seguinte:

1. A coluna *Operação* descreve o resultado da execução da instrução representada na coluna *Mnemónica* ( $a \leftarrow b$  significa que o registo A é carregado com o conteúdo do registo B, mantendo-se B inalterado). Nas operações de transferência de dados representa-se sempre em primeiro lugar o destino e depois a origem (`ld a,b` carrega A com o conteúdo de B).
2. As *flags* afectadas pela execução de cada instrução estão indicadas na coluna com este nome. Para cada *flag* existem quatro possibilidades: não é afectada, é colocada sempre em 0, é colocada sempre em 1, ou é colocada em 0 ou em 1 conforme o resultado da operação.

### **Imediato**

Trata-se de um modo de endereçamento em que o operando de 8 bits envolvido numa instrução está contido na posição de memória imediatamente a seguir à(s) do código de instrução, como se ilustra nos exemplos da tabela 3.2:

Mnemónica	Operação	Flags	Código	#Byte	#M	#T
ld a,0ffh	$a \leftarrow \text{ffh}$	nenhuma	3E FF	2	2	7
ld (ix+20h),5ah	$(\text{ix}+20\text{h}) \leftarrow 5\text{ah}$	nenhuma	DD 36 20 5A	4	5	19

Tabela 3.2: Exemplo de instruções que usam o modo de endereçamento imediato.

### Imediato estendido

Idêntico ao anterior mas referente aos operandos de 16 bits, tal como se ilustra nos exemplos da tabela 3.3:

Mnemónica	Operação	Flags	Código	#Byte	#M	#T
ld hl,1000h	$\text{hl} \leftarrow 1000\text{h}$	nenhuma	21 00 10	3	3	10
ld sp,8000h	$\text{sp} \leftarrow 8000\text{h}$	nenhuma	31 00 80	3	3	10
ld iy,8000h	$\text{iy} \leftarrow 8000\text{h}$	nenhuma	FD 21 00 80	4	4	14

Tabela 3.3: Exemplo de instruções que usam o modo de endereçamento imediato estendido.

### Página zero modificado

É o modo de endereçamento empregue nas oito instruções de *restart*, que provocam a chamada a uma subrotina cujo endereço tem por *byte* mais significativo 00 (trata-se portanto da primeira “página” no espaço de memória do microprocessador, indicando-se apenas no operando da instrução o *byte* menos significativo do endereço). Ilustra-se na tabela 3.4 o efeito destas instruções de *restart*:

Mnemónica	Operação	Flags	Código	#Byte	#M	#T
rst 00h	$(\text{sp}-1) \leftarrow \text{pCH}$ $(\text{sp}-2) \leftarrow \text{pCL}$ $\text{sp} \leftarrow \text{sp}-2$ $\text{pCH} \leftarrow 00\text{h}$ $\text{pCL} \leftarrow 00\text{h}$	nenhuma	C7	1	3	11
rst 38h	$(\text{sp}-1) \leftarrow \text{pCH}$ $(\text{sp}-2) \leftarrow \text{pCL}$ $\text{sp} \leftarrow \text{sp}-2$ $\text{pCH} \leftarrow 00\text{h}$ $\text{pCL} \leftarrow 38\text{h}$	nenhuma	FF	1	3	11

Tabela 3.4: Exemplo de instruções que usam o modo de endereçamento página zero modificado.

As oito instruções de *restart* vectorizam a execução do programa para endereços que distam entre si 8 *bytes* (00h, 08h, 10h, ..., 30h, 38h), onde em princípio estará contida uma instrução de salto para outro local (os 8 *bytes* até ao próximo endereço de atendimento serão normalmente insuficientes para conter o código que se pretende executar). Repare-se que a instrução `rst 00h` é atendida pela

rotina de inicialização (*reset*), embora o estado interno do processador e dos periféricos não sofra a inicialização por *hardware* que tem lugar quando se liga a alimentação. Estas instruções têm sobretudo interesse para vectorizar as respostas a pedidos de interrupção no pino /INT, quando o modo 0 está seleccionado (recordem-se os comentários feitos neste capítulo a respeito do atendimento a pedidos de interrupção). Aliás, quando o modo seleccionado é o 1, o próprio microprocessador responde ao pedido de interrupção em /INT executando automaticamente uma instrução *rst 38h*.

### Relativo

Trata-se de um modo de endereçamento em que o operando das instruções de salto (o novo endereço, portanto) é indicado implicitamente, através da diferença (*offset*) entre o endereço da instrução que se segue à actual e o endereço para onde se pretende redireccionar a execução do programa. Na tabela 3.5 ilustram-se alguns exemplos de instruções que usam o modo de endereçamento relativo:

Mnemónica	Operação	Flags	Código	#Byte	#M	#T
jr z,marca	se z=1	nenhuma	28 05	2	3	12
	pc ← marca			2	2	se z=1 7
	se não continua					se z≠1
jr marca	pc ← marca	nenhuma	18 05	2	3	12
djnz marca	b ← b-1	nenhuma	10 05	2	2	8
	se b=0			2	3	se b=0 13
	continua					se b≠0
	se não continua					
	pc ← marca					

Tabela 3.5: Exemplo de instruções que usam o modo de endereçamento salto relativo.

Refira-se ainda que o *offset* é representado em memória como um *byte* no formato complemento para dois, pelo que o bit mais significativo em 1 indicará que o salto é para trás, sendo para a frente no caso contrário. Isto significa que a amplitude do salto não pode exceder os 127 *bytes* para a frente ou os 128 *bytes* para trás, como se conclui da tabela 3.6, em que se apresenta a correspondência entre a representação em complemento para dois e os decimais e hexadecimais equivalentes (a regra para determinar o complemento para dois de um número

negativo consiste em complementar todos os bits da sua representação binária e adicionar 1)<sup>†</sup> :

Decimal	Complemento para dois
-128	10000000 (hexadecimal 80)
-127	10000001 (hexadecimal 81)
-126	10000010 (hexadecimal 82)
...	...
-2	11111110 (hexadecimal FE)
-1	11111111 (hexadecimal FF)
0	00000000 (hexadecimal 00)
1	00000001 (hexadecimal 01)
2	00000010 (hexadecimal 02)
...	...
126	01111110 (hexadecimal 7E)
127	01111111 (hexadecimal 7F)

Tabela 3.6: Representação em complemento para dois.

Este modo de endereçamento merece uma atenção especial, atendendo à frequência da sua utilização. Uma instrução de salto relativo (condicional ou incondicional) ocupa apenas dois *bytes* em memória (as de salto não relativo ocupam 3 *bytes*), sendo o primeiro o código da instrução e o segundo a diferença (*offset*) em complemento para 2 entre o endereço para onde se pretende efectuar o salto e o endereço do código de instrução seguinte ao actual (no final da execução desta instrução de salto relativo o PC estará a apontar para o código da instrução seguinte).

É importante referir que os projectistas raramente se darão ao trabalho de calcular o valor deste *offset*, preferindo normalmente escrever esta instrução na forma `jr z,marca` (aqui na forma de um salto condicional), como se fez acima na tabela 3.5, em que *marca* representa o nome simbólico (*label*) da linha de programa para onde se pretende fazer o salto. Quando escrevemos o nosso programa em linguagem simbólica (*assembly*) podemos efectivamente ignorar o cálculo do *offset*, devendo o *cross-assembler* que efectua a conversão para código objecto determinar o seu valor e utilizá-lo para preencher o segundo *byte* da instrução. O argumento 05 que aparece como *offset* na tabela 3.5 indica que, no exemplo considerado, a linha para onde se pretende efectuar o salto (e a que se

---

<sup>†</sup> Uma outra regra consiste em ler os bits da direita para a esquerda até aparecer o primeiro 1 e a partir daí complementar todos os restantes bits.

deu o nome simbólico de *marca*) se encontra 5 *bytes* após o fim da instrução de salto relativo, como se ilustra na figura 3.12.

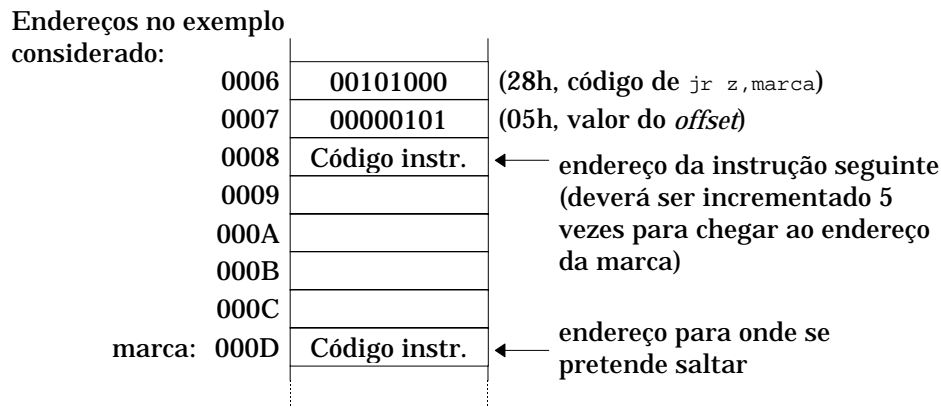


Figura 3.12: Determinação do *offset* nas instruções de salto relativo.

## Estendido

O endereçamento estendido é também frequentemente designado por endereçamento directo e é empregue quando, nas instruções cujo operando é um endereço, este está contido nos dois *bytes* seguintes ao(s) do código da instrução. Na tabela 3.7 ilustram-se alguns exemplos de endereçamento estendido:

Mnemónica	Operação	Flags	Código	#Byte	#M	#T
ld hl,(1000h)	$h \leftarrow (1001h)$ $l \leftarrow (1000h)$	nenhuma	2A 00 10	3	5	16
ld ix,(2000h)	$ix_H \leftarrow (2001h)$ $ix_L \leftarrow (2000h)$	nenhuma	DD 2A 00 20	4	6	20
ld (3000h),iy	$(3001h) \leftarrow iy_H$ $(3000h) \leftarrow iy_L$	nenhuma	FD 22 00 30	4	6	20

Tabela 3.7: Exemplo de instruções que usam o modo de endereçamento estendido.

## Indexado

O modo de endereçamento indexado permite o acesso a operandos cujo endereço está contido nos registos IX ou IY, directamente ou através da soma de um deslocamento (*offset*). O deslocamento surge em complemento para 2 (representa um valor negativo se o bit 7 estiver em 1), pelo que esta instrução permite também o acesso a operandos cujo endereço é inferior ao que está contido no registo de base (IX ou IY). Na tabela 3.8 ilustram-se algumas instruções que usam o modo de endereçamento indexado:



Mnemónica	Operação	Flags	Código	#Byte	#M	#T
ld c,(ix)	$c \leftarrow (ix)$	nenhuma	DD 4E 00	3	5	19
add a,(iy+20h)	$a \leftarrow a+(iy+20h)$	s, z, h, v, n=0, c	FD 86 20	3	5	19
ld (ix+20h),5ah	$(ix+20h) \leftarrow 5ah$	nenhuma	DD 36 20 5A	4	5	19

Tabela 3.8: Exemplo de instruções que usam o modo de endereçamento indexado.

Esta é a primeira das tabelas apresentadas até agora em que uma instrução afecta as *flags*, pelo que se justificam alguns comentários a este respeito. A notação s, z, h, v, n=0, c, significa que as *flags* de sinal (s), zero (z), *half-carry* (h), *overflow* (v) e *carry* (c) serão colocadas em 0 ou em 1 conforme o resultado da operação, enquanto que a *flag* de adição / subtracção será colocada em 0 (n só será colocada em 1 quando a operação realizada for uma subtracção, como se poderá verificar em algumas das instruções que serão consideradas adiante).

Repare-se que o endereçamento indexado assume sempre a existência de um deslocamento para adicionar ao endereço base, mesmo quando este deslocamento é nulo (repare-se na instrução ld c,(ix) acima, cujo código inclui o *byte* 00 que representa o deslocamento a adicionar ao conteúdo de IX).


### Ao registo

No endereçamento ao registo o próprio código de instrução contem um conjunto de bits que indicam qual (ou quais) o(s) registo(s) envolvido(s). A tabela 3.9 ilustra algumas instruções em que o endereçamento ao registo é empregue:

Mnemónica	Operação	Flags	Código	#Byte	#M	#T
ld a,b	$a \leftarrow b$	nenhuma	78	1	1	4
ld c,(hl)	$c \leftarrow (hl)$	nenhuma	4E	1	2	7
ld a,0ffh	$a \leftarrow fffh$	nenhuma	3E FF	2	2	7

Tabela 3.9: Exemplo de instruções que usam o modo de endereçamento ao registo.

A combinação que identifica qual o registo de destino está contida nos bits 5, 4 e 3 do código de instrução, tal como se ilustra na figura 3.13 (os restantes bits codificam a instrução a executar).

7 6 5 4 3 2 1 0	Combinacoo	Registo
	000	B
	001	C
	010	D
	011	E
	100	H
	101	L
	110	ilegal
	111	A

(byte que contem o cdigo de instruo)

Figura 3.13: Codificao do registo destino no modo de endereamento ao registo.

A determinao do cdigo objecto correspondente a cada instruo possvel  naturalmente uma tarefa que cabe ao *cross-assembler* empregar, bastando ao programador descrever na mnemnica quais os registos envolvidos.

### Indirecto por registo

Neste modo  usado um par de registos que especifica o endereo a que se pretende aceder. Repare-se que se trata de um modo distinto do modo indexado, onde se assume sempre a existncia de um deslocamento a acrescentar ao endereo base. Na tabela 3.10 ilustram-se alguns exemplos de instrues que usam este modo de endereamento.

Mnemnica	Operao	Flags	Cdigo	#Byte	#M	#T
ld a,(bc)	$a \leftarrow (bc)$	nenhuma	0A	1	2	7
ld c,(hl)	$c \leftarrow (hl)$	nenhuma	4E	1	2	7
ld (de),a	$(de) \leftarrow a$	nenhuma	12	1	2	7

Tabela 3.10: Exemplo de instrues que usam o modo de endereamento indirecto por registo.

### Implicito

Neste modo de endereamento o prprio cdigo de instruo tem implcito (por oposio ao endereamento ao registo, no qual a especificao  explcita) qual o registo que estar envolvido na operao. Na tabela 3.11 ilustram-se alguns exemplos deste modo de endereamento:

Mnemnica	Operao	Flags	Cdigo	#Byte	#M	#T
cpl	$a \leftarrow /a$	$h=1, n=1$	2F	1	1	4
cp 30h	$a-30h$	$s, z, h, v, n=1, c$	FE 30	2	2	7

Tabela 3.11: Exemplo de instrues que usam o modo de endereamento implcito.

## Ao bit

Neste modo de endereçamento os bits 5, 4 e 3 do código de instrução incluem a representação binária da ordem do bit sobre o qual a operação (de *set*, *reset* ou teste) será realizada. Este modo de endereçamento está ilustrado nas instruções apresentadas na tabela 3.12:

Mnemónica	Operação	Flags	Código	#Byte	#M	#T
bit 3,(ix+7fh)	$z \leftarrow /(ix+7fh)[3]$	z, h=1, n=0	DD CB 7F 5E	4	5	20
set 0,a	$a[0] \leftarrow 1$	nenhuma	CB C7	2	2	8
res 0,a	$a[0] \leftarrow 0$	nenhuma	CB 87	2	2	8

Tabela 3.12: Exemplo de instruções que usam o modo de endereçamento ao bit.

A concluir esta secção, pensamos ser útil voltar a chamar a atenção do leitor para os seguintes aspectos principais:

1. A classificação e a terminologia relativos aos modos de endereçamento variam com frequência de autor para autor, pelo que a compreensão dos mecanismos possíveis para a leitura ou escrita dos dados constitui de facto o aspecto mais importante a realçar. Adoptou-se aqui a mesma aproximação que é seguida na folha de características do componente, pelo que o leitor não encontrará quaisquer dificuldades na respectiva consulta, sendo este o instrumento de trabalho que deve sempre ser privilegiado na realização de qualquer projecto.
2. Vários modos de endereçamento surgem com frequência combinados numa mesma instrução, como se pode verificar pelo facto de algumas das instruções que ilustraram os modos apresentados surgirem em mais do que uma tabela (por exemplo, `ld (ix+20h),5ah` surge em duas tabelas, já que esta instrução usa claramente dois dos modos de endereçamento descritos).

## 3.6 TIPOS DE INSTRUÇÕES

O Z80 disponibiliza várias centenas de instruções (está-se a levar em conta que existem muitas instruções válidas com vários modos de endereçamento), o que torna naturalmente inviável que um projectista as memorize todas. O desenvolvimento de aplicações baseadas neste (ou em qualquer outro) microprocessador requer deste modo que se conheçam, não as instruções individualmente mas sim os seus tipos principais, que agrupam as instruções que realizam operações funcionalmente semelhantes.

A consulta às tabelas que caracterizam as instruções pertencentes a cada tipo permite eliminar dúvidas que surjam na sua utilização, ou mesmo determinar quais as melhores instruções a usar em cada caso. A prática na escrita de programas de aplicação constitui no entanto a melhor ajuda para que o projectista se familiarize rapidamente com as instruções mais comuns, pelo que a consulta destas tabelas acaba posteriormente por assumir apenas um carácter esporádico.

As instruções suportadas pelo Z80 agrupam-se nos onze tipos principais apresentados a seguir:

1. Transferência de dados de 8 bits (*8-bit loads*)
2. Transferência de dados de 16 bits (*16-bit loads*)
3. Instruções de troca, transferência de blocos e pesquisa em blocos (*exchanges, block transfers and searches*)
4. Instruções aritméticas e lógicas para dados de 8 bits (*8-bit arithmetic and logic operations*)
5. Instruções genéricas do tipo aritmético e para controlo do CPU (*general-purpose arithmetic and CPU control*)
6. Instruções aritméticas para dados de 16 bits (*16-bit arithmetic operations*)
7. Instruções de rotação e deslocamento (*rotates and shifts*)
8. Instruções de manipulação ao bit (*bit set, reset and test operations*)
9. Instruções de salto (*jumps*)
10. Chamada de subrotinas e retorno (*calls, returns and restarts*)
11. Instruções de entrada e saída (*input and output operations*)

Não se considere no entanto esta divisão como rígida, quer porque existem algumas instruções cuja funcionalidade levanta dúvidas acerca do grupo em que deveriam ser inseridas, quer porque facilmente encontraremos diferentes divisões, de acordo com a abordagem realizada em cada livro que se debruça sobre este assunto. Neste sentido, optou-se uma vez mais por considerar aqui a divisão que é adoptada nas próprias folhas de características do Z80, o que facilitará as consultas que o leitor necessite posteriormente de efectuar. Cada um destes onze tipos de instruções será agora descrito em maior detalhe, incluindo a

apresentação de exemplos, recomendando-se ao leitor que procure compreender em pormenor cada tipo / exemplo apresentado, já que um bom domínio deste tema facilita substancialmente o desenvolvimento de aplicações concretas, como se verá em capítulos posteriores.

### Transferência de dados de 8 bits

Como o nome indica, este grupo de instruções transfere palavras de 8 bits para os registos ou para a memória. O valor a transferir pode ser indicado imediatamente a seguir à mnemónica da instrução ou então estar residente num outro registo ou numa posição de memória. Dentro deste grupo podem citar-se os exemplos apresentados na tabela 3.13:

Mnemónica	Operação	Flags	Código	#Byte	#M	#T
ld a,b	$a \leftarrow b$	nenhuma	78	1	1	4
ld a,(1000h)	$a \leftarrow (1000h)$	nenhuma	3A 00 10	3	4	13
ld b,(hl)	$b \leftarrow (hl)$	nenhuma	46	1	2	7
ld (ix+20h),c	$(ix+20h) \leftarrow c$	nenhuma	DD 71 20	3	5	19

Tabela 3.13: Exemplo de instruções para transferência de dados de 8 bits.

De acordo com a notação seguida para descrever as operações realizadas por cada instrução, teremos que `ld a,(1000h)` carregará no registo A o conteúdo da posição de memória com endereço 1000h (em hexadecimal). De igual modo, `ld b,(hl)` carrega no registo B o conteúdo da posição de memória apontada pelo par de registos HL (isto é, cujo endereço está contido neste par de registos, com H a conter o *byte* mais significativo e L o menos significativo). Finalmente, a instrução `ld (ix+20h),c` transfere o conteúdo do registo C para posição de memória cujo endereço é determinado somando 20h ao conteúdo do registo IX (que tem 16 bits). Repare-se ainda que o código objecto desta última instrução tem dois *bytes* (DD 71) e que nenhuma das instruções apresentadas afecta qualquer *flag*.

### Transferência de dados de 16 bits

Semelhante ao anterior, este grupo de instruções transfere palavras de 16 bits para os registos ou para a memória. Dentro deste grupo podem citar-se os exemplos apresentados na tabela 3.14:

Mnemónica	Operação	Flags	Código	#Byte	#M	#T
ld hl,1000h	$hl \leftarrow 1000h$	nenhuma	21 00 10	3	3	10
ld hl,(1000h)	$h \leftarrow (1001h)$ $l \leftarrow (1000h)$	nenhuma	2A 00 10	3	5	16
ld sp,hl	$sp \leftarrow hl$	nenhuma	F9	1	1	6
push ix	$(sp-1) \leftarrow ix_H$ $(sp-2) \leftarrow ix_L$ $sp \leftarrow sp-2$	nenhuma	DD E5	2	4	15

Tabela 3.14: Exemplo de instruções para transferência de dados de 16 bits.

Repare-se que uma vez mais nenhuma destas instruções afecta qualquer *flag*. Os comentários apresentados a respeito das instruções anteriores deverão permitir interpretar de forma simples o resultado da execução das que são agora apresentadas nesta tabela 3.14, com excepção da instrução `push ix`. O sentido desta operação tornar-se-à no entanto claro se nos recordarmos dos comentários efectuados a respeito dos registos existentes no Z80, e em particular do que se disse a respeito do apontador para a *stack*. A instrução `push ix` armazena na *stack* o conteúdo do registo IX (que tem 16 bits), sendo o apontador para a *stack* (o SP, *stack pointer*) decrementado duas unidades durante a execução desta instrução. O funcionamento desta instrução pode ilustrar-se se recorrermos de novo à figura 3.11, mas agora considerando que no lugar do PC se encontra o IX.

Para que não se corra o risco de esgotar a área de *stack*, devemos ter sempre o cuidado de recuperar o valor de um registo que lá tenha sido armazenado através da instrução que repõe o valor do SP. A cada instrução de `push` dever-se-à deste modo fazer corresponder uma instrução de `pop` (a instrução `pop ix` recupera o valor armazenado na *stack* e repõe a situação ilustrada à esquerda na figura 3.11).

### Instruções de troca, transferência de blocos e pesquisa em blocos

Este grupo inclui instruções de grande utilidade para a manipulação de tabelas de dados, algumas delas equivalentes mesmo a pequenas subrotinas, como se pode avaliar pelos exemplos apresentados na tabela 3.15:

Mnemónica	Operação	Flags	Código	#Byte	#M	#T
ex de,hl	de $\leftrightarrow$ hl	nenhuma	EB	1	1	4
exx	bc $\leftrightarrow$ bc' de $\leftrightarrow$ de' hl $\leftrightarrow$ hl'	nenhuma	D9	1	1	4
ldir	(de) $\leftarrow$ (hl) de $\leftarrow$ de+1 hl $\leftarrow$ hl+1 bc $\leftarrow$ bc-1 repete até bc=0	nenhuma	ED B0	2  2	5  4	21 se bc $\neq$ 0  16 se bc=0
cpdr	a - (hl) hl $\leftarrow$ hl-1 bc $\leftarrow$ bc-1 repete até a=(hl) ou bc=0	nenhuma	ED B9	2  2	5  4	21 se bc $\neq$ 0 e a $\neq$ (hl) 16 se bc=0 ou a=(hl)

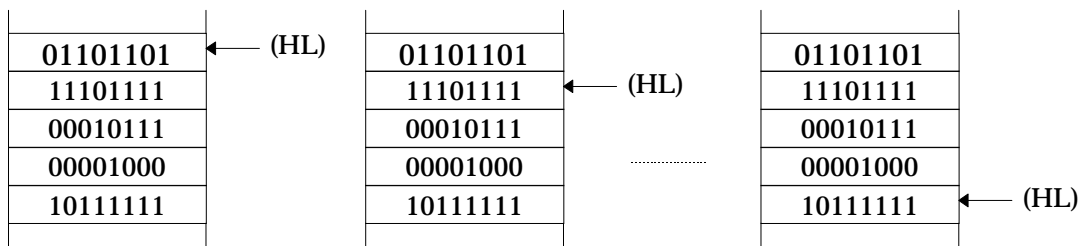
Tabela 3.15: Exemplo de instruções para troca, transferência de blocos e pesquisa em blocos.

Nota: O número de ciclos necessários para a execução das instruções `ldir` e `cpdr` depende naturalmente do conteúdo do par de registos BC, pelo que os valores da coluna “#T” devem ser lidos da seguinte forma: enquanto não se verifica a condição de fim da instrução (BC $\neq$ 0), a duração do ciclo referido na coluna “Operação” vale o maior dos valores indicados em “#T”; na última vez (quando BC chega a 0), vale o menor dos valores indicados em “#T”.

Dos exemplos apresentados merecem especial destaque as instruções `ldir` e `cpdr`, respectivamente destinadas à transferência e à pesquisa em blocos de dados. A instrução `ldir` copia uma tabela com um número de *bytes* que é dado pelo conteúdo de BC (16 bits), a partir de um bloco em memória cuja primeira posição tem por endereço o conteúdo de HL, para um outro bloco cuja primeira posição tem por endereço o conteúdo de DE. A execução desta instrução está ilustrada na figura 3.14, onde se assume que o tamanho da tabela a transferir (isto é, o conteúdo de BC) é de 5 *bytes*.

A instrução `cpdr` tem um funcionamento semelhante ao de `ldir`, excepto que em vez de uma transferência se efectua uma pesquisa. Com efeito, `cpdr` efectua *byte* a *byte* uma comparação entre o conteúdo do registo A e o conteúdo da posição de memória apontada por HL, terminando quando BC chegar a zero (isto é, quando se atingir o fim da tabela) ou quando o conteúdo da posição actual for igual ao conteúdo do registo A (isto é, quando se encontrar o valor procurado).

Bloco origem:



Bloco destino:

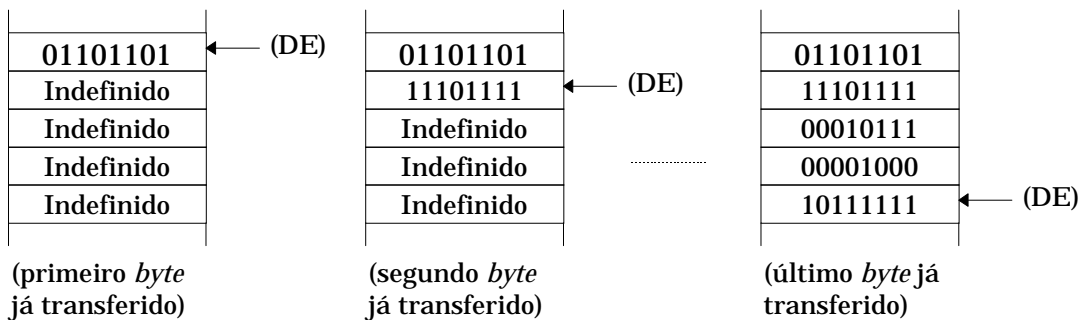


Figura 3.14: Resultado da execução da instrução `ldir` para uma tabela com 5 *bytes*.

### Instruções aritméticas e lógicas para dados de 8 bits

Este grupo inclui as principais instruções aritméticas e lógicas para dados de 8 bits, algumas das quais estão exemplificadas na tabela 3.16:

Mnemónica	Operação	Flags	Código	#Byte	#M	#T
<code>add a,(iy+10h)</code>	$a \leftarrow a + (iy + 10h)$	s, z, h, v, n=0, c	FD 86 10	3	5	19
<code>or (hl)</code>	$a \leftarrow a \vee (hl)$	s, z, h=0, p, n=0, c=0	B6	1	2	7
<code>cp e</code>	$a - e$	s, z, h, v, n=1, c	BB	1	1	4
<code>dec (hl)</code>	$(hl) \leftarrow (hl) - 1$	s, z, h, v, n=1	35	1	3	11

Tabela 3.16: Exemplo de instruções aritméticas e lógicas para dados de 8 bits.

Repare-se que a instrução `cp e` efectua a comparação entre o conteúdo dos registos A e E, realizando para tal uma subtracção dos seus conteúdos, como se indica na operação correspondente. Esta subtracção não altera no entanto o conteúdo de nenhum destes registos, como se infere igualmente pela descrição da operação efectuada ( $a - e$  apenas, em vez de  $a \leftarrow a - e$ ).

### Instruções genéricas do tipo aritmético e para controlo do CPU

Para além de algumas instruções que poderiam eventualmente fazer parte do grupo anterior, este grupo inclui outras instruções que afectam directamente o



estado interno do CPU, tal como se ilustra nos exemplos apresentados na tabela 3.17:

Mnemónica	Operação	Flags	Código	#Byte	#M	#T
ccf	$cy \leftarrow /cy$	n=0, c	3F	1	1	4
halt	pára o CPU	nenhuma	76	1	1	4
di	$iff1 \leftarrow 0$	nenhuma	F3	1	1	4
im 1	atende /INT em modo 1	nenhuma	ED 56	2	2	8

Tabela 3.17: Exemplo de instruções genéricas do tipo aritmético e para controlo do CPU.

Repare-se em particular no exemplo da instrução *di*, que inibe o atendimento aos pedidos de interrupção efectuados através da entrada /INT do microprocessador, colocando para tal em 0 o *flip-flop* que habilita o atendimento (IFF1). Existe naturalmente também a instrução *ei*, que habilita o atendimento das interrupções quando tal for pretendido. O processamento dos pedidos de interrupção no pino /INT foi já anteriormente discutido em secção própria, quando se efectuou a apresentação dos três modos possíveis de atendimento. A instrução *im 1* selecciona o modo 1, sendo os outros dois modos seleccionados pelas instruções *im 0* e *im 2*.

### Instruções aritméticas para dados de 16 bits

Este grupo inclui instruções semelhantes às que foram apresentadas já anteriormente para os dados de 8 bits, com a diferença adicional de que para dados de 16 bits não existem instruções lógicas mas apenas aritméticas. Dentro deste grupo podem citar-se os exemplos apresentados na tabela 3.18:

Mnemónica	Operação	Flags	Código	#Byte	#M	#T
adc hl,de	$hl \leftarrow hl+de+cy$	s, z, v, n=0, c	ED 5A	2	4	15
sbc hl,de	$hl \leftarrow hl-de-cy$	s, z, h=0, p, n=1, c=0	ED 52	2	4	15
inc ix	$ix \leftarrow ix+1$	nenhuma	DD 23	2	2	10
dec iy	$iy \leftarrow iy-1$	nenhuma	FD 2B	2	2	10

Tabela 3.18: Exemplo de instruções aritméticas para dados de 16 bits.

### Instruções de rotação e deslocamento

As instruções de rotação e deslocamento podem ser realizadas sobre os registos ou sobre o conteúdo de posições de memória, apresentando-se na tabela 3.19 alguns exemplos deste grupo.

Mnemónica	Operação	Flags	Código	#Byte	#M	#T
rla	$cy \leftarrow \text{bit}[7],$ $\text{bit}[i] \leftarrow \text{bit}[i-1],$ $\text{bit}[0] \leftarrow cy$ (registo A)	$h=0, n=0, c$	17	1	1	4
rlc (ix+20h)	$cy \leftarrow \text{bit}[7],$ $\text{bit}[i] \leftarrow \text{bit}[i-1],$ $\text{bit}[0] \leftarrow \text{bit}[7]$ (em memória)	$s, z, h=0, p,$ $n=0, c$	DD CB 20 06	4	6	23
sla (hl)	$cy \leftarrow \text{bit}[7],$ $\text{bit}[i] \leftarrow \text{bit}[i-1],$ $\text{bit}[0] \leftarrow 0$ (em memória)	$s, z, h=0, p,$ $n=0, c$	CB 26	2	4	15
rrd	$a[3:0] \leftarrow (hl)[3:0]$ $(hl)[3:0] \leftarrow (hl)[7:4]$ $(hl)[7:4] \leftarrow a[3:0]$ (registo A e em memória)	$s, z, h=0, p,$ $n=0, c$	ED 67	2	5	18

Tabela 3.19: Exemplo de instruções de rotação e deslocamento.

A notação  $(hl)[3:0]$  representa os bits 3 a 0 (os quatro bits menos significativos) da posição de memória cujo endereço é dado pelo conteúdo de HL. De um modo geral, as instruções de rotação efectuem uma rotação circular do operando (isto é, o que sai por um lado — esquerda ou direita — entra pelo outro, eventualmente através da *flag* de *carry*), enquanto nas instruções de deslocamento isto não sucede. A figura 3.15 ilustra esta diferença através de duas das instruções apresentadas acima, `rla` e `sla (hl)`.

Repare-se finalmente que a instrução `rlc (ix+20h)` ocupa 4 *bytes* em memória, sendo este o número máximo de *bytes* ocupados por uma só instrução (o que sucede também com várias outras instruções, deste e de outros grupos).

RLA - Rotação:



SLA (HL) - Deslocamento:

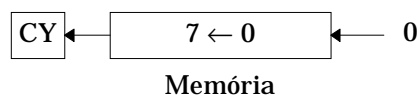


Figura 3.15: Exemplo de instruções de rotação e deslocamento.

### Instruções de manipulação ao bit

Este grupo inclui as instruções que actuam directamente sobre bits individuais, permitindo efectuar operações de teste, colocação em 1 e colocação em 0. Dentro deste grupo podem citar-se os exemplos apresentados na tabela 3.20:

Mnemónica	Operação	Flags	Código	#Byte	#M	#T
bit 5,a	$z \leftarrow /a[5]$	z, h=1, n=0	CB 6F	2	2	8
bit 5,(hl)	$z \leftarrow /(hl)[5]$	z, h=1, n=0	CB 6E	2	3	12
set 3,(ix+7ah)	$(ix+7ah)[3] \leftarrow 1$	nenhuma	DD CB 7A DE	4	6	23
res 7,e	$e[7] \leftarrow 0$	nenhuma	CB BB	2	2	8

Tabela 3.20: Exemplo de instruções de manipulação ao bit.

A notação  $/(hl)[5]$  representa a negação do bit 5 da posição de memória apontada por HL. Repare-se que o teste de um bit corresponde a colocar em 1 a *flag* z (activar a *flag* de zero) se este bit for 0 e vice-versa (colocar em 0 se for 1), como se pode verificar pela operação indicada para a instrução bit 5,(hl).

### Instruções de salto

Este grupo contem as instruções de salto condicional e incondicional que permitem redireccionar o fluxo de execução de um programa. Dentro deste grupo podem citar-se os exemplos apresentados na tabela 3.21:

Mnemónica	Operação	Flags	Código	#Byte	#M	#T
jp 20ffh	$pc \leftarrow 20ffh$	nenhuma	C3 FF 20	3	3	10
jp z,1000h	se z=1 $pc \leftarrow 1000h$ se não continua	nenhuma	CA 00 10	3	3	10
jr z,marca	se z=1 $pc \leftarrow marca$ se não continua	nenhuma	28 05	2	3	12
				2	2	se z=1 7 se z≠1
jp (hl)	$pc \leftarrow hl$	nenhuma	E9	1	1	4

Tabela 3.21: Exemplo de instruções de salto.

Repare-se que os exemplos considerados na tabela 3.21 incluem saltos condicionais e não condicionais, que ocupam entre 1 a 3 posições de memória, de acordo com o modo de endereçamento empregue.

### Chamada de subrotinas e retorno

As instruções de chamada de subrotinas e retorno são semelhantes às de salto, na medida em que permitem redireccionar o fluxo de execução de um programa, mas apresentam em relação àquelas a diferença de afectarem o conteúdo da *stack*. Com efeito, e como se pode ver nos exemplos apresentados na tabela 3.22, todas elas acedem à *stack* para guardar ou extrair o valor do apontador de programa (PC), actualizando o apontador para a *stack* (SP, *stack pointer*) em conformidade.

Mnemónica	Operação	Flags	Código	#Byte	#M	#T
call 1000h	$(sp-1) \leftarrow pc_H$ $(sp-2) \leftarrow pc_L$ $pc \leftarrow 1000h$ $sp \leftarrow sp-2$	nenhuma	CD 00 10	3	5	17
ret nz	se $z=0$ $pc_L \leftarrow (sp)$ $pc_H \leftarrow (sp+1)$ $sp \leftarrow sp+2$ se não continua	nenhuma	C0	1  1	1  3	5 se $z \neq 0$ 11 se $z=0$
reti	$pc_L \leftarrow (sp)$ $pc_H \leftarrow (sp+1)$ $sp \leftarrow sp+2$	nenhuma	ED 4D	2	4	14
rst 20h	$(sp-1) \leftarrow pc_H$ $(sp-2) \leftarrow pc_L$ $sp \leftarrow sp-2$ $pc_H \leftarrow 0$ $pc_L \leftarrow 20h$	nenhuma	E7	1	3	11

Tabela 3.22: Exemplo de instruções para chamada de subrotinas e retorno.

A actualização do valor do SP tem lugar durante a execução destas instruções, sendo o SP decrementado nas instruções de `call` e incrementado nas de `ret`. Esta actualização tem lugar exactamente da mesma forma que se ilustrou anteriormente na figura 3.11, quando se descreveu a resposta do Z80 aos pedidos de interrupção.

O efeito das instruções `ret` é naturalmente o oposto ao das instruções `call`, de modo a que o SP retorne ao valor que tinha antes de se efectuar a chamada à subrotina. Este efeito está ilustrado pelas operações indicadas na tabela 3.22 para os exemplos das instruções `ret nz` e `reti` (respectivamente retorno se a *flag* *z* não estiver activa e retorno de interrupção em `/INT`), que incrementam o SP durante a sua execução. Refira-se que o retorno de uma interrupção pedida em `/NMI` deve sempre efectuar-se através da instrução `retn`, já que o respectivo atendimento coloca em 0 o IFF1 (*interrupt flip-flop*, apresentado quando se efectuou a descrição do processamento dos pedidos de interrupção, na secção 3.4), cujo valor “pré-/NMI” terá que ser respondido.

A instrução `rst 20h` pertence ao conjunto de 8 instruções a que se dá habitualmente a designação de *restart* (`rst 0h`, `rst 08h`, ..., `rst 38h`) e que permitem a chamada de subrotinas cujo endereço de entrada tem o *byte* mais significativo em 00 (daí a designação frequente de “página 0 no espaço de

memória”). Estas instruções foram já anteriormente consideradas quando se descreveu o modo de endereçamento *Página zero modificado*.

### Instruções de entrada e saída

O último grupo que nos falta apresentar é o que diz respeito às instruções de entrada e saída. Como foi já anteriormente referido, este microprocessador dispõe dos pinos /MREQ e /IORQ para distinguir entre ciclos de acesso a memória e ciclos de acesso a entrada / saída. As instruções de entrada / saída activam o sinal de /IORQ durante a respectiva execução, podendo referir-se neste grupo os exemplos apresentados na tabela 3.23:

Mnemónica	Operação	Flags	Código	#Byte	#M	#T
in a,(7fh)	$a \leftarrow (7fh)$	nenhuma	DB 7F	2	3	11
inir	$(hl) \leftarrow (c)$ $b \leftarrow b-1$ $hl \leftarrow hl+1$ repete até $b=0$	$z=1, n$	ED B2	2	5	21 se $b \neq 0$
				2	4	16 se $b=0$
out (20h),a	$(20h) \leftarrow a$	n	D3 20	2	3	11
otir	$(c) \leftarrow (hl)$ $b \leftarrow b-1$ $hl \leftarrow hl+1$ repete até $b=0$	$z=1, n$	ED B3	2	5	21 se $b \neq 0$
				2	4	16 se $b=0$

Tabela 3.23: Exemplo de instruções de entrada e saída.

Repare-se que os endereços de acesso a dispositivos de entrada / saída são codificados em apenas um *byte*, como já tinha sido anteriormente referido quando se apresentou o diagrama temporal para este tipo de ciclos. Vale também a pena chamar a atenção do leitor para a possibilidade de, com uma única instrução (*inir*), se poder construir uma tabela em memória com um conjunto de valores adquiridos consecutivamente numa entrada (o tamanho da tabela corresponde ao conteúdo do registo B). Também a instrução complementar está disponível (*otir*), neste caso para transferir o conteúdo de uma tabela para um dispositivo de saída (o registo C contém o endereço de entrada / saída a usar em ambos os casos).

Uma vez estudados os conceitos principais relativos ao microprocessador Z80, concluiremos este capítulo com a apresentação de um conjunto de fontes de informação adicionais às quais o leitor poderá recorrer para complementar qualquer dos assuntos tratados neste capítulo.

### 3.7 OUTRAS FONTES DE INFORMAÇÃO

Para além dos catálogos de microprocessadores e periféricos da Zilog, de consulta necessariamente obrigatória, poder-se-à igualmente aceder a muita desta informação através da página desta empresa no WWW, e em particular através do URL <http://www.zilog.com/datacom.html>, que disponibiliza a informação relacionada com os assuntos estudados neste capítulo.

O URL <http://www.comlab.ox.ac.uk/archive/cards.html> permite o acesso a uma apresentação uniformizada das instruções de um grande número de microprocessadores, incluindo o Z80, cuja tabela “normalizada” de instruções pode aceder-se directamente via <ftp://ftp.comlab.ox.ac.uk/pub/Cards/Z80>.

A maioria dos fabricantes de sistemas de desenvolvimento publicitam os seus produtos em revistas da especialidade (tal como a *Electronic Product News*, de distribuição gratuita), sendo normalmente simples a obtenção de informações técnicas por esta via (as próprias revistas incluem cupões postais para solicitar a informação sobre os produtos seleccionados). Começa também a ser comum que os fabricantes mais importantes destes sistemas disponibilizem informações e mesmo versões de demonstração através do WWW, como sucede com a Nohau, através do URL <http://www.vena.com/nohau/>.

Existem naturalmente muitos livros escritos sobre o Z80, entre os quais poderemos destacar os seguintes:

1. Ramesh Gaonkar, *The Z80 Microprocessor: Architecture, Interfacing, Programming and Design*, Prentice-Hall Macmillan, 1993, ISBN 0-02-340484-1 (700 páginas).
2. John Uffenbeck, *Microprocessors and Microcomputers: The 8080, 8085 and Z-80. Programming, Interfacing and Troubleshooting*, Prentice-Hall, 1985, ISBN 0-13-580051-X (690 páginas).