STEVE WEBB

MSX

PROGRAMAS EM LINGUAGEM DE MÁQUINA

EDITORA MANOLE LTDA.

MSX

PROGRAMAS EM LINGUAGEM DE MÁQUINA

STEVE WEBB

MSX

PROGRAMAS EM LINGUAGEM DE MÁQUINA

EDITORA MANOLE
1986

Traduzido do Original Inglês
Pratical **MSX** Machine Code Programming

Tradução:
Maria da Graça Brasil Rocha

Revisão de Texto:
Orlando Parolini

Editor:
Roberto Manole

Capa:
Jerardo Cofré

Composição:
Comarte Comunicação e Arte Ltda

EDITORA MANOLE LTDA.
Rua 13 de Maio, 1.026 – Bela Vista
CEP 01327 – São Paulo – SP
Caixa Postal 1489 – telefone 287-0746

Impresso no Brasil
Printed in Brazil

ÍNDICE

NOTA DO AUTOR

Este livro é somente uma introdução à programação em linguagem de máquina. Um grande número de códigos de máquina do microprocessador Z80 não foram descritos. O sentido desses poderá ser achado em uma das numerosas obras especializadas que são dedicadas exclusivamente a esses códigos. O leitor que começa a escrever os seus próprios programas em linguagem de máquina poderá encontrar certas dificuldades cujas soluções não são descritas nesse livro de iniciação. O leitor precisa saber que a finalidade dos programas apresentados no livro é essencialmente didática. Os programas não foram escritos com a finalidade de autopromoção mas para serem facilmente entendidos.

STEVE WEBB
janeiro 1985

INTRODUÇÃO

Este livro é uma introdução à programação em linguagem de máquina para os computadores MSX. Ele foi concebido para permitir um aprendizado progressivo e, portanto, é necessário uma leitura linear da obra. É aconselhável não abordar um capítulo antes de haver assimilado o conteúdo dos capítulos precedentes.

A programação em linguagem de máquina não é assim tão difícil como se diz algumas vezes. Tendo um mínimo de prática na linguagem BASIC, os conceitos utilizados para a programação em linguagem de máquina podem ser assimilados muito rapidamente.

Os primeiros capítulos desta obra tratam da teoria da linguagem de máquina mostrando, particularmente, como são armazenados os números. A descrição de instruções fundamentais equivalentes a numerosas instruções BASIC, e da organização interna de um computador MSX, permite, em seguida, encontrar elementos para responder à questão: "O que é a linguagem de máquina? ". O capítulo principal mostra como se escreve, nesta linguagem, um programa de jogo muito simples, começando por estabelecer o fluxograma pelo qual cada bloco é convertido, em seguida, em pequenos subprogramas em linguagem de máquina.

Os últimos capítulos descrevem alguns subprogramas utilitários e mostram como se pode tirar partido das possibilidades derivadas dos computadores MSX. No decorrer desta obra, algumas questões permitem ao leitor exercitar os novos conhecimentos adquiridos. As respostas a estas questões são dadas no final do livro. Ao responder erradamente, é aconselhável retomar a leitura do capítulo correspondente, até que seja capaz de fornecer a resposta certa.

O PADRÃO MSX

No momento em que esta obra foi escrita, o padrão MSX foi adotado por mais de vinte e cinco construtores de computadores. A definição de um padrão constitui uma etapa importante no desenvolvimento da microinformática e oferece garantias fundamentais para o usuário e também para o programador. A regra diz respeito tanto à parte material (hardware) como à parte lógica (software).

Suponhamos que um programa escrito em BASIC ou em linguagem de máquina tenha sido copiado em fita, para um computador MSX de marca Sony. O padrão permite utilizar esta fita e o programa que ela contém em qualquer outra marca de computador MSX. O programa rodará corretamente sem necessitar de nenhuma modificação, tendo como única condição, naturalmente, que ele tenha sido escrito respeitando as especificações do padrão.

Esta compatibilidade, ou seja, possibilidade de intercambiar um sistema lógico para computadores de marcas diferentes, supõe igualmente uma padronização máxima ao nível material e ao nível do *firmware* (possibilidade de programar cartuchos equipados de EPROM). Os vários sistemas são de fato sempre armazenados nos mesmos locais de memória, de um computador MSX para outro, e as saídas para o vídeo são necessariamente as mesmas. Nestas condições, por que escolher uma marca de computador MSX ao invés de outra? No que diz respeito à alta fidelidade, a escolha pode ser em parte guiada por critérios mais ou menos subjetivos, tais como o prestígio ou apego a uma marca ou a estética do material proposto. Entretanto, existem outros critérios (pelo menos tão importantes) que podem motivar a escolha. Certos fabricantes podem oferecer um produto de características suplementares, não impostas pela norma, tal como uma interface para *light pen*, por exemplo. É importante lembrar que estas características suplementares não estão necessariamente disponíveis para outros computadores do mesmo padrão.

Um programa que foi escrito para ser utilizado com uma *light pen* não rodará naturalmente em um computador MSX que não disponha desta saída. Deve-se notar, entretanto, que o padrão define uma normalização para a maior parte das unidades periféricas essenciais, tal como as impressoras, por exemplo.

Uma coisa importante não é definida pelo padrão: trata-se do tamanho da memória (existe, de fato, uma norma, mas de tamanho tão pequeno que a maior parte dos construtores propõe uma maior). Pode-se, assim, encontrar computadores MSX de 32, 48 ou 64K, conforme o modelo e a marca. Um programa escrito para um computador MSX rodará em qualquer outro computador que disponha pelo menos da mesma capacidade de memória (poderá eventualmente rodar em computador dispondo de um tamanho menor de memória desde que não ultrapasse a capacidade da mesma). Pode-se pensar que 64K tornar-se-á rapidamente a norma dos computadores MSX.

Ressaltamos, enfim, que MSX é um padrão universal. Um programa escrito para um computador MSX vendido na França, rodará igualmente em um computador deste modelo vendido no Japão ou em Papua – Nova Guiné.

1

A LINGUAGEM DE MÁQUINA

A unidade de tamanho da memória central de um computador é o kilo byte (ou K). Um kilo byte corresponde a um pouco mais de 1000 bytes (1 024 exatamente), de forma que um computador de 64K dispõe de um pouco mais de 64 000 posições de memória (ou endereços) diferentes. Cada posição de memória pode ser considerada como uma caixa contendo um número compreendido entre 0 e 65535. Estes números são virtuais e servem unicamente para indicar cada um dos endereços. Logo que o computador é ligado, apenas um pouco mais de 28 000 destas posições de memória podem ser utilizadas para programas em linguagem BASIC, ou programas em linguagem de máquina. De fato, 4 000 bytes são reservados para as variáveis do sistema. Para os computadores dispondo de 64K, 32K bytes suplementares estão disponíveis e se juntam aos 28K descritos anteriormente. Entretanto, estes 32K suplementares não são acessíveis senão a programadores experientes em linguagem de máquina que ultrapassam o objetivo desta obra de iniciação. Todos os programas dados aqui podem ser rodados em computadores MSX dispondo de pelo menos 32K de memória.

Cada posição de memória pode conter um número compreendido entre 0 e 255 inclusive. O fato de não poder armazenar um número superior a 255 constitui naturalmente uma limitação, de modo que foi pesquisado um método para armazenar números muito maiores. O exemplo abaixo ilustra o método adotado.

Suponhamos que o número 29 248 deva ser armazenado na memória central de um computador. A primeira operação a ser efetuada consiste em dividir este número por 256, e separar a parte inteira desta divisão. Assim, 29 248 dividido por 256 dá 114,25; o valor 114 é então armazenado. Ele é chamado parte fixa do número a ser armazenado e representa o número de múltiplos inteiros de 256 que ele contém. Multiplicando a parte fixa do número por 256 e subtraindo o resultado obtido do número a ser armazenado obtém-se o resto da divisão inteira precedente:

$$114 \times 256 = 29\,184$$
$$29\,248 - 29\,184 = 64$$

64 chama-se a parte flutuante deste número.

O número 29 248 será armazenado na memória carregando a parte flutuante (64) em um endereço de memória escolhido e a parte fixa (114) no endereço seguinte. O armazenamento de um número superior a 255 necessita então a utilização de duas posições de memória. Assim, logo que se lê que um número maior que 255 está armazenado, por exemplo, no endereço 50000, isto significa que ele ocupa de fato os endereços 50000 e 50001.

Quais são as partes fixas e flutuantes do número 45 621?

Qual é o número que tem 31 como parte flutuante e 64 como parte fixa?

Cada posição de memória que pode conter um número compreendido entre 0 e 225 corresponde a um byte. Um programa tendo tamanho de 5 000 bytes corresponde assim a 5 000 posições de memória.

A parte da memória central onde são armazenados os programas utilitários (que atuam sobre os programas escritos em BASIC ou em linguagem de máquina) é denominada RAM (*Random Access Memory*/ memória de acesso aleatório). O conteúdo da memória RAM é apagado logo que o computador é desligado. Existe uma outra parte da memória central chamada ROM (*Read Only Memory* / memória somente de leitura). Como seu nome indica, é possível somente ler o conteúdo dos endereços correspondentes e não modificá-los. Este conteúdo não se apaga quando o computador é desligado, o que é normal porque se encontra em particular neste local o sistema monitor, sem o qual o computador não seria mais do que um bloco de metal inerte. O sistema monitor contém um certo número de subprogramas em linguagem de máquina que geram as funções principais do computador; eles permitem, por exemplo, ler o teclado, efetuar os cálculos fundamentais e verificar a sintaxe dos programas escritos em BASIC.

O coração do computador é seu microprocessador (do tipo Z80). Ele não tem a inteligência de um cérebro. A única coisa que ele pode fazer são as operações aritméticas totalmente elementares. Sua vantagem em relação ao cérebro humano reside no fato de que o microprocessador efetua um número muito grande de operações elementares por segundo, o que pode algumas vezes lhe dar a aparência de uma certa "inteligência".

O microprocessador só compreende instruções em linguagem de máquina. Existem mais de 600 variações possíveis para estas instruções. Cada instrução corresponde a um número ou a uma combinação de números. Por exemplo, o número 198 corresponde à instrução que manda adicionar dois números, de modo que, logo que o microprocessador encontra o número 198, ele sabe que a operação que deve ser imediatamente efetuada é a adição (o procedimento que ele utiliza para realizar esta operação será descrito mais à frente neste livro).

De que modo o microprocessador, não compreendendo senão a linguagem de máquina, pode intervir na execução dos programas escritos em BASIC? A resposta a esta questão remete à existência de um interpretador, equivalente na informática a um intérprete de uma língua estrangeira. Da mesma forma que um intérprete tem por função traduzir um texto de uma língua para outra, o BASIC se comunica com o microprocessador por intermédio de um interpretador. Conclui-se facilmente que o recurso de um intérprete ou de um interpretador tem por inconveniente diminuir o ritmo da conversação entre duas pessoas ou entre o programa BASIC e o microprocessador.

A possibilidade de programar em linguagem de máquina permite, ao tornar direto o acesso ao microprocessador, dispensar-se o interpretador, o que aumenta a rapidez da execução de um programa. De modo geral, um programa escrito em linguagem de máquina é executado 50 a 100 vezes mais rápido que o programa equivalente escrito em BASIC.

2

EQUIVALÊNCIA DE ALGUMAS INSTRUÇÕES BASIC EM LINGUAGEM DE MÁQUINA

Certos comandos BASIC tais como LIST, NEW, RENUM, DELETE e AUTO não têm nenhuma razão de existir em linguagem de máquina e não têm, portanto, equivalentes. Outras instruções BASIC tais como RUN, READ, DATA, PRINT, VPOKE e VPEEK, não têm realmente equivalentes em linguagem de máquina, mas veremos que é possível simular sua ação por meio de certos códigos (ou instruções em linguagem de máquina).

Os códigos aqui apresentados são os mais freqüentemente utilizados. Seu conhecimento é suficiente para escrever pequenos programas em linguagem de máquina.

Em BASIC, está-se habituado a utilizar variáveis tais como A, B, C, ..., X, Y, Z. Em linguagem de máquina, tais variáveis não existem. Seus equivalentes mais próximos são os registradores. Existe um pequeno número de registradores e os que são mais utilizados são definidos pelas letras:

A, B, C, D, E, H, L

Um outro registrador definido por F, será descrito um pouco mais adiante. Cada registrador pode ser considerado como representante de uma posição de memória; assim, não podem conter senão números compreendidos entre 0 e 255. Para poder utilizar os registradores para armazenar números maiores, seis dentre eles foram agrupados em pares, da seguinte maneira:

HL
BC
DE

Estes reagrupamentos não impedem a utilização de cada um destes registradores individualmente.

A instrução BASIC:

LET A = 5

tem por código de máquina equivalente:

LD A,5

LD é uma abreviação da palavra LOAD (carregar). O verdadeiro significado do código acima é "Carregar o registrador A com o valor 5".

Cada um dos outros registradores pode, deste modo, receber um valor compreendido entre 0 e 255:

```
LD H, 199 (Carregar o registrador H com o valor 199)
LD D,2    (Carregar o registrador D com o valor 2)
```

Foi mencionado anteriormente que cada registrador podia ser considerado como representante de um posição de memória. Foi igualmente dito que um número como 827 tinha o valor 3 (parte inteira da divisão de 827 por 256) como parte fixa e o valor 59 como parte flutuante. Logo que o código de máquina LD HL,827 é executado, a parte fixa do número 827 é carregada para o registrador H e a parte flutuante para o registrador L (H e L são, aliás, as iniciais das palavras *high* (alto) e *low* (baixo). Da mesma forma, no registrador duplo BC, B é a parte alta enquanto que no par DE, E é a parte baixa.

BASIC: LET A=B
Significado: Atribuir à variável A, o mesmo valor armazenado na variável B.
Código de máquina: LD A,B
Significado: Carregar no registrador A o mesmo valor contido no registrador B.

Assim, é possível carregar para um regsitrador simples o mesmo valor que está contido em qualquer outro registrador simples. Deste modo, pode-se ter os seguintes códigos:

```
LD A,H
LD E,A
LD H,C
```

O código de máquina LD é provavelmente o mais utilizado (é mesmo difícil escrever um programa em código de máquina sem fazer uso do mesmo).

BASIC: LET A = A + 5
Significado: Adicionar 5 ao valor corrente de A.
Código de máquina: ADD A,5
Significado: Aumentar em 5 o conteúdo do registrador A.

O registrador A é o único por meio do qual é possível aumentar de modo direto o valor nele contido. Os códigos de máquina seguintes são, por exemplo, impossíveis:

```
ADD B,9
ADD E,3
```

Esta limitação aparente do microprocessador Z80 pode ser facilmente contornada, como descrito mais adiante.

BASIC: LET A = A + B.
Significado: Adicionar o valor de B ao de A e armazenar o valor obtido em A.
Código de máquina: ADD A,B
Significado: Aumentar o valor corrente do registrador A por aquele contido no registrador B.

Assim, é possível adicionar ao valor corrente do registrador A o conteúdo de qualquer um dos sete registradores anteriormente mencionados:

```
ADD A,B
ADD A,C
ADD A,H
ADD A,L
ADD A,D
ADD A,E
ADD A,A
```

O último código, ADD A,A, tem por efeito duplicar o valor corrente do registrador A. Qualquer outra combinação que não as indicadas acima é proibida; é o caso, por exemplo, dos seguintes códigos:

```
ADD B,H
ADD D,C
```

Os registradores podem ser igualmente adicionados. Entretanto, as únicas combinações permitidas são as seguintes:

```
ADD HL,BC
ADD HL,DE
ADD HL,HL
```

O resultado deve, então, ser sempre armazenado em HL. O último código, ADD HL,HL, tem por efeito duplicar o valor contido no registrador HL. Não é possível adicionar de modo direto o valor contido em um registrador simples ao de um registrador duplo.

Para adicionar um número ao conteúdo de um outro registrador que não o registrador A, o procedimento a ser seguido é o seguinte: suponha, por exemplo, que se deseja adicionar 9 ao valor corrente 14 do registrador B.

Etapa 1: **LD A,B**
O registrador A tem nesse caso o mesmo valor que o contido em B.

Etapa 2: **ADD A,9**
O conteúdo do registrador A é aumentado de 9.

Etapa 3: **LD B,A**
O resultado da operação é transferido para o registrador B.

Assim, é possível por meio de três instruções, adicionar um número ao valor contido em qualquer um dos sete registradores.

Uma outra limitação do microprocessador Z80 reside no fato de que só é possível adicionar ao registrador A, o conteúdo de um outro registrador. Assim, não se pode escrever:

```
ADD B,H
```

Esta restrição pode ser contornada da seguinte maneira: suponhamos que B tivesse valor 5; H valor 7 e que se desejasse somar o conteúdo destes dois registradores e armazenar o resultado em B:

Etapa 1: **LD A,B**
O registrador A tem, portanto, o mesmo valor do registrador B.

Etapa 2: **ADD A,H**
A soma é efetuada no registrador A.

Etapa 3: **LD B,A**
O resultado é transferido para o registrador B.

Novamente, três etapas foram suficientes para somar o conteúdo de dois registradores quaisquer. Também é possível somar desta maneira o valor de um registrador a ele mesmo.

Do mesmo modo, pode-se somar o conteúdo de dois registradores duplos quaisquer. Para adicionar, por exemplo, o conteúdo do registrador BC ao de DE, e para que o resultado deva ser armazenado em BC, as três etapas seguintes devem ser efetuadas:

Etapa 1: **LD H,B**
LD L,C
O registrador duplo HL toma, deste modo, o valor do registrador BC.

Etapa 2: **ADD HL,DE**
A soma é efetuada em HL.

Etapa 3: **LD B,H**
LD C,L
O resultado é transferido para o registrador duplo BC.

A maioria das limitações do microprocessador Z80 podem ser, assim, facilmente contornadas; um mínimo de prática permite selecionar o método mais eficaz para responder a uma necessidade específica.

BASIC: **LET A = A – 5**
Significado: **Subtrair 5 ao valor corrente da variável A.**
Código de máquina: **SUB A,5**
Significado: **Subtrair 5 ao valor contido no registrador A.**

Como no caso da soma, o registrador A é o único pelo qual se pode subtrair diretamente um número. No caso dos outros seis registradores, o procedimento a ser empregado para efetuar esta operação é o seguinte: suponha, por exemplo, que o valor 5 deva ser subtraído do registrador D:

Etapa 1: **LD A,D**
O registrador A toma, deste modo, o valor de D.

Etapa 2: **SUB A,5**
A subtração é efetuada no registrador A.

Etapa 3: **LD D,A**
O resultado da operação é transferido para o registrador D.

BASIC: **LET A = A – B**
Significado: **Subtrair B de A e armazenar o resultado em A.**
Código de máquina: **SUB B**
Significado: **Subtrair o valor do registrador B do valor do registrador A.**

Agora, o registrador A é o único pelo qual é possível subtrair o valor contido em outro registrador.

Para subtrair o valor de um registrador de outro que não o registrador A, é necessário passar por três etapas semelhantes àquelas descritas anteriormente.

Do mesmo modo, é possível subtrair o conteúdo de um registrador duplo de um outro. Entretanto, as únicas combinações permitidas são as seguintes:

```
SBC HL,BC
SBC HL,DE
SBC HL,HL
```

SBC é uma forma particular da subtração. A instrução SUB não pode ser utilizada para registradores duplos no microprocessador Z80. O significado real do código SBC será dado um pouco mais adiante. Por ora, é suficiente considerá-lo como um simples operador de subtração. Uma instrução muito utilizada, INC, permite adicionar (INCrementar) uma unidade ao conteúdo de um registrador simples ou de um registrador duplo. As combinações possíveis são as seguintes:

```
INC A
INC B
INC C
INC D
INC E
INC H
INC L
INC HL
INC BC
INC DE
```

De modo semelhante, a instrução DEC permite subtrair (de DECrémenter, palavra que significa diminuir em francês) uma unidade ao conteúdo de qualquer um dos registradores (simples ou duplos).

BASIC: GOTO (número da linha).
Significado: Desvio para a linha especificada.
Código de máquina: JP (endereço de memória).
Significado: Ir para o endereço de memória especificado.

O código de máquina JP tem uma função muito próxima da instrução BASIC "GOTO", o endereço de memória substitui aqui o número da linha. O capítulo seguinte mostrará como os códigos de máquina são armazenados na memória. A utilização deles, então, tornar-se-á muito clara.

BASIC: GOSUB (número da linha).
Significado: Ligação com um subprograma onde o número da primeira linha está especificado. O subprograma deve terminar com uma instrução RETURN.
Código de máquina: CALL (endereço de memória).
Significado: Ligação com um subprograma começando no endereço especificado. Como os subprogramas em BASIC, os escritos em linguagem de máquina devem terminar com uma instrução RET.
BASIC: IF A=5 THEN (GOTO/GOSUB/LET/etc.).
Significado: Se A=5, então executar a instrução ou o desvio especificado pela cláusula THEN.

Em linguagem de máquina, o equivalente à instrução IF...THEN existe, mas não é implantada da mesma maneira que em BASIC.

A condição IF se refere sempre, em linguagem de máquina, ao resultado da última operação que tenha sido efetuada. O equivalente típico de uma instrução IF...THEN é:

CALL Z, (endereço de memória especificado)

Esta instrução significa que, se o resultado do último cálculo é zero, o subprograma que se encontra no endereço especificado deve ser chamado.

A lista dos códigos de máquina equivalente às instruções IF...THEN mais freqüentemente utilizadas é dada logo abaixo:

CALL NZ,nn
Se o resultado do último cálculo efetuado é diferente de zero, o subprograma que se encontra no endereço nn, é chamado.

CALL M,nn
Se o resultado do último cálculo efetuado é negativo, o subprograma que se encontra no endereço nn é chamado.

CALL P,nn
Se o resultado do último cálculo efetuado é positivo, o subprograma que se encontra no endereço nn é chamado.

JP Z,nn
Se o resultado do último cálculo efetuado é zero, a execução do programa é desviada para o endereço nn.

JP NZ,nn
Se o resultado do último cálculo efetuado é diferente de zero, a execução do programa é desviada para o endereço nn.

JP M,nn
Se o resultado do último cálculo efetuado é negativo, a execução do programa é desviada para o endereço nn.

JP P,nn
Se o resultado do último cálculo efetuado é positivo, a execução do programa é desviada para o endereço nn.

BASIC: FOR A = 1 TO 100
A(s) instrução(ões) deve(m) ser executada(s) 100 vezes.
NEXT A
Significado: Efetuar 100 vezes a(s) instrução(ões).
Código de máquina: LD A,100
A(s) instrução(ões) deve(m) ser executada(s) 100 vezes.
SUB A,1
JP NZ, endereço da primeira instrução da repetição.

Assim, para simular o ciclo FOR/NEXT indicado, o valor 100 deve ser carregado no registrador A. Este valor representa o número de vezes que as instruções que se encontram no interior do ciclo devem ser executadas. Após cada execução do ciclo, o valor 1 é subtraído do conteúdo do registrador A. Se o resultado desta subtração é NZ (não zero), a execução é desviada para a primeira instrução do ciclo. O processo prossegue, até que a subtração de uma unidade do conteúdo do registrador A dê por resultado zero, o que significa, então, que o ciclo foi executado tantas vezes quanto estava especificado.

As intruções BASIC "PEEK" e "POKE" são, nesta linguagem, aquelas que mais se aproximam dos códigos de máquina. Elas permitem, de fato, o acesso direto ao conteúdo da memória.

BASIC: LET A = PEEK (40000)
Significado: Atribuir à variável A, o valor contido no endereço de memória 40000.
Código de máquina: LD A,(40000).
Significado: Carregar o registrador A com o valor que se encontra no endereço 40000.

Se o valor 81 está armazenado no endereço 40000 e o código LD A,(40000) é executado, o valor 81 será carregado no registrador A. Este registrador é o único pelo qual é possível determinar de modo direto um valor que se encontra em um endereço de memória. Por exemplo, a seguinte instrução é incorreta:

LD D,(40000)

Ao contrário, os três registradores duplos podem ser utilizados desta maneira para simular uma instrução PEEK.

Código de máquina: LD HL,(40000)

Esta instrução tem como efeito carregar para o registrador L o valor que se encontra no endereço 40000 e para o registrador H, o valor que se encontra no endereço 40001. Isto decorre naturalmente do modo como os números maiores que 255 são armazenados na memória principal do computador.

Se os valores 5 e 15 são, respectivamente, armazenados nos endereços 40000 e 40001, qual será o valor contido no registrador HL, após o código de máquina LD HL,(40000) ter sido executado?

BASIC: **POKE(40000),A**
Significado: **Carregar o valor da variável A para o endereço 40000.**
Código de máquina: **LD(40000),A**
Significado: **Carregar o valor que se encontra no regsitrador A para o endereço 40000.**

O registrador A é o único registrador simples que permite que o valor seja carregado diretamente para qualquer local de memória. Ao contrário, os três registradores duplos podem ser, assim, utilizados para simular a execução de uma instrução POKE.

Código de máquina: **LD(40000),HL**
Significado: **Carregar para o endereço 40000 o valor contido no registrador L e para o endereço 40001 o valor contido no registrador H.**

Se o registrador HL contém o valor 35621, qual será o conteúdo dos endereços de memória 40000 e 40001 após o código de máquina LD(40000),HL ter sido executado?

BASIC:
```
LET A=5
LET B=40000
POKE(B),A
```
Significado: Carregar o valor da variável A para o endereço da memória especificado por B.

Códigos de máquina:
```
LD A,5
LD HL,40000
LD (HL),A
```
Significado: Carregar o valor contido no registrador A para o endereço de memória correspondente ao valor contido em HL.

BASIC:
```
LET B=40000
LET A=PEEK(B)
```
Significado: Atribuir para a variável A o valor que se encontra no endereço especificado por B.

Códigos de máquina:
```
LD HL,40000
LD A,(HL)
```
Significado: Carregar para o registrador A o valor que se encontra no endereço correspondente ao valor contido em HL.

O registrador A é o único registrador simples que pode ser utilizado com um endereço especificado pelo conteúdo de um dos três registradores duplos. As combinações permitidas são, então, as seguintes:

```
LD A,(HL)
LD A,(BC)
LD A,(DE)
```

Os seis outros registradores simples não podem ser carregados assim, uma vez que o endereço está especificado pelo registrador duplo HL, por meio das seguintes combinações:

```
LD B,(HL)
LD C,(HL)
LD D,(HL)
LD E,(HL)
LD H,(HL)
LD L,(HL)
```

Códigos de máquina:
```
PUSH
POP
```

Estas duas intruções não possuem, na verdade, equivalentes em BASIC. Em compensação, elas são essenciais para a programação em linguagem de máquina

e o exemplo a seguir permitirá ilustrar sua função. Suponhamos um programa que utilize os sete registradores da seguinte maneira:

HL
BC
DE
A

Se se deseja executar várias vezes esta série de instruções em BASIC, utilizar-se-á o ciclo FOR/NEXT. Como indicado anteriormente, um ciclo deste tipo pode ser simulado por instruções em linguagem de máquina. No exemplo apresentado, estas instruções são as seguintes:

```
                    LD A,8 (número de vezes que o ciclo deve ser repeti-
                           do).

(início do subprograma) HL
                        BC
                        DE
                        A
                        SUB A,1
                        JP NZ, (início do subprograma)
                        END
```

Com estas instruções, o programa não rodará. De fato, é necessário que seja definido o ciclo FOR/NEXT, tal que FOR A=1 TO 8 e que a variável A seja utilizada não apenas como contador, mas também como parâmetro no interior do ciclo. Nestas condições, o valor de A que controla o ciclo será modificado pela sua utilização no interior do mesmo. Em BASIC, tal problema é facilmente evitado, porque temos disponível um número muito grande de variáveis diferentes. Em linguagem de máquina o mesmo não ocorre, e as instruções PUSH e POP são destinadas a superar esta dificuldade.

O suprograma abaixo apresentado mostra como estas instruções devem ser utilizadas para operar o subprograma precedente:

```
                        LD A,8
(início do subprograma) PUSH A
                        HL
                        BC
                        DE
                        A
                        POP A
                        SUB A,1
                        JP NZ,(início do subprograma)
```

Depois que o valor 8 foi carregado no registrador A, a instrução PUSH A é executada. Isto significa que o valor corrente deste registrador é colocado de lado, num local chamado pilha. As instruções seguintes podem então ser executadas sem risco de perder este valor em particular. Uma vez que o subprograma tenha sido executado, a instrução POP A permite que o registrador A retome seu valor original. PUSH é denominado instrução de empilhar e POP instrução de desempilhar.

Pode-se, assim, empilhar ou desempilhar qualquer um dos registradores duplos. Em compensação, estas operações não podem ser efetuadas nos registradores simples, de modo contrário ao que se pode ser levado a crer pelo exemplo dado acima. Este foi escrito apenas para evitar qualquer confusão, mas o código de máquina PUSH A não existe. Em compensação, é possível desempilhar um registrador duplo chamado AF. O registrador F é um registrador particular que não pode ser utilizado do mesmo modo que os outros sete registradores simples. É possível, portanto, escrever programas bastante elaborados em linguagem de máquina sem recorrer ao registrador F.

A primeira função do registrador F é de servir ao microprocessador Z80 para indicar o resultado de diferentes cálculos. Em função do valor contido neste registrador o microprocessador pode, por exemplo, indicar se o resultado do último cálculo efetuado é nulo, positivo ou negativo, ou se ele gerou um transporte. É precisamente este transporte que dá a diferença entre os códigos de máquina SUB (subtração) e SBC (subtração com transporte). Se o indicador de transporte está posicionado, o transporte foi feito por conta da subtração efetuada pela instrução SBC. A utilização do registrador F é muito delicada para os programadores pouco familiarizados com a linguagem de máquina. Assim, ela não será mais evocada no decorrer deste livro.

3

ARMAZENAMENTO DOS CÓDIGOS DE MÁQUINA NA MEMÓRIA

Até agora, somente a maneira como os números são armazenados na memória foi descrita. Foi mencionado, entretanto, que os códigos de máquina eram representados por um único número ou por uma combinação de números. Na prática, isto significa que alguns códigos de máquina são representados por apenas um número, compreendido entre 0 e 255, enquanto que outros o são por dois números compreendidos entre os mesmos limites.

Os números que representam os códigos de máquina são armazenados na memória de maneira idêntica àquela utilizada para o armazenamento de qualquer outro número. Certos códigos de máquina necessitam de uma única posição de memória (quer dizer, apenas um byte), enquanto outros necessitam de dois.

Se a primeira instrução do programa fosse INC A (incrementar o valor do registrador A), a primeira coisa a ser conhecida seria o valor do número correspondente a esta instrução. Na verdade, trata-se do número 60. Teríamos, em seguida, necessidade de saber em qual endereço de memória deve começar este programa. Suponhamos que este seja o endereço 40000. O valor 60 deve, então, ser carregado no endereço 40000 (o procedimento a ser adotado para realizar esta operação será explicado mais adiante). Uma vez que o primeiro código de máquina tenha sido carregado para a memória central, o número correspondente à segunda instrução, por exemplo, 52 para o código INC HL, deve ser carregado no endereço seguinte, neste caso, 40001.

Todo o programa é assim construído passo a passo, armazenando nos endereços consecutivos os números correspondentes às sucessivas instruções. Os dois códigos de máquina dados como exemplo são armazenados em um único byte, o que quer dizer que cada um deles não ocupa mais do que uma posição de memória. Entretanto, outras instruções como ADD A,5 necessitam de dois bytes. A primeira posição de memória contém o número correspondente à instrução ADD A, ao passo que a posição seguinte contém o valor que deve ser adicionado ao conteúdo do registrador A. Se ADD A,5 fosse a primeira instrução de um programa que começasse no endereço 40000, o valor 198 (que é o número correspondente ao código de máquina ADD A) deveria ser carregado no endereço 40000, e o valor 5 carregado no endereço 40001.

Quando o programa é iniciado, o microprocessador examina o conteúdo do primeiro endereço de memória, neste caso, ADD A. Em seguida, ele verifica, no endereço 40001, qual é o valor que deve ser adicionado ao valor do registrador A. Uma vez efetuada a soma, o microprocessador busca o código de máquina da instrução que se encontra no endererço seguinte, neste caso 40002.

Certos códigos de máquina ocupam dois bytes e devem ser seguidos de um ou dois argumentos que ocupam cada um, um byte. Estas instruções em linguagem de máquina ocupam, assim, três ou quatro bytes no total. Agora, estes bytes devem ser armazenados em endereços consecutivos na memória principal.

Uma diferença essencial em relação ao BASIC reside no fato de que as instruções em linguagem de máquina não são precedidas por um número de

linha. Elas são armazenadas em endereços consecutivos na memória principal. O microprocessador guarda permanentemente um ponteiro para o endereço da instrução que está prestes a ser executada. O ponteiro desloca-se para o endereço da instrução seguinte uma vez que a primeira tenha sido executada. Apesar da ausência de números de linha, é possível, entretanto, existir o equivalente às instruções BASIC de desvio GOTO e GOSUB. Ao invés de ser dirigida para um número de linha, a execução é desviada para um endereço de memória.

Uma vez que se escreva um programa em linguagem de máquina, os números não são digitados utilizando-se o sistema clássico de notação decimal. Em seu lugar, é utilizada uma notação chamada "hexadecimal" (HEX é a notação abreviada), que quer dizer, uma notação que se refere ao sistema de numeração na base 16. Qualquer dos exemplos abaixo ilustra a correspondência entre os sistemas decimal e hexadecimal:

Decimal	Hexadecimal
0	00
9	09
10	OA
15	OF
16	10
255	FF

Um quadro completo de conversão é dado no apêndice. Neste livro, para evitar qualquer confusão, os números expressos em notação decimal são seguidos da letra d e os em notação hexadecimal são seguidos da letra h:

8d = 8 decimal
12h = 12 hexadecimal

Qual é o equivalente decimal de E3h?
Se FBh é a parte fixa de um número e CBh é sua parte flutuante, que número é este em decimal?

O quadro de conversão decimal/hexadecimal dado no final deste livro pode ser utilizado para responder a estas questões.

A conversão de um número de um sistema para outro torna-se bastante fácil assim que se possui um mínimo de prática. Uma das vantagens do sistema hexadecimal é sua enorme facilidade de utilização na informática.

Dois algarismos hexadecimais são suficientes para representar qualquer número decimal compreendido entre 0 e 255 (ou seja, números compostos por um, dois ou três algarismos decimais).

Existem vários métodos que permitem colocar números ou códigos de máquina na memória central do computador. O programa BASIC apresentado

abaixo pode ser utilizado para colocar e verificar bastante rapidamente um programa escrito em linguagem de máquina. Este programa BASIC deve ser cuidadosamente batido e após se ter verificado a inexistência de erro de cópia, ser armazenado em fita por meio da instrução:

SAVE "CAS:ENTHEX"

```
10 CLEAR 200,39999
15 CLS
20 LOCATE 0,0
25 PRINT "Verifique as maiusculas"
30 LOCATE 0,4
35 PRINT "Tecle E para entrar um codigo hexadecimal"
40 LOCATE 0,8
45 PRINT "Tecle V para verificar um codigo hexadecimal"
50 LOCATE 0,12
55 PRINT "Tecle X para verificar todos os codigos digitados"
60 LOCATE 0,16
65 PRINT "Tecle P para parar"
70 A$=INKEY$
75 IF A$="E" THEN 185
80 IF A$="V" THEN 380
85 IF A$="X" THEN 100
90 IF A$="P" THEN STOP ELSE 70
100 CLS
105 LOCATE 0,0
110 INPUT "Entre o endereco inicial";AS
120 LOCATE 0,5
125 INPUT "Entre o endereco final";AE
135 LET D=0
140 FOR I=AS TO AE
145 LET D=D+PEEK(I)
150 NEXT I
155 CLS
160 PRINT "Totalisador=";D
165 LOCATE 0,20
170 PRINT "Tecle M para retornar ao menu"
175 IF INKEY$ <> "M" THEN 175 ELSE 15
185 CLS
190 LOCATE 0,0
195 PRINT "Verifique as maiusculas"
200 LOCATE 0,4
205 INPUT "Entre o endereco inicial";S
215 IF S<40000 THEN 370
220 LET A$=""
225 LOCATE 0,23
230 LET ET=S
235 IF A$="" THEN INPUT A$
240 LET BAD=0
245 IF A$="M" THEN 15
250 LET E=LEN(A$)-1
260 LET C$=A$
265 FOR D=1 TO E STEP 2
```

```
270 LET B$=LEFT$(C$,2)
275 LET C=VAL("&H"+B$)
280 IF C=0 THEN GOSUB 360
285 LET C$=MID$(C$,3)
290 NEXT D
295 IF BAD=1 THEN 345
300 LOCATE 0,21:PRINT S;" "A$
305 LET B$=LEFT$(A$,2)
310 IF LEN(B$)=1 THEN 345
315 LET C=VAL("&H"+B$)
320 POKE (S),C
325 LET S=S+1
330 LET A$=MID$(A$,3)
335 IF A$="" THEN 230 ELSE 305
345 LOCATE 0,22:PRINT "Entrada incorreta.Tente novamente"
350 LET S=ET
355 GOTO 220
360 IF B$<>"00" THEN LET BAD=1
365 RETURN
370 PRINT "O endereco inicial deve ser maior ou igual a 40000"
375 GOTO 205
380 LET ND=0
385 CLS
390 LOCATE 0,0
395 INPUT "Entre o endereco inicial";AS
405 LOCATE 0,5
410 INPUT "Entre o endereco final";AE
420 CLS
425 IF AS+20>AE THEN 490
430 FOR C=AS TO AS+20
435 IF PEEK(C)<16 THEN 480
440 PRINT C;"";HEX$(PEEK(C))
445 NEXT C
450 IF ND=1 OR C>AE THEN 505
460 PRINT "Tecle M se voce deseja continuar"
465 LET AS=C
470 IF INKEY$ <> "M" THEN 470 ELSE 425
480 PRINT C;"0";HEX$(PEEK(C))
485 GOTO 445
490 FOR C AS TO AE
495 LET ND=1
500 GOTO 435
505 PRINT "Tecle M para voltar ao menu"
510 IF INKEY$ <> "M" THEN 510 ELSE 15
```

Para carregar este programa a partir da fita, é suficiente digitar o comando:

LOAD "CAS:",r

Este comando permite começar automaticamente a execução do programa, uma vez que estiver carregado na memória central. O seguinte menu é, então, mostrado:

Tecle E para entrar um código hexadecimal.
Tecle V para verificar um código hexadecimal.
Tecle X para verificar todos os códigos digitados.
Tecle P para parar.

Para entrar com uma instrução em código de maquina é suficiente, então, teclar E. Uma mensagem indicando para verificar as maiúsculas é, então, mostrada.

O pequeno programa em linguagem de máquina dado abaixo permitirá ao leitor testar se o programa BASIC "ENTHEX" foi copiado corretamente. Este programa de demonstração adiciona dois números e armazena o resultado no endereço 40100. O processo a ser seguido é o seguinte (em resposta às mensagens mostradas no vídeo):

1. Teclar E.
2. Apertar a tecla para verificar maiúsculas.
3. O endereço inicial é 40000.
4. Os códigos hexadecimais podem agora ser digitados. O primeiro é 3E05 (pressione em seguida a tecla "carriage return"). Isto corresponde aos dois bytes que se encontram nos endereços 40000 e 40001.
5. Entrar da mesma forma as três outras linhas, teclando "carriage return" no final de cada uma delas.
6. Após a última linha ter sido digitada (e o "carriage return" teclado), é suficiente teclar M (sempre seguido de um "carriage return") para voltar ao menu.

Endereço inicial 40000
Endereço final 40007
Total hexadecimal 852

```
3E05      LD A,5
C610      ADD A,16
32A49C    LD (40100),A
C9        RET
```

Deve-se, em seguida, testar se os códigos entraram corretamente. Para isto, é suficiente teclar V (estando no menu). Pede-se, então, a entrada do endereço inicial do código de máquina que deve ser testado, isto é, 40000. Após o endereço final ser requisitado, digitar 40007. As posições de memória e seus conteúdos são, então, mostrados no vídeo. Estes devem ser cuidadosamente verificados. Se um erro for detectado, basta voltar ao menu e entrar novamente com o código de máquina correto.

A ativação da tecla X permite, assim que o menu seja mostrado, efetuar uma verificação suplementar. De novo, os endereços inicial e final devem ser fornecidos. O programa ENTHEX adiciona, deste modo, o conteúdo dos valores que se encontram em todos estes endereços e fixa o total, que neste exemplo, é 852. Se isto não acontecer, é que um código está incorreto ou, numa hipótese menos provável, o programa ENTHEX foi mal batido. O problema deve ser resolvido antes de prosseguir.

Todas estas verificações são essenciais porque, contrariamente ao que

ocorre no BASIC, onde a execução do programa se interrompe assim que um erro é encontrado, um erro em programa escrito em linguagem de máquina leva o computador a realizar ciclos sem fim, dos quais o único meio de sair é interromper o funcionamento da máquina.

A fim de testar o programa em linguagem de máquina que acaba de ser escrito, é suficiente sair do programa ENTHEX respondendo P à escolha proposta pelo menu. Estas duas linhas devem ser escritas em seguida:

```
1000 def usr = 40000
1005 a = usr (1)
```

Estas duas linhas têm um significado equivalente àquele da instrução GOSUB, sendo a única diferença que, neste caso, o subprograma chamado é escrito em linguagem de máquina. Pode-se, além disso, observar que a última instrução deste subprograma é uma instrução RET, equivalente do RETURN do BASIC. Assim que o microprocessador encontra esta instrução, ele desvia a execução para a instrução de chamada, quer dizer, neste caso, ao programa BASIC.

Para executar o subprograma em linguagem de máquina, basta digitar de modo direto o comando GOTO 1000, seguido de PRINT PEEK (40100). O número 21 deve, então, chamar a atenção; ele corresponde ao resultado da soma de 16 e 5.

O procedimento a ser utilizado para entrar e verificar todos os programas em linguagem de máquina que se encontram neste livro é idêntico àquele que acaba de ser detalhado. Ele não será, portanto, mais explicado. É importante sempre notar os endereços inicial e final destes programas, e, também, a soma dos códigos hexadecimais colocados em jogo. Estas condições são fornecidas no alto de cada listagem. Da mesma forma, os testes a serem efetuados são sempre idênticos aos que acabaram de ser descritos. Por outro lado, quaisquer linhas de programas BASIC podem ser igualmente indicadas no alto de cada listagem: elas começam sempre na linha 1000 e serão executáveis por meio do comando GOTO 1000. Assim que os testes forem efetuados, o programa ENTHEX poderá ser reiniciado por meio do comando RUN.

4

A ESCOLHA DO VÍDEO

Os computadores MSX dispõem de quatro modos de vídeo diferentes:

MODO 0	**24 linhas, 40 caracteres por linha**
MODO 1	**24 linhas, 32 caracteres por linha**
MODO 2	**modo gráfico de alta resolução**
MODO 3	**modo multicolorido**

Cada um destes modos possui características específicas, e a escolha de cada um deles depende da natureza do programa a ser executado.

No restante deste livro, suporemos, salvo especificações em contrário, que o modo 1 foi o escolhido. Entretanto, um capítulo em particular foi consagrado à descrição dos outros modos. O modo 1 é, provavelmente, aquele que melhor se adaptou à realização de programas de jogos.

A questão essencial que se vai tentar responder neste capítulo é: "Como a imagem mostrada no vídeo é construída?". Os caracteres que compõem o texto que se deseja mostrar são armazenados na memória sob a forma de números. O computador examina constantemente a parte da memória que contém estes números e efetua as operações necessárias para transformá-los em uma imagem de vídeo.

Mencionou-se anteriormente que os programas escritos em BASIC ou em linguagem de máquina eram armazenados em área específica da memória principal do computador, chamada memória RAM. Existe, nos computadores MSX, uma outra área de memória destinada ao armazenamento das informações que se referem a imagens do vídeo. Esta área é chamada de memória RAM do vídeo, ou VRAM. Ela ocupa 16 384 bytes.

A memória VRAM é utilizada para aramazenar os dados referentes à imagem a ser mostrada, bem como outras informações, tais como o tamanho dos caracteres e dos *sprites* (o próximo capítulo é destinado à descrição dos *sprites*). No modo vídeo 1, um quadro de 32 x 24 = 768 bytes está definido na memória VRAM. Cada elemento deste quadro corresponde a uma posição de caractere para o vídeo (32 colunas x 24 linhas = 768 posições). O número 65 corresponde à letra A; se o primeiro elemento (quer dizer, o primeiro byte) da tabela contém o número 65, a letra A será mostrada no canto superior esquerdo do vídeo. Para compreender mais detalhadamente o processo, convém conhecer o endereço na memória VRAM, do primeiro dos 768 bytes consecutivos da tabela. Para isto, basta digitar as duas linhas seguintes:

```
SCREEN 1
PRINT BASE (5)
```

A primeira instrução seleciona o modo vídeo 1. A segunda fixa o endereço da base (isto é, o endereço do primeiro byte) desta tabela. Este endereço em particular será chamado INICIOVIDEO. Os computadores MSX iniciam este endereço em 6144. Para carregar o valor 65 no endereço INICIOVIDEO, basta digitar a instrução:

VPOKE(INICIOVIDEO),65

A palavra INICIOVIDEO não deve ser digitada, e sim substituída por seu valor, por exemplo 6144, se o endereço da base desta tabela não foi modificado pelo programador. A letra A aparecerá, então, no canto superior esquerdo do vídeo. Agora, é muito simples calcular o endereço de qualquer outra posição, sabendo-se que um byte da tabela corresponde a uma posição de caractere. Por exemplo, o endereço que define a posição no canto superior direito do vídeo é INICIOVIDEO+31. Deve-se notar que em alguns televisores a imagem não está sempre perfeitamente centralizada, de modo que os ângulos da imagem não são sempre visíveis.

Qualquer número compreendido entre 0 e 255 pode ser assim carregado por meio da instrução VPOKE, para qualquer dos 768 bytes que formam a tabela. O caractere correspondente ao número será mostrado em uma posição definida pelo endereço do byte na tabela. A memória VRAM contém igualmente uma outra tabela, na qual são armazenados os dados que correspondem às formas dos caracteres. Cada forma é definida por 8 bytes. Como estão disponíveis 256 caracteres diferentes, a tabela de formas se estende por 8 x 256 = 2 048 bytes. O endereço da base desta tabela na memória VRAM pode ser conhecido digitando:

PRINT BASE (7)

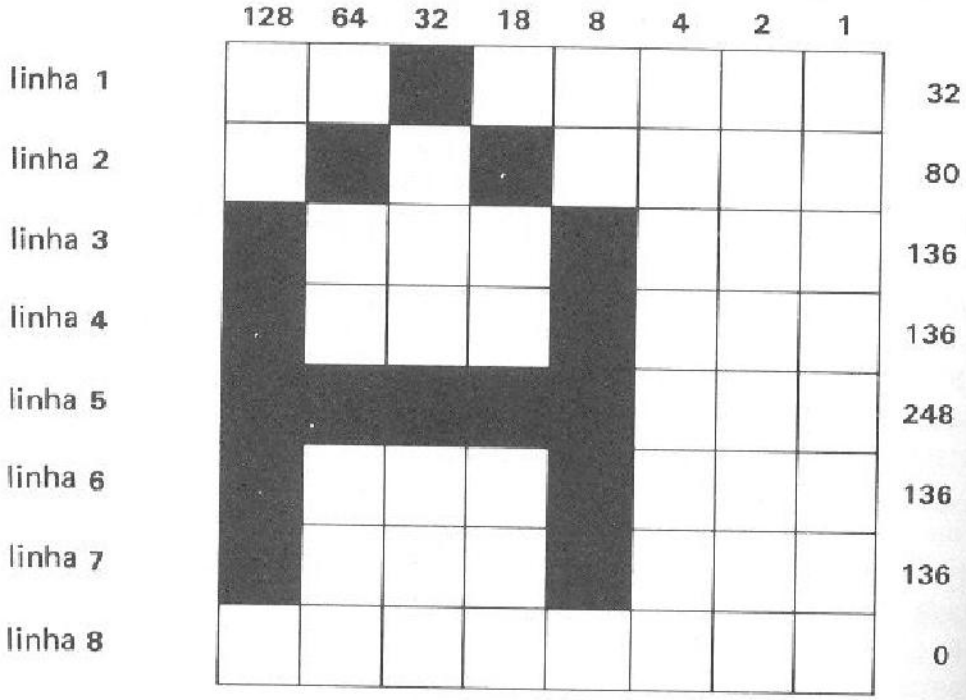
No modo vídeo 1, os computadores MSX iniciam esta tabela de formas em zero, que deve ser, então, o valor mostrado logo que a instrução acima é executada (salvo se o endereço da base desta tabela foi modificado pelo programador). Este endereço de iniciação será designado em seguida por INICIOCAR. O endereço do primeiro byte que regula a forma de qualquer um dos caracteres é então dado pela fórmula:

(Número do caractere x 8) + INICIOCAR

Como é definida e armazenada uma forma de caractere na memória VRAM? Consideremos, por exemplo, a letra A, que tem por número 65 e cujo primeiro byte de forma se encontra, portanto, na memória VRAM no seguinte endereço:

(65 x 8) + INICIOCAR = 520

Cada uma das formas do caractere pode ser visualizada em uma rede de 8 x 8, como mostra o exemplo a seguir, no caso da letra A.



Cada uma das oito linhas horizontais representa um byte. Cada linha é convertida em um número cujo valor depende da representação dos quadros pretos e brancos que a compõem (ver no apêndice o parágrafo consagrado à notação binária). São estes oito números (32, 80, 136, . . ., 0) que são armazenados na memória VRAM e que definem a forma da letra A. O exemplo seguinte mostra como estes dados de forma são armazenados:

```
VPOKE(INICIOVIDEO+5),65
```

Esta instrução permite mostrar a letra A no vídeo. Suponhamos que nós desejamos modificar a forma desta letra. Para isto, as linhas seguintes podem ser digitadas de modo direto. Seu efeito sobre a forma do caractere a ser mostrado pode ser visualizado após digitar a tecla "carriage return" (INICIOCAR deve ser substituído pelo endereço de início da base da tabela de formas, que é normalmente zero):

```
VPOKE(INICIOCAR+520),255
VPOKE(INICIOCAR+521),129
VPOKE(INICIOCAR+522),129
VPOKE(INICIOCAR+523),129
VPOKE(INICIOCAR+524),129
VPOKE(INICIOCAR+525),129
VPOKE(INICIOCAR+526),129
VPOKE(INICIOCAR+527),255
```

A forma correta do caractere número 65 (letra A) foi, assim, completamente redesenhada. O procedimento utilizado é exatamente o mesmo que

deve ser realizado para criar um caractere que não está normalmente definido no conjunto de caracteres MSX. No apêndice, é fornecido um programa que permite redesenhar rapidamente um jogo completo de caracteres e armazená-los em fita para utilização em outros programas.

Uma terceira tabela, a tabela de cores, é iniciada na memória VRAM. Ela se estende por 32 bytes, cada byte respondendo pela cor de um bloco de 8 caracteres. É impossível definir diferentes cores dentro de um mesmo bloco, de modo que, assim que um valor (correspondente a um número de cor) tenha sido destinado a um byte, os oito caracteres do bloco correspondente são fixados nesta cor. O endereço da base da tabela das cores pode ser conhecido por meio da instrução:

```
PRINT BASE(6)
```

No modo vídeo 1, o endereço da base desta tabela é iniciado em 8192 na memória VRAM. Este endereço será designado por INICIOCOR. Para ilustrar o modo pelo qual a tabela de cores controla a fixação dos caracteres, as duas linhas seguintes podem ser digitadas:

```
SCREEN 1
VPOKE(INICIOVIDEO+5),65
```

A letra A é assim mostrada no vídeo. Para modificar o byte que responde pela cor na qual está este caractere, deve-se começar por calcular o endereço deste byte de cor, ou seja:

```
INICIOCOR + (número do caractere / 8)
```

Só a parte inteira do resultado desta operação é retida. Se o endereço de início da base da tabela de cores não foi modificado pelo programador, o resultado é:

```
8192 + (65 / 8) = 8200
```

Nestas condições, a cor de fixação da letra A pode ser modificada pelas instruções abaixo:

```
VPOKE(8200),241
```

A letra A é, então, mostrada em branco sobre fundo negro.

```
VPOKE(8200),159
```

A letra A é, então, mostrada em vermelho sobre fundo branco.

Dezesseis cores diferentes estão disponíveis, cada cor sendo identificada por um número:

0 transparente
1 preto
2 verde
3 verde claro
4 azul escuro
5 azul claro
6 vermelho escuro
7 ciano
8 vermelho
9 vermelho claro
10 amarelo escuro
11 amarelo claro
12 verde escuro
13 magenta
14 cinza
15 branco

O valor do argumento a ser fornecido à instrução VPOKE para definir as cores escolhidas pode ser calculado da seguinte maneira:

- Multiplicar por 16 o número da cor previamente escolhida e somar a este resultado o número da cor de fundo.
- Para afixar, por exemplo, um caractere magenta em fundo branco, o argumento deve ter por valor:

(13 X 16) + 15 = 223

5

OS SPRITES

Os sprites constituem uma das particularidades dos computadores MSX. Suponhamos que um programa de jogo tenha sido escrito e que ele necessite do deslocamento de uma figura sobre o vídeo. Na falta do sprite, seria necessário fixar uma forma definida em uma posição específica do vídeo, e apagá-la, depois de redefinir a forma ao nível de uma outra posição no mesmo, etc. Este procedimento não é apenas trabalhoso, mas tem igualmente um desempenho ruim em relação à rapidez do deslocamento que seria possível programar desta forma.

A possibilidade de definir os sprites facilita consideravelmente a concepção de programas que necessitam de deslocamento de figuras gráficas predefinidas. Imaginemos que um sprite seja simplesmente um caractere qualquer; é possível definir sua forma e sua cor. Um sprite pode ser afixado em qualquer posição do vídeo e não necessariamente em um local predeterminado por um caractere. Os sprites não têm necessidade de ser apagados de sua posição para simular a impressão de movimento. O deslocamento de um sprite, tanto em BASIC quanto em linguagem de máquina, não necessita mais do que a modificação do conteúdo de um local de memória.

No máximo, 32 sprites podem ser afixados simultaneamente no vídeo. Entretanto, existe uma limitação suplementar que será abordada mais adiante. Os sprites podem ser afixados em quatro tamanhos diferentes. Contudo, todos os sprites que se encontram no vídeo em um dado instante devem necessariamente ter o mesmo tamanho. No exemplo abaixo, o tamanho mínimo foi escolhido, isto é, um tamanho correspondente àquele em que são afixados os caracteres normais. Os 32 sprites são numerados de 0 a 31. Para posicionar o tamanho do sprite selecionado basta digitar:

```
VDP (1) = 224
```

Os outros tamanhos de sprites disponíveis serão descritos no próximo capítulo. Existe, na memória VRAM, uma tabela na qual são armazenadas as informações concernentes às formas que foram definidas para os sprites. O endereço desta tabela pode ser conhecido, digitando:

```
PRINT BASE (9)
```

Este endereço é iniciado em 14336, no modo vídeo 1 (como aliás nos outros modos de vídeo que os sprites podem ser utilizados). Este endereço será designado por SPRITEFORMA. Como os caracteres habituais, os sprites deste tamanho (mínimo) são definidos para 8 bytes, o que quer dizer que os dados de forma que se referem ao primeiro sprite são armazenados entre os endereços 14336 e 14343. Qualquer uma das instruções seguintes mostra como os valores podem ser carregados para estes oitos bytes, a fim de que o sprite deste modo definido represente o desenho de um "invasor do espaço".

```
VPOKE SPRITEFORMA + 0,60
VPOKE SPRITEFORMA + 1,90
VPOKE SPRITEFORMA + 2,255
VPOKE SPRITEFORMA + 3,231
VPOKE SPRITEFORMA + 4,128
VPOKE SPRITEFORMA + 5,36
VPOKE SPRITEFORMA + 6,66
VPOKE SPRITEFORMA + 7,36
```

Estas instruções não são suficientes para fixar no vídeo a forma assim definida. É necessário também precisar em qual lugar do vídeo o sprite deve ser posicionado. Uma tabela suplementar na memória VRAM contém as informações sobre a posição e a cor dos sprites. Esta tabela (tabela dos atributos dos sprites), contém, para cada um dos 32 sprites, quatro informações:

1. Sua posição vertical
2. Sua posição horizontal
3. Seu número
4. Sua cor

O endereço da base da tabela dos atributos dos sprites pode ser conhecido digitando:

```
PRINT BASE (8)
```

Os computadores MSX iniciam a base desta tabela no endereço 6912. Este endereço será designado aqui por SPRITEATRIB. Em conseqüência, o endereço do primeiro byte que regulamenta os quatro atributos correspondentes a um dado sprite pode ser calculado por meio da fórmula:

(número do sprite x 4) + SPRITEATRIB

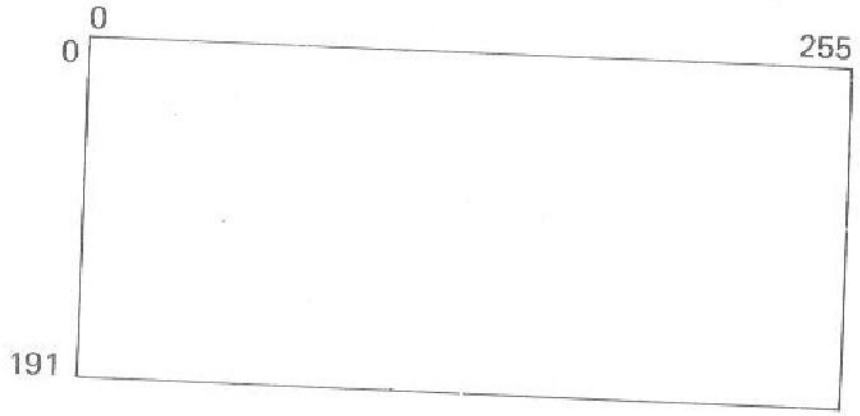
Por exemplo, na hipótese de que o endereço da base da tabela de atributos de sprites não tenha sido modificado pelo programador, o primeiro byte que regula os atributos do sprite número 15 se encontra no endereço:

(15 x 4) + 6912 = 6972

As quatro linhas seguintes permitem que apareça no vídeo o sprite previamente definido:

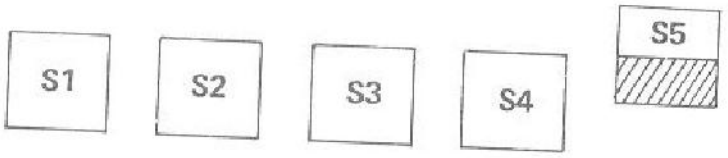
```
VPOKE (SPRITEATRIB + 0), 100
VPOKE (SPRITEATRIB + 1), 50
VPOKE (SPRITEATRIB + 2), 0
VPOKE (SPRITEATRIB + 3), 10
```

Os parâmetros se referem às posições horizontal e vertical que correspondem ao canto superior esquerdo de cada sprite. Para posicionar os sprites no vídeo, os seguintes pontos de referência podem ser definidos:



Deste modo, é possível, dando-se diferentes valores aos parâmetros de posição das instruções VPOKE, deslocar um sprite em diferentes partes do vídeo. Se o parâmetro correspondente à posição vertical é maior que 191, o sprite some progressivamente atrás da tela. Por outro lado, existe um registrador de colisão que é posicionado logo que dois sprites se superpõem, ao menos parcialmente no vídeo. Sua utilização será ilustrada no programa apresentado no próximo capítulo.

Existe uma importante limitação quanto às possibilidades de fixação dos sprites na tela. Se bem que o número total de sprites fixados simultaneamente seja 32, apenas 4 deles podem aparecer ao mesmo tempo em uma mesma linha horizontal. Deve-se ter sempre presente esta limitação quanto ao sprite quando se escreve um programa, particularmente se for um programa de jogo. O diagrama abaixo mostra a maneira pela qual são afixados os sprites suplementares quando esta limitação é ultrapassada:



(A parte hachurada do sprite 5 não aparece no vídeo.)

A descrição da utilização dos sprites mereceria um livro inteiramente dedicado ao assunto (ver por exemplo o excelente livro de Mike Shaw, *MSX guide du graphisme*, Sybex, 1985). No apêndice 4, damos um programa que permite desenhar as formas dos sprites e armazená-las em fita para utilização em outros programas.

6

LISTAGEM DE UM PROGRAMA EM LINGUAGEM DE MÁQUINA: OS INVASORES DO ESPAÇO

Um conjunto de subprogramas escritos em linguagem de máquina e cuja função está ligada à entrada/saída do sistema BASIC (subprograma BIOS) encontram-se na memória ROM dos computadores MSX. Estes subprogramas podem ser utilizados para facilitar a concepção de um programa escrito em linguagem de máquina. Os mais importantes entre eles serão descritos em detalhes no próximo capítulo. Entretanto, como o programa apresentado aqui utiliza alguns destes subprogramas, as indicações indispensáveis serão dadas já neste nível.

O programa de jogo apresentado aqui é simples. Entretanto, todos os princípios mostrados são aplicáveis à realização de programas mais sofisticados. Um programa pode ser decomposto em uma sucessão de blocos, sendo em seguida cada bloco decomposto em uma série de subprogramas. Segundo este princípio, o leitor deve ser capaz, ao final deste capítulo, de escrever seus próprios programas em linguagem de máquina.

Este capítulo se inicia com uma introdução à noção de fluxograma, já familiar aos leitores que tenham prática em linguagem BASIC. Prossegue com a descrição da organização de um mapa de memória. Esta etapa, que não é necessária em BASIC, é indispensável quando se começa a escrever programas em linguagem de máquina. Cada um dos blocos do fluxograma será detalhado em seguida. Um destes blocos diz respeito, por exemplo, ao deslocamento de uma bola no vídeo. O método utilizado para realizar este deslocamento será explicado e o fluxograma de um subprograma interno será igualmente apresentado nesta ocasião.

Todos os subprogramas foram escritos com a finalidade de poderem ser testados individualmente, ou em combinação com outros subprogramas que já o foram. O programa completo pode assim ser examinado etapa por etapa e pode-se ver qual é exatamente a função de cada um dos seus subprogramas que o constituem.

FLUXOGRAMA

O fluxograma é uma descrição gráfica das diferentes operações a serem efetuadas pelo programa, o esquema devendo mostrar em princípio a articulação lógica entre os diferentes blocos. O fluxograma apresentado abaixo é despojado de todo e qualquer simbolismo para não deixar aparecer senão a divisão em blocos do programa "Os invasores do espaço".

Início do jogo
CHAMADA DO SUBPROGRAMA DE INICIAÇÃO (INIC)

CICLO DO JOGO

CHAMADA DO SUBPROGRAMA DE DESLOCAMENTO DOS INVASORES (DESINV)
CHAMADA DO SUBPROGRAMA DE ATRASO (TEMPO). O INVASOR RECEBEU UM TIRO?
SE SIM, RAMIFICAÇÃO PARA O SUBPROGRAMA GERADOR DE UMA EXPLOSÃO (EXPLO)

UMA BALA FOI ATIRADA?
SE SIM, CHAMADA DO SUBPROGRAMA TRAÇO (TIRBAL)

A TECLA DE DESLOCAMENTO DO CURSOR PARA A ESQUERDA FOI PRESSIONADA? SE SIM, CHAMADA DO SUBPROGRAMA DE DESLOCAMENTO DO FUZIL PARA A ESQUERDA (FUZESQ)

CHAMADA DO SUBPROGRAMA TEMPO

A TECLA DE DESLOCAMENTO DO CURSOR PARA A DIREITA FOI PRESSIONADA? SE SIM, CHAMADA DO SUBPROGRAMA DE DESLOCAMENTO DO FUZIL PARA A DIREITA (FUZDIR)

A TECLA DE ESPAÇO FOI PRESSIONADA? SE SIM, CHAMADA DO SUBPROGRAMA DE TIRO (BALA)

CHAMADA DO SUBPROGRAMA TEMPO

A TECLA X FOI PRESSIONADA? SE SIM, VOLTE AO BASIC.

RAMIFICAÇÃO PARA O CICLO DO JOGO.

70 bytes

SUBPROGRAMA TEMPO	6 bytes
SUBPROGRAMA FUZESQ	11 bytes
SUBPROGRAMA FUZDIR	11 bytes
SUBPROGRAMA BALA	37 bytes
SUBPROGRAMA DESINV	62 bytes
SUBPROGRAMA TIRBAL	20 bytes
SUBPROGRAMA DESBAL	27 bytes
SUBPROGRAMA EXPLO	33 bytes
SUBPROGRAMA INIC	125 bytes

O MAPA DA MEMÓRIA

Já foi observado que não existe número de linhas em um programa escrito em linguagem de máquina. Isto significa que ao se desejar acrescentar uma instrução a um programa já escrito, é necesário modificar-se todas as instruções que se encontram depois daquela que se quer inserir, o que necessita que se modifiquem os endereços absolutos dados como argumentos às instruções, tais como CALL e JUMP. Suponhamos, por exemplo, que um subprograma de atraso TEMPO tenha sido escrito a partir do endereço 40000. Para ter acesso a este subprograma, o programa principal contém uma instrução CALL 40000. Se por uma razão qualquer, uma simples instrução de três bytes deva ser acrescentada ao final do subprograma precedente que termina normalmente no endereço 39999, os endereços 40000, 40001 e 40002 deverão ser utilizados. É necessário, então, deslocar o subprograma TEMPO a fim de que se inicie no endereço 40003. Isto feito, as instruções CALL 40000 que se encontram no programa deverão ser mudadas para CALL 40003. A utilização de tal método seria extremamente trabalhosa e exigiria um tempo considerável.

Estas dificuldades podem ser evitadas criando-se um mapa das posições de memória a partir do fluxograma do programa. Para isto, é necessário conhecer o número aproximado de bytes ocupados por cada um dos subprogramas. As posições de memória podem então ser creditadas aos subprogramas, deixando-se alguns bytes livres entre o final de um subprograma e o início do seguinte. Estes bytes suplementares permitirão ampliar, se necessário, um subprograma sem que se faça necessário, para tanto, modificar todo o programa. O fluxograma apresentado abaixo mostra como criar um mapa de memória (no caso de um programa muito simples):

Neste fluxograma, o tamanho de cada subprograma é escrito observando este fato. O subprograma de apagar a tela pode ser deste modo carregado

partir do endereço 40000, o subprograma de atraso a partir do endereço 40050 e o subprograma de afixação a partir do endereço 40080. Alguns bytes são deixados livres entre os dois subprogramas consecutivos, facilitando, assim, sua eventual extensão.

Neste espírito, o tamanho aproximado de cada um dos subprogramas consecutivos do programa "Os invasores do espaço" é indicado no fluxograma correspondente. A partir destas informações, é possível construir o mapa de memória apresentado abaixo. A coluna da esquerda contém o nome de cada subprograma e a da direita o endereço de início correspondente:

CICLO	**40000**
TIRBAL	**40082**
EXPLO	**40133**
BALA	**40157**
DESBAL	**40205**
DESINV	**40243**
FUZESQ	**40316**
FUZDIR	**40338**
INIC	**40363**
TEMPO	**40474**

Uma região da memória foi igualmente associada aos parâmetros. A função destes será brevemente explicada.

DEFINIÇÕES

A explicação de alguns termos e expressões dados abaixo facilitará a descrição do programa.

1. **Ciclo do jogo** (Ciclo): Trata-se do ciclo principal do programa. Este ciclo controla, em particular, se as teclas correspondentes aos deslocamentos para a direita ou para a esquerda ou ainda para o disparo do tiro foram pressionadas. Em conseqüência, este ciclo efetua as chamadas repetidas ao subprograma, gerando o deslocamento do invasor do espaço. Uma vez que todas as funções foram executadas, o programa é redirigido para a primeira instrução e o ciclo recomeça (daí o nome ciclo do jogo).

2. **Indcol**: Existe um determinado lugar da memória central do computador chamado registrador de colisões. Este registrador ocupa um único byte (endereço 41000) e é normalmente carregado com o valor zero. Ele tem o valor 1 quando uma bala atinge o invasor (quer dizer, quando há uma colisão).

3. **Indicador de balas atiradas** (Indbal): Trata-se de uma única posição de memória (um byte) que está normalmente posicionado (isto é, que tem

por valor, 1). Assim que uma bala é atirada, o byte é modificado para zero. Este indicador, que se encontra no endereço 41001, é utilizado para evitar que várias balas apareçam simultaneamente no vídeo.

4. **Indicador de direção do invasor** (Inddir): Este indicador, que se estende por um único byte (endereço 41002), é posicionado se o invasor se desloca da direita para a esquerda. Ele é zero se o invasor se desloca da esquerda para a direita.

SUBPROGRAMA DE INICIALIZAÇÃO

Este programa realiza as seguintes operações:

- posiciona o tamanho dos sprites de modo que estes sejam aumentados (de um fator de 2);
- posiciona o endereço da base da tabela de formatos dos sprites (endereço SPRITEFORMA) em 14336;
- posiciona o endereço da base da tabela dos atributos dos sprites (endereço SPRITEATRIB) em 6912;
- determina um valor para os quatro parâmetros que governam a afixação dos 3 sprites que foram definidos (quer dizer, o invasor, a bala e a base de tiro);
- desenha as formas destes três sprites.

Os 24 bytes necessários à definição das formas dos sprites são carregados para os endereços 14336 até 14359 pelo subprograma de inicialização.

Endereço inicial 40363
Endereço final 40463
Total hexadecimal 6910

```
3E01      2050 INIC:    LD    A,1
32AFFC    2060          LD    (64687),A
CD5F00    2070          CALL  95
21EC9D    2080          LD    HL,dado
110038    2090          LD    DE,14336
011800    2100          LD    BC,24
CD5C00    2110          CALL  92
0E05      2120          LD    C,5
0636      2130          LD    B,54
CD4700    2140          CALL  71
21049E    2150          LD    HL,dado1
11001B    2160          LD    DE,6912
```

```
010C00    2170          LD    BC,12
CD5C00    2180          CALL  92
3E01      2190          LD    A,1
32269E    2200          LD    (indcol),A
32279E    2210          LD    (indbal),A
32289E    2220          LD    (inddir),A
0E01      2230          LD    C,1
06E1      2240          LD    B,225
CD4700    2250          CALL  71
0E02      2260          LD    C,2
0606      2270          LD    B,6
CD4700    2280          CALL  71
C9        2290          RET
18181187E 2300 Dado     DEFB  24,24,126
FFFFFF00  2310          DEFB  255,255,255,0
3C7E99FF  2320          DEFB  60,126,153,255
663C4224  2330          DEFB  102,60,66,36
18181818  2340          DEFB  24,24,24,24
18181818  2350          DEFB  24,24,24,24
AA64000F  2360 Dado1    DEFB  170,100,0,15
0000010C  2370          DEFB  0,0,1,12
C8000201  2380          DEFB  200,0,2,1
```

Este subprograma de inicialização pode ser testado digitando as linhas escritas em BASIC dadas abaixo:

```
1000 DEF USR=40363
1005 A=USR (1)
1010 GOTO 110
```

O desenho do invasor se fixará em verde no canto superior esquerdo do vídeo e o do fuzil será centralizado na base do vídeo. Este último aparecerá em branco.

O SUBPROGRAMA DE DESLOCAMENTO DO FUZIL PARA A ESQUERDA ("FUZESQ")

Este subprograma é chamado quando a tecla que prevê este efeito é pressionada. Começa por testar se o fuzil não se encontra já ao nível da borda esquerda do vídeo, caso em que a execução é imediatamente desviada para o ciclo do jogo. Em caso contrário, o fuzil é reposicionado para a esquerda. Para isto, a posição horizontal corrente do fuzil é sempre lida. Uma unidade é, em seguida, subtraída deste valor e o resultado da operação é carregado para o endereço correspondente na memória VRAM, ou seja, aquela que define a posição horizontal do fuzil.

Endereço inicial **40316**
Endereço final **40327**
Total hexadecimal **1084**

```
21011B    1720 FUZESQ   LD    HL,6913
CD4A00    1730          CALL  74
3D        1740          DEC   A
C8        1750          RET   Z
CD4D00    1760          CALL  77
C9        1770          RET
```

Algumas linhas do programa BASIC abaixo permitem testar este subprograma (o subprograma INIC deve se encontrar já na memória central do computador):

```
1000 DEF USR=40363
1005 A=USR (1)
1010 DEF USR=40316
1015 A=USR (1)
1020 FOR B=1 TO 100
1025 NEXT B
1030 GOTO 1015
```

O fuzil se desloca assim para atingir a borda esquerda do vídeo.

O SUPROGRAMA DE DESLOCAMENTO DO FUZIL PARA A DIREITA ("FUZDIR")

Este subprograma é muito semelhante ao anterior. O teste inicial diz respeito desta vez à presença eventual do fuzil na extremidade direita do vídeo. Por outro lado, uma unidade é adicionada (e não subtraída) à posição horizontal corrente:

Endereço inicial **40338**
Endereço final **40352**
Total hexadecimal **1916**

```
21011B    1880 FUZDIR   LD    HL,6913
CD4A00    1890          CALL  74
D6F0      1900          SUB   240
C8        1910          RET   Z
C6F1      1920          ADD   A,241
CD4D00    1930          CALL  77
C9        1940          RET
```

Este subprograma pode ser testado por meio do programa BASIC precedente, trocando-se a linha 1010 por:

1010 DEF USR=40338

O SUPROGRAMA DE DESLOCAMENTO DO INVASOR ("DESINV")

Este subprograma é o mais elaborado de todos. A relação das operações que ele realiza sucessivamente é dada abaixo:

1. Se o indicador da direção do invasor está posicionado, a execução é desviada para o subprograma de deslocamento de sprite da direita para a esquerda (6).
2. Estando o indicador de direção normalmente em zero, o invasor se desloca da esquerda para a direita.
3. Se o invasor se encontra na extremidade direita do vídeo, o programa posiciona o indicador de direção e retorna ao ciclo do jogo.
4. Não se encontrando o invasor na extremidade direita do vídeo, ele é deslocado de uma unidade para a direita; o princípio deste deslocamento é totalmente semelhante àquele do fuzil: a posição horizontal corrente é sempre lida e, no caso de um deslocamento para a direita, uma unidade é acrescida a este valor. O novo valor é então carregado no local pedido, na memória VRAM, quer dizer, ao endereço de armazenamento da posição horizontal do invasor.
5. Retorno ao ciclo do jogo.
6. O indicador de direção do invasor é posicionado de modo que se desloque da direita para a esquerda.
7. Se o invasor se encontra na extremidade esquerda do vídeo, o programa coloca em zero o indicador de direção e a execução é desviada para o ciclo do jogo.
8. Não se encontrando o invasor na extremidade esquerda do vídeo, ele é deslocado de uma unidade para a esquerda. Para isto, uma unidade é subtraída da posição corrente deste sprite.
9. Retorno ao ciclo do jogo.

Endereço inicial **40243**
Endereço final **40305**
Total hexadecimal **5797**

O subprograma de deslocamento do invasor pode ser testado por meio das seguintes linhas:

```
1000 DEF USR=40363
1005 A=USR (0)
1010 DEF USR=40243
1015 A=USR (0)
1020 GOTO 1015
```

```
3A289E    1350 DESINV   LD    A,(inddir)
3D        1360          DEC   A
CA569D    1370          JP    Z,AESQ
21051B    1380          LD    HL,6917
CD4A00    1390          CALL  74
D6F0      1400          SUB   240
C24B9D    1410          JP    NZ, ADIR1
3E01      1420          LD    A,1
32289E    1430          LD    (inddir),A
C9        1440          RET
21051B    1450 ADIR1    LD    HL,6917
CD4A00    1460          CALL  74
3C        1470          INC   A
CD4D00    1480          CALL  77
C9        1490          RET
21051B    1500 AESQ:    LD    HL,6917
CD4A00    1510          CALL  74
D601      1520          SUB   1
C2679D    1530          JP    NZ, AESQ1
3E00      1540          LD    A,0
32289E    1550          LD    (inddir),A
C9        1560          RET
21051B    1570 AESQ1    LD    HL,6917
CD4A00    1580          CALL  74
3D        1590          DEC   A
CD4D00    1600          CALL  77
C9        1610          RET
```

O SUBPROGRAMA "BALA"

Este subprograma é chamado assim que se pressiona a barra de espaço. Sua primeira função é verificar se já não existe uma bala em movimento no vídeo. Neste caso, a execução é imediatamente desviada para o ciclo do jogo. Este teste é efetuado examinando-se o valor do indicador de bala atirada (indbal). Se ele é zero, significa que uma bala está em jogo. Se ele vale 1, não existe bala no vídeo e o resto do subprograma é executado.

A etapa seguinte consiste em posicionar horizontal e verticalmente o sprite correspondente ao desenho da bala, de modo que esta pareça sair do cano do fuzil. O parâmetro que define a posição horizontal tem o mesmo valor que aquele correspondente à posição do fuzil. O parâmetro que define a posição vertical tem por valor 154–16 = 138, 154 correspondendo à coordenada vertical do fuzil.

O indicador de bala atirada é então colocado em zero. Isto permite, por um lado, impedir que uma outra bala seja atirada e, por outro, que o

subprograma de deslocamento da bala (ver logo abaixo) seja regularmente chamado durante a execução do ciclo do jogo.

Endereço inicial **40157**
Enderço final **40194**
Total hexadecimal **2697**

```
3A279E      870 TIRO:  LD   A,(indbal)
3C          880        INC  A
3D          890        DEC  A
C8          900        RET  Z
21011B      910        LD   HL,6913
CD4A00      920        CALL 74
21091B      930        LD   HL,6921
CD4D00      940        CALL 77
21001B      950        LD   HL,6912
CD4A00      960        CALL 74
D611        970        SUB  17
21081B      980        LD   HL,6920
CD4D00      990        CALL 77
3E00       1000        LD   A,0
32279E     1010        LD   (indbal),A
C9         1020        RET
```

As linhas abaixo permitem testar o subprograma BALA. A bala aparece na saída do fuzil.

```
1000 DEF USR=40363
1005 A=USR (0)
1010 DEF USR=40157
1015 A=USR (0)
```

O SUBPROGRAMA "DESBAL"

Este subprograma gera o deslocamento no vídeo da bala que acaba de ser atirada. Este movimento é obtido diminuindo-se de uma unidade, a cada chamada, o valor da posição vertical corrente do sprite. O subprograma testa igualmente se a bala atingiu a borda superior do vídeo. Neste caso, o indicador de bala atirada (indbal) é posicionado pelo programa e o parâmetro que define a posição vertical da bala toma um valor tal que o sprite desaparece atrás da tela do vídeo.

Endereço inicial **40205**
Endereço final **40232**
Total hexadecimal **2400**

```
21081B    1130 DESBAL  LD    HL,6920
CD4A00    1140         CALL  74
3D        1150         DEC   A
CA1B9D    1160         JP    Z,CIMA
CD4D00    1170         CALL  77
C9        1180         RET
3EC8      1190 CIMA:   LD    A,200
21081B    1200         LD    HL,6920
CD4D00    1210         CALL  77
3E01      1220         LD    A,1
32279E    1230         LD    (indbal),A
C9        1240         RET
```

As linhas do programa abaixo permitem testar o subprograma. A bala se desloca constantemente de baixo para cima no vídeo.

```
1000 DEF USR=40363
1015 A=USR (0)
1020 DEF USR=40157
1025 A=USR (0)
1030 DEF USR=40205
1035 A=USR (0)
1040 GOTO 1035
```

O SUBPROGRAMA "TIRBAL"

O indicador de bala atirada (indbal) é testado permanentemente durante a execução do ciclo do jogo. Se ele é zero, o subprograma TIRBAL é então chamado. A primeira função deste subprograma é chamar três vezes em seguida o subprograma de deslocamento da bala (DESBAL). Esta adquire assim, um movimento muito rápido. O subprograma TIRBAL testa em seguida se no movimento deste deslocamento a bala tocou o invasor. Para isto, o valor do registrador de colisão que se encontra na memória VRAM é examinado. Este registrador é posicionado assim que dois sprites se superpõem, o que quer dizer, neste programa, logo que a bala toca o invasor. Ao contrário, se este registrador está em zero, o indicador de colisão (indcol) não está posicionado (ele vale então zero). Logo que uma colisão é produzida, o invasor recebe um tiro e uma explosão deve ser simulada chamando o subprograma EXPLO.

Endereço inicial 40082
Endereço final 40102
Total hexadecimal 2416

```
CD0D9D    420 TIRBAL    CALL DESBAL
CD0D9D    430           CALL DESBAL
CD0D9D    440           CALL DESBAL
CD3E01    450           CALL 318
CB6F      460           BIT  5,A
C8        470           RET  Z
3E00      480           LD   A,0
32269E    490           LD   (indcol),A
C9        500           RET
```

O SUBPROGRAMA "EXPLO"

O ciclo do jogo examina permanentemente o indicador de colisão. Se este valor for zero, o subprograma EXPLO, que simula visualmente uma explosão, é chamado. Este efeito espetacular é gerado da seguinte maneira: o vídeo é sempre reativado no modo 3 (modo multicolorido). Os quadros que se encontram na memória VRAM são nesse caso preenchidos por uma série de valores aleatórios que produzem uma fixação de quadrados de diferentes cores. O processo é repetido várias vezes.

Endereço inicial 40113
Endereço final 40146
Total hexadecimal 3587

```
3E03      610 EXPLO:    LD   A,3
32AFFC    620           LD   (64687),A
CD5F00    630           CALL 95
3E64      640           LD   A,100
210000    650           LD   HL,0
E5        660 EXPLO1    PUSH HL
F5        670           PUSH AF
110008    680           LD   DE,2048
01E803    690           LD   BC,1000
CD5C00    700           CALL 92
F1        710           POP  AF
E1        720           POP  HL
24        730           INC  H
3D        740           DEC  A
C2BE9C    750           JP   NZ,EXPLO1
C3409C    760           JP   INICIO
```

Este subprograma pode ser testado por meio das seguintes linhas:

```
1000 POKE 40000,201
1005 DEF USR=40113
1010 A=USR (0)
```

O SUBPROGRAMA DE ATRASO ("TEMPO")

Este subprograma permite moderar a execução de certas etapas a fim de que elas possam ser discernidas pelo jogador.

O ciclo de espera é realizado carregando-se o valor 255 no registrador A, após diminuí-lo em uma unidade até atingir o valor 0. Não é possível testar este subprograma tão facilmente quanto os anteriores. De fato, mesmo com um valor máximo de 255, a execução é ainda muito rápida para poder ser seguida passo a passo. Entretanto, no final deste capítulo, algumas indicações serão dadas para ilustrar o funcionamento do ciclo de espera.

Endereço inicial 40474
Endereço final 40480
Total hexadecimal 959

```
3EFF    2490 TEMPO:  LD    A,255
3D      2500 TEM:    DEC   A
C21C9E  2510         JP    NZ,TEM
C9      2520         RET
```

O CICLO DO JOGO

O ciclo do jogo realiza as seguintes operações:

1. Chamada do subprograma de deslocamento do invasor (DESINV).
2. Exame da situação do indicador de colisão (indcol). Se este estiver em zero, desvio para o subprograma EXPLO.
3. Exame da situação do indicador de bala atirada (indbal). Se este estiver em zero, desvio para o subprograma TIRBAL.
4. Se a tecla de deslocamento do cursor para a esquerda está ativada, chamada do subprograma de deslocamento do fuzil para a esquerda (FUZESQ).
5. Se a tecla de deslocamento do cursor para a direita está ativada, chamada do subprograma de deslocamento do fuzil para a direita (FUZDIR).
6. Se a barra de espaço é tocada, chamada do subprograma BALA.
7. Se a tecla X é pressionada, saída do programa e retorno para o BASIC.

O ciclo do jogo chama igualmente várias vezes o subprograma de atraso (TEMPO). Este é indispensável para que o deslocamento dos sprites possa ser percebido.

A maneira pela qual o programa detecta que uma tecla foi pressionada merece alguns comentários. O programa não possui nenhuma instrução que lhe permita saber qual tecla foi acionada. Ao contrário, ele pode testar se uma determinada tecla foi pressionada. Um subprograma do BIOS permite de fato cumprir esta missão. Antes que este subprograma possa ser chamado, é necessário que lhe sejam fornecidas algumas informações. A tabela abaixo vai permitir ilustrar o procedimento. Esta tabela possui 9 linhas numeradas de 0 a 8, algumas destas linhas contendo 8 teclas:

	7	6	5	4	3	2	1	0
0	7	6	5	4	3	2	1	0
1	;	]	[	\	=	-	9	8
2	B	A	£	/	.	,	'	`
3	J	I	H	G	F	E	D	C
4	R	Q	P	O	N	M	L	K
5	Z	Y	X	W	V	U	T	S
6	F3	F2	F1	CODE	CAP	GRAPH	CTRL	SHIFT
7	RETURN	SEL	BACK SPACE	STOP	TAB	ESC	F5	F4
8	CURSOR A DIREITA	CURSOR A BAIXO	CURSOR ACIMA	CURSOR À esquerda	DEL	INS	HOME	ESPAÇO

Suponhamos que se deseja saber se a tecla correspondente ao deslocamento do cursor para a esquerda foi pressionada. O procedimento é o seguinte:

- **Carregar o valor 8 para o registrador A** (8 corresponde, segundo o esquema precedente, ao número da linha sobre a qual se encontra a tecla a ser testada).
- **Chamar o subprograma em 321d (CALL 321)** (trata-se do endereço do subprograma que lê o número da linha que foi especificada).

Repetindo, o subprograma carrega para o registrador A um valor correspondente ao número da tecla sobre a linha especificada, que foi pressionada. O esquema anterior mostra, de fato, que as colunas são numeradas de 0 a 7. A tecla de deslocamento do cursor para a esquerda corresponde à coluna 4. Para testar se esta tecla foi pressionada, é suficiente utilizar a instrução:

BIT 4,A

A instrução seguinte deve ser:

CALL Z, subprograma de deslocamento do fuzil para a esquerda.

Se o resultado da instrução BIT 4,A é zero (quer dizer, se a tecla em questão foi presssionada), o subprograma em questão é chamado. A listagem do ciclo do jogo é dada abaixo:

Endereço inicial 40000
Endereço final 40071
Total hexadecimal 8681

```
CDAB9D    20 INICIO:  CALL  INIC
CD339D    30 CICLO    CALL  DESINV
CD1A9E    40          CALL  TEMPO
3A269E    50          LD    A,(indcol)
3C        60          INC   A
3D        70          DEC   A
CAB19C    80          JP    Z,EXPLO
3A279E    90          LD    A,(indbal)
3C       100          INC   A
3D       110          DEC   A
CC929C   120          CALL  Z,TIRBAL
3E08     130          LD    A,8
CD4101   140          CALL  321
CB67     150          BIT   4,A
CC7C9D   160          CALL  Z,FUZES0
CD1A9E   170          CALL  TEMPO
3E08     180          LD    A,8
CD4101   190          CALL  321
CB7F     200          BIT   7,A
CC929D   210          CALL  Z,FUZDIR
3E08     220          LD    A,8
CD4101   230          CALL  321
CB47     240          BIT   0,A
CCDD9C   250          CALL  Z,TIRO
CD1A9E   260          CALL  TEMPO
3E05     270          LD    A,5
CD4101   280          CALL  321
CB6F     290          BIT   5,A
C8       300          RET   Z
C3439C   310          JP    CICLO
```

O jogo agora pode ser totalmente testado digitando:

```
1000 DEF USR=40000
1005 A=USR (0)
```

Se se deseja compreender a influência do ciclo de atraso, é suficiente, para o suprimir, digitar a seguinte instrução:

POKE 40475,1

e rodar o programa. O endereço 40475 contém normalmente o valor 255, isto é, o valor que corresponde à demora.

7

CARACTERÍSTICAS DOS COMPUTADORES MSX

Este capítulo trata de alguns assuntos importantes que não dizem respeito diretamente à programação em linguagem de máquina. Uma obra de iniciação tal como esta não pode pretender abordar cada assunto em detalhe. As obras especializadas devem ser consultadas, para um conhecimento mais aprofundado sobre certos pontos particulares. Entretanto, do mesmo modo que não é indispensável conhecer o funcionamento do motor a pistão para poder guiar um carro, não é necessário conhecer os mínimos detalhes de uma característica particular de um computador para poder começar a utilizá-lo com proveito. Este capítulo será assim mais centralizado nas possibilidades de utilização, do que nas estruturas ou nos princípios de funcionamento.

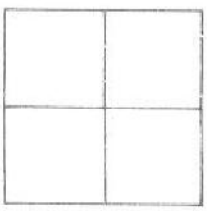
OS QUATRO MODOS DE VÍDEO

O único modo de vídeo que foi descrito para os programas apresentados até agora é o modo vídeo 1. Trata-se provavelmente do modo mais adaptado à realização de jogos de fliperama.

O modo vídeo 0 dispõe de 24 linhas de 40 caracteres. Ele se inicia por meio da instrução SCREEN 0. Somente duas cores são utilizáveis simultaneamente neste modo, uma para o fundo e outra para a superfície. A borda fixa-se necessariamente na cor do fundo. Este modo é essencialmente utilizado para os programas que não fazem chamada a gráficos e para aplicações tais como o tratamento de texto. Os sprites não podem ser utilizados no modo vídeo 0.

O modo 2 parece muito com o modo 1. Um máximo de 768 caracteres diferentes podem ser afixados simultaneamente no vídeo. Além disso, cada um dos oito bytes que determinam a forma de um caractere pode corresponder a uma combinação de duas cores quaisquer.

O modo 3, ou modo multicolorido, é de menor interesse e tem uma utilização mais delicada. Como no modo 1, o vídeo é dividido em 24 linhas de 32 caracteres, perfazendo um total de 768 posições de caracteres. Entretanto, ao invés de afixar os caracteres, este modo afixa os blocos de cores. Cada posição de caractere é dividida em quatro quadrados unidos, como mostra o esquema abaixo:



Cada quadrado pode ser afixado individualmente em uma das dezesseis cores disponíveis.

OS SPRITES

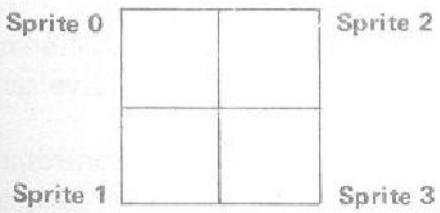
Um dos modos possíveis de afixar os sprites foi descrito no capítulo precedente. Os outros três modos disponíveis serão brevemente evocados aqui. As linhas seguintes permitem fazer aparecer no vídeo um sprite definido em 8 bytes e que se afixa em um tamanho normal (SPRITEATRIB representa o endereço da base da tabela dos atributos de sprites e SPRITEFORMA, o da base da tabela de formas dos sprites):

```
SCREEN 1
VDP(1)=224
VPOKE(SPRITEATRIB+0),100
VPOKE(SPRITEATRIB+1),50
VPOKE(SPRITEATRIB+2),0
VPOKE(SPRITEATRIB+3),10
VPOKE(SPRITEFORMA+0),255
VPOKE(SPRITEFORMA+1),129
VPOKE(SPRITEFORMA+2),129
VPOKE(SPRITEFORMA+3),129
VPOKE(SPRITEFORMA+4),129
VPOKE(SPRITEFORMA+5),129
VPOKE(SPRITEFORMA+6),129
VPOKE(SPRITEFORMA+7),255
```

Um sprite que tem a forma de uma caixa é assim afixado no meio do vídeo. Digitemos agora:

```
VDP(1) = 255
```

O sprite é, neste caso, afixado em tamanho duplo. O sprite afixado deste segundo modo pode ser deslocado no vídeo, modificando-se suas coordenadas horizontais e verticais. Rebatamos a instrução VDP(1)=224 para que o sprite volte ao seu tamanho original. O terceiro modo de afixação corresponde a um sprite gigante (definido por 32 bytes) que corresponde à associação de quatro sprites normais (definidos por 8 bytes), de acordo com o seguinte esquema:



Assim que os atributos do primeiro sprite são modificados por intermédio da instrução VPOKE, os três outros sprites também o são. O programa BASIC dado abaixo permite carregar na memória VRAM os dados de formas que correspondem a três sprites suplementares:

```
10 FOR A=8 TO 31
20 READ B
30 VPOKE(SPRITEFORMA+A),B
40 NEXT A
50 STOP
60 DATA 255, 195, 165, 153, 153, 165, 195, 255
70 DATA 255, 153, 153, 255, 255, 153, 153, 255
80 DATA 231, 165, 255, 36, 36, 255, 165, 231
```

Digitando em seguida VDP(1)=226, observa-se a afixação dos quatro sprites individuais sob a forma de um sprite gigante (definido por 32 bytes). Se modificarmos a posição horizontal ou vertical deste sprite, a forma gráfica desloca-se totalmente. Assim, é possível definir as formas de 64 sprites gigantes (cada um deles ocupando 32 bytes na tabela de formas). O endereço do primeiro byte que define os atributos de um sprite definido por 32 bytes pode ser conhecido através da seguinte fórmula:

(número do sprite x 16) + SPRITEATRIB

Os sprites afixados são igualmente numerados de 0 a 31 neste modo.

O último modo de afixação dos sprites permite afixar os sprites definidos por 32 bytes, mas a imagem é aumentada no vídeo por um fator de 2. Este modo pode ser obtido digitando-se:

VDP(1)=227

A MEMÓRIA DE VÍDEO RAM (OU VRAM)

A memória VRAM dispõe de 16 384 bytes, que são possíveis de ser respresentados de diferentes modos. Esta memória é controlada pelo processador de afixação no vídeo (VDP) que possui oito registradores para escrita, numerados de 0 a 7. O valor atribuído a alguns destes registradores determina a configuração da memória VRAM. Não se devem escolher estes valores ao acaso sob pena de bloquear o funcionamento normal do computador e de ter como única solução desligá-lo (perdendo-se nesta operação todas as informações armazenadas na memória RAM ou VRAM).

O registrador 0 do VDP serve para posicionar funções que não serão descritas aqui. Seu valor de inicialização não deve ser modificado sob pena de ocorrerem maiores problemas.

A principal função do registrador 1 é controlar o tipo e o tamanho dos sprites. Quando este registrador tem por valor 224, as formas dos sprites são definidas por 8 bytes e não são aumentadas no vídeo. O valor 225 corresponde igualmente a sprites definidos por 8 bytes, mas que são afixados no vídeo aumentados por um fator de 2. Quando o valor 226 é carregado no registrador 1 do VDP, os sprites são definidos por 32 bytes e não são aumentados no vídeo. Nestas condições, eles ocupam no vídeo um lugar que corresponde a quatro

sprites normais não aumentados. O valor 227 corresponde igualmente a sprites definidos por 32 bytes, mas desta vez aumentados por um fator de 2 (nas duas direções) no vídeo.

O valor contido no registrador 2 do VDP inicia o endereço da base da tabela de nomes, na memória VRAM. Este endereço foi designado por INICIOVIDEO nos capítulos anteriores. Existem dezesseis possibilidades diferentes, conforme o valor do registrador 2, para iniciar o endereço da base desta tabela:

VDP 2	Endereço da base da tabela de nomes
0	0
1	1024
2	2048
3	3072
4	4096
5	5120
6	6144
7	7168
8	8192
9	9216
10	10240
11	11264
12	12288
13	13312
14	14336
15	15360

O valor contido no registrador 3 do VDP determina o endereço da base da tabela de cores. Este registrador pode tomar qualquer valor compreendido entre 0 e 255. O endereço da base da tabela de cores é obtido multiplicando-se este valor decimal por 64. Por exemplo, se o registrador 3 do VDP tem por valor 21, o primeiro byte da tabela de cores se encontrará no endereço 21 x 64 = 1344.

O valor do registrador 4 do VDP determina o endereço da base da tabela de formas dos caracteres. Não existem mais do que oito possibilidades:

VDP 4	Endereço da base da tabela de formas dos caracteres
0	0
1	2048
2	4096
3	6144
4	8192
5	10240
6	12288
7	14336

O valor do registrador 5 do VDP determina o endereço da base da tabela dos atributos dos sprites. Este valor pode estar compreendido entre 0 e 127,

inclusive este último. O endereço correspondente é obtido multiplicando-se o valor contido no registrador 5 por 128. Se este valor é 45, por exemplo, o primeiro byte da tabela dos atributos dos sprites se encontrará no endereço 45 x 128 = 5760, na memória VRAM.

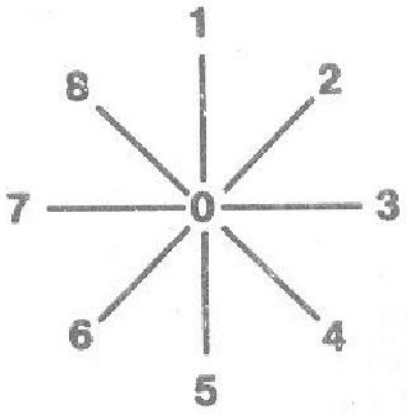
O registrador 5 define o endereço da base da tabela de formas dos sprites. Este registrador pode tomar os seguintes valores:

VDP 6	Endereço da base da tabela de formas dos sprites
0	0
1	2048
2	4096
3	6144
4	8192
5	10240
6	12288
7	14336

O registrador 7 do VDP é de pouco interesse para o programador de linguagem de máquina. Seu valor permite, particularmente, definir quais serão as cores (fundo e superfície) afixadas no modo vídeo 0, uma vez que uma instrução BASIC CLS tenha sido executada.

OS JOYSTICKS

Todos os computadores MSX dispõem de uma saída de *joystick* para um ou dois conectores. A posição do joystick pode ser testada por meio de dois subprogramas que se encontram na memória ROM. O primeiro subprograma apresenta a posição da haste de um determinado joystick. O número (1 ou 2) do joystick deve ser carregado no registrador A e o subprograma chamado para o endereço 213. Em retorno, o valor que corresponde à posição ligada se encontra no registrador A. O esquema abaixo indica a correspondência entre os valores apresentados e a posição da haste. O valor 0 corresponde à posição central.



O subprograma em linguagem de máquina escrito abaixo pode ser utilizado para ler a posição da haste. O valor 1 (que corresponde à posição do joystick n.º 1) é sempre carregado no registrador A. O subprograma é, em seguida, chamado para o endereço 213. O valor contido no registrador A é finalmente armazenado no endereço 50000:

Endereço inicial 40000
Endereço final 40008
Total hexadecimal 1007

```
3E01
CDD500
3250C3
C9
```

As linhas BASIC abaixo permitem testar este subprograma:

```
1000 SCREEN 1
1005 DEF USR=40000
1010 A=USR(0)
1015 B=PEEK(50000)
1020 PRINT B
1025 GOTO 1010
```

Quando este programa é executado, o valor que corresponde à posição da haste do joystick n.º 1 é mostrado no vídeo.

O segundo subprograma que se encontra na memória ROM serve para testar se o botão de acionamento de um dos dois joysticks está pressionado. Como no caso anterior, o valor 1 ou 2 deve sempre ser carregado no registrador A, conforme o número do joystick que deve ser testado. Em retorno, o valor 255 é carregado no registrador A se o botão de acionamento correspondente foi pressionado. Em caso contrário, este registrador contém o valor 0. O programa em linguagem de máquina dado abaixo ilustra a utilização deste subprograma na memória ROM. (Este programa não deve ser executado se não se dispõe de joysticks. De fato, o único meio de retornar ao BASIC consiste em pressionar o botão de acionamento do joystick.)

Endereço inicial 40000
Endereço final 40008
Total hexadecimal 1025

```
3E01
CDD800
3C
20F8
C9
```

Este programa pode ser testado por intermédio das seguintes linhas BASIC:

```
1000 DEF USR=40000
1005 A=USR (0)
1010 CLS
1015 PRINT "O botão do joystick foi pressionado"
```

O BIOS (Sistema de operação das entradas/saídas do BASIC)

O BIOS contém um grande número de subprogramas em linguagem de máquina que facilitam a escrita de outros programas. Uma lista dos subprogramas mais importantes é dada abaixo:

Objetivo: Preenchimento da memória VRAM.

Comentário: Logo que é chamado, este subprograma carrega dados de um local específico da memória VRAM. Por exemplo, o caractere número 32 pode ser carregado nos 768 bytes da tabela de nomes, no modo vídeo 1. Isto terá como efeito apagar o vídeo.

Procedimento: O endereço do primeiro byte do bloco a ser carregado deve ser passado ao registrador HL. O comprimento do bloco é carregado em BC e os dados (aqui, o valor 32) no registrador A.

Código de chamada: CD 56 00

Objetivo: Deslocamento de um bloco de dados da memória VRAM para a memória RAM.

Comentário: Este subprograma pode, por exemplo, ser utilizado para copiar na memória RAM o conteúdo da memória de vídeo.

Procedimento: O endereço VRAM do primeiro byte do bloco a ser deslocado deve ser carregado no registrador HL; o primeiro byte a ser utilizado na memória RAM deve ser carregado no registrador DE e o comprimento do bloco é passado como valor para o registrador BC.

Código de chamada: CD 59 00

Objetivo: Deslocamento de um bloco de dados da memória RAM para a memória VRAM.

Comentário: Este subprograma pode, por exemplo, ser utilizado para recopiar em memória VRAM um novo conjunto de ca-

racteres definido pelo programador e armazenado na memória RAM.

Procedimento: O endereço na memória RAM do primeiro byte do bloco a ser deslocado deve ser carregado no registrador HL; o primeiro byte a ser utilizado na memória VRAM deve ser carregado no registrador DE e o comprimento do bloco a ser deslocado é passado como valor no registrador BC.

Código de chamada: CD 5C 00

Objetivo: Leitura de um caractere via teclado.

Comentário: Assim que este subprograma é chamado, ele recebe um caractere via teclado, após retorna o número de código deste caractere no registrador A. Este subprograma é muito utilizado no modo conversacional.

Código de chamada: CD 9F 00

Objetivo: Posicionamento do cursor.

Comentário: Este subprograma coloca o cursor num local específico do vídeo. Sua função é então semelhante à instrução BASIC "LOCATE".

Procedimento: O número da coluna que corresponde à posição definida pelo cursor deve ser carregado no registrador H. No modo vídeo 1, este valor está compreendido entre 0 e 31. O número da linha que define a posição vertical do cursor deve ser carregado no registrador L.

Código de chamada: CD C6 00

Objetivo: Supressão da afixação no vídeo do significado das teclas de função.

Comentário: Assim que este subprograma é executado, a linha consagrada, na base do vídeo, à afixação do significado das teclas de função programáveis é apagada.

Código de chamada: CD CC 00

Objetivo: Restabelecimento da afixação do significado das teclas de funções programáveis.

Comentário: Assim que este subprograma é executado, as primeiras letras que correspondem aos valores atribuídos a estas teclas são afixadas na última linha do vídeo.

Código de cahmada: CD CF 00

Objetivo: Escrever na memória VRAM.

Comentário: Este subprograma equivale à instrução BASIC "VPOKE (endereço VRAM),n", onde o endereço VRAM é definido por um número compreendido entre 0 e 16 383, e n (compreendido entre 0 e 255) é o valor a ser carregado para este endereço.

Procedimento: O endereço VRAM deve ser carregado no registrador HL e o valor (compreendido entre 0 e 255) a ser escrito neste endereço deve ser carregado no registrador A.

Código de chamada: CD 4D 00

Objetivo: Leitura do conteúdo de um byte na memória VRAM.

Comentário: Este subprograma equivale à instrução BASIC "VPEEK(endereço VRAM)". No retorno deste subprograma, o registrador A contém o valor que foi lido no endereço especificado.

Procedimento: O endereço VRAM, cujo conteúdo deve ser lido, deve ser carregado no registrador HL.

Código de chamada: CD 4A 00

8

SUBPROGRAMAS UTILITÁRIOS

Este capítulo apresenta alguns subprogramas utilitários escritos em linguagem de máquina. Eles denotam, particularmente, a força desta linguagem. Estes subprogramas situam-se em lugares diferentes da memória RAM, de modo que é possível carregar vários deles simultaneamente.

Objetivo: Análise, da esquerda para a direita, dos caracteres contidos em uma linha de texto.

Comentário: Este subprograma permite analisar, partindo-se da esquerda, os caracteres que se encontram em qualquer uma das 24 linhas disponíveis no modo vídeo 1.

Procedimento: O endereço de memória 50000 deve conter um valor compreendido entre 0 e 23, que corresponda ao número da linha que deve ser analisada.

Endereço inicial 40000
Endereço final 40044
Total hexadecimal 4158

```
3A50C3    20          LD    A,(50000)
012000    30          LD    BC,32
211F18    40          LD    HL,6175
3C        50          INC   A
3D        60 CICLO    DEC   A
CA529C    70          JP    Z,CICL1
09        80          ADD   HL,BC
C34A9C    90          JP    CICLO
CD4A00   100 CICL1:   CALL  74
F5       110          PUSH  AF
011F00   120          LD    BC,31
2B       130 CICL2:   DEC   HL
CD4A00   140          CALL  74
23       150          INC   HL
CD4D00   160          CALL  77
2B       170          DEC   HL
0B       180          DEC   BC
79       190          LD    A,C
B0       200          OR    B
C2599C   210          JP    NZ,CICL2
F1       220          POP   AF
CD4D00   230          CALL  77
C9       240          RET
```

Objetivo: Análise, da direita para a esquerda, dos caracteres contidos em uma linha de texto.

Comentário: Este subprograma permite analisar, partindo-se da direita, os caracteres que se encontram em qualquer uma das 24 linhas disponíveis no modo vídeo 1.

Procedimento: O endereço de memória 50001 deve conter um valor compreendido entre 0 e 23 que corresponda ao número da linha que deve ser analisada.

Endereço inicial 40100
Endereço final 40144
Total hexadecimal 4420

```
3A51C3   20           LD    A,(50001)
012000   30           LD    BC,32
210018   40           LD    HL,6144
3C       50           INC   A
3D       60  CICL0    DEC   A
CAB69C   70           JP    Z,CICL1
09       80           ADD   HL,BC
C3AE9C   90           JP    CICL0
CD4A00  100  CICL1:   CALL  74
F5      110           PUSH  AF
011F00  120           LD    BC,31
23      130  CICL2:   INC   HL
CD4A00  140           CALL  74
2B      150           DEC   HL
CD4D00  160           CALL  77
23      170           INC   HL
0B      180           DEC   BC
79      190           LD    A,C
B0      200           OR    B
C2BD9C  210           JP    NZ,CICL2
F1      220           POP   AF
CD4D00  230           CALL  77
C9      240           RET
```

Objetivo: Análise, de baixo para cima, dos caracteres contidos em uma coluna de texto.

Comentário: Este subprograma permite analisar, partindo-se de baixo, os caracteres que se encontram em qualquer uma das 32 colunas do texto disponíveis no modo vídeo 1.

Procedimento: O endereço de memória 50002 deve conter um valor compreendido entre 0 e 31 que corresponda ao número da coluna que deve ser analisada.

Endereço inicial 40300
Endereço final 40345
Total hexadecimal 4540

```
3A52C3    20          LD    A,(50002)
3C        30          INC   A
210018    40          LD    HL,6144
3D        50 CICLO    DEC   A
CA7B9D    60          JP    Z,CICL1
23        70          INC   HL
C3739D    80          JP    CICLO
CD4A00    90 CICL1:   CALL  74
F5       100          PUSH  AF
011700   110          LD    BC,23
112000   120          LD    DE,32
19       130 CICL2:   ADD   HL,DE
CD4A00   140          CALL  74
ED52     150          SBC   HL,DE
CD4D00   160          CALL  77
19       170          ADD   HL,DE
0B       180          DEC   BC
79       190          LD    A,C
B0       200          OR    B
C2859D   210          JP    NZ,CICL2
F1       220          POP   AF
CD4D00   225          CALL  77
C9       230          RET
```

Objetivo: Análise, de cima para baixo, dos caracteres contidos em uma coluna de texto.

Comentário: Este subprograma permite analisar, partindo-se de cima, os caracteres que se encontram em qualquer uma das 32 colunas de texto disponíveis no modo vídeo 1.

Procedimento: O endereço de memória 50003 deve conter um valor compreendido entre 0 e 31, que corresponda ao número da coluna que deve ser analisada.

Endereço inicial 40200
Endereço final 40246
Total hexadecimal 4761

```
3A53C3    20          LD    A,(50003)
3C        30          INC   A
21E01A    40          LD    HL,6880
3D        50 CICL0    DEC   A
CA179D    60          JP    Z,CICL1
23        70          INC   HL
C30F9D    80          JP    CICL0
CD4A00    90 CICL1:   CALL  74
F5       100          PUSH  AF
011700   110          LD    BC,23
112000   120          LD    DE,32
ED52     130 CICL2:   SBC   HL,DE
CD4A00   140          CALL  74
19       150          ADD   HL,DE
CD4D00   160          CALL  77
ED52     170          SBC   HL,DE
0B       180          DEC   BC
79       190          LD    A,C
B0       200          OR    B
C2219D   210          JP    NZ,CICL2
F1       220          POP   AF
CD4D00   230          CALL  77
C9       240          RET
```

```
1000 SCREEN 1
1005 VDP(2)=6
1010 X=65
1015 FOR A=6144 TO 6880 STEP 32
1020 Y=X
1025 FOR B=0 TO 31
1030 VPOKE(A+B),Y
1035 Y=Y+1
1040 NEXT B
1045 X=X+1
1050 NEXT A
1055 POKE (50000),10
1060 DEF USR=40000
1065 A=USR(1):GOTO 1065
```

O programa BASIC acima pode ser utilizado para testar um dos quatro subprogramas em linguagem de máquina dados anteriormente. Apenas as linhas 1055 e 1060 têm de ser modificadas em função do programa que deve ser testado, e do número da linha ou coluna cujo conteúdo deva ser analisado.

Objetivo: Gerar um barulho de fuzil a laser.

Comentário: O barulho gerado por este programa é típico daqueles produzidos em jogos de fliperama tais como "Os invasores do espaço".

Endereço inicial 40600
Endereço final 40667
Total hexadecimal 7249

```
3E08     20          LD    A,8
1E0F     30          LD    E,15
CD9300   40          CALL  147
3E07     50          LD    A,7
1EFE     60          LD    E,254
CD9300   70          CALL  147
3E00     80          LD    A,0
1E6E     90          LD    E,110
CD9300   100         CALL  147
3E01     110         LD    A,1
1E00     120         LD    E,0
CD9300   130         CALL  147
1E6E     140         LD    E,110
3E00     150 CICLO   LD    A,0
CD9300   160         CALL  147
3EC8     170         LD    A,200
F5       180 TEMPO:  PUSH  A,F
3E0A     190         LD    A,10
3D       200 TEM:    DEC   A
C2C09E   210         JP    NZ,TEM
F1       220         POP   AF
3D       230         DEC   A
C2BD9E   240         JP    NZ,TEMPO
7B       250         LD    A,E
C606     260         ADD   A,6
CAD49E   270         JP    Z,FIM
3D       280         DEC   A
5F       290         LD    E,A
C3B69E   300         JP    CICLO
3E07     310 FIM:    LD    A,7
1EFF     320         LD    E,255
CD9300   330         CALL  147
C9       340         RET
```

Objetivo: Emissão de um barulho simulando a explosão de uma bomba.

Comentário: O programa começa por gerar o assobio da bomba e depois o barulho de sua explosão.

Endereço inicial 40400
Endereço final 40514
Total hexadecimal 13909

```
3E07      20
1EFE      30
CD9300    40
3E0A      50
1E0F      60
CD9300    70
1E28      80
3E00      90 CICLO  LD    A,0
CD9300   100        CALL  147
3E0A     110        LD    A,10
F5       120 TEMPO: PUSH  AF
3EFF     130        LD    A,255
3D       140 TEM:   DEC   A
C2EA9D   150        JP    NZ,TEM
F1       160        POP   AF
3D       170        DEC   A
C2E79D   180        JP    NZ,TEMPO
7B       190        LD    A,E
D696     200        SUB   150
CAFF9D   210        JP    Z,SEG
C697     220        ADD   A,151
5F       230        LD    E,A
C3E09D   240        JP    CICLO
3E00     250 SEG:   LD    A,0
1E00     260        LD    E,0
CD9300   270        CALL  147
3E07     280        LD    A,7
1EF7     290        LD    E,247
CD9300   300        CALL  147
3E00     310        LD    A,0
F5       320 SEG1:  PUSH  AF
5F       330        LD    E,A
CD9300   340        CALL  147
3E32     350        LD    A,50
```

```
F5       360 LUCIA:  PUSH  AF
3EFF     370         LD    A,255
3D       380 PAULO:  DEC   A
C2199E   390         JP    NZ,PAULO
F1       400         POP   AF
3D       410         DEC   A
C2169E   420         JP    NZ,LUCIA
F1       430         POP   AF
D61F     440         SUB   31
CA2D9E   450         JP    Z,LDEL
C620     460         ADD   A,32
C30F9E   470         JP    SEGI
3E64     480 LDEL:   LD    A,100
F5       490 LDEL1:  PUSH  AF
3EFF     500         LD    A,255
3D       510 LDEL2:  DEC   A
C2329E   520         JP    NZ,LDEL2
F1       530         POP   AF
3D       540         DEC   A
C22F9E   550         JP    NZ,LDEL1
3E07     560         LD    A,7
1EFF     570         LD    E,255
CD9300   580         CALL  147
C9       590         RET
```

APÊNDICE 1

CÓDIGOS DE MÁQUINA DO MICROPROCESSADOR Z80

ADC A, (HL)	8E
ADC A, (IX + d)	DD8Ed
ADC A, (IY + d)	FD8Ed
ADC A, A	8F
ADC A, B	88
ADC A, C	89
ADC A, D	8A
ADC A, E	8B
ADC A, H	8C
ADC A, L	8D
ADC A, n	CEn
ADC HL, BC	ED4A
ADC HL, DE	ED5A
ADC HL, HL	ED6A
ADC HL, SP	ED7A
ADD A, (HL)	86
ADD A, (IX + d)	DD86d
ADD A, (IY + d)	FD86d
ADD A, A	87
ADD A, B	80
ADD A, C	81
ADD A, D	82
ADD A, E	83
ADD A, H	84
ADD A, L	85
ADD A, n	C6n
ADD HL, BC	09
ADD HL, DE	19
ADD HL, HL	29
ADD HL, SP	39
ADD IX, BC	DD09
ADD IX, DE	DD19
ADD IX, IX	DD29
ADD IX, SP	DD39
ADD IY, BC	FD09
ADD IY, DE	FD19
ADD IY, IY	FD29
ADD IY, SP	FD39
AND(HL)	A6
AND (IX + d)	DDA6d
AND (IY + d)	FDA6d
AND A	A7
AND B	A0
AND C	A1
AND D	A2
AND E	A3
AND H	A4
AND L	A5
AND n	E6n
BIT 0, (HL)	CB46
BIT 0, (IX + D)	DDCBd46
BIT 0, (IY + d)	FDCBd46
BIT 0, A	CB47
BIT 0, B	CB40
BIT 0, C	CB41
BIT 0, D	CB42
BIT 0, E	CB43
BIT 0, H	CB44
BIT 0, L	CB45
BIT 1, (HL)	CB4E
BIT 1, (IX + d)	DDCBd4E
BIT 1, (IY + d)	FDCBd4E
BIT 1, A	CB4F
BIT 1, B	CB48
BIT 1, C	CB49
BIT 1, D	CB4A
BIT 1, E	CB4B
BIT 1, H	CB4C
BIT 1, L	CB4D
BIT 2, (HL)	CB56
BIT 2, (IX + d)	DDCBd56
BIT 2, (IY + d)	FDCBd56
BIT 2, A	CB57
BIT 2, B	CB50
BIT 2, C	CB51
BIT 2, D	CB52
BIT 2, E	CB53
BIT 2, H	CB54
BIT 2, L	CB55
BIT 3, (HL)	CB5E
BIT 3, (IX + d)	DDCBd5E
BIT 3, (IY + d)	FDCBd5E

BIT 3, A	CB5F
BIT 3, B	CB58
BIT 3, C	CB59
BIT 3, D	CB5A
BIT 3, E	CB5B
BIT 3, H	CB5C
BIT 3, L	CB5D
BIT 4, (HL)	CB66
BIT 4, (IX + d)	DDCBd66
BIT 4, (IY + d)	FDCBd66
BIT 4, A	CB67
BIT 4, B	CB60
BIT 4, C	CB61
BIT 4, D	CB62
BIT 4, E	CB63
BIT 4, H	CB64
BIT 4, L	CB65
BIT 5, (HL)	CB6E
BIT 5, (IX + d)	DDCBd6E
BIT 5, (IY + D)	FDCBd6E
BIT 5, A	CB6F
BIT 5, B	CB68
BIT 5, C	CB69
BIT 5, D	CB6A
BIT 5, E	CB6B
BIT 5, H	CB6C
BIT 5, L	CB6D
BIT 6, (HL)	CB76
BIT 6, (IX + d)	DDCBd76
BIT 6, (IY + d)	FDCBd76
BIT 6, A	C877
BIT 6, B	CB70
BIT 6, C	CB71
BIT 6, D	CB72
BIT 6, E	CB73
BIT 6, H	CB74
BIT 6, L	CB75
BIT 7, (HL)	CB7E
BIT 7, (IX + d)	DDCBd7E
BIT 7, (IY + d)	FDCBd7E
BIT 7, A	CB7F
BIT 7, B	CB78
BIT 7, C	CB79
BIT 7, D	CB7A
BIT 7, E	CB7B
BIT 7, H	CB7C
BIT 7, L	CB7D
CALL C, nn	DCnn
CALL M, nn	FCnn
CALL NC, nn	D4nn
CALL nn	CDnn
CALL NZ, nn	C4nn
CALL P, nn	F4nn
CALL PE, nn	ECnn
CALL PO, nn	E4nn
CALL Z, nn	CCnn
CCF	3F
CP (HL)	BE
CP (IX + d)	DDBEd
CP (IY + d)	FDBEd
CP A	BF
CP B	B8
CP C	B9
CP D	BA
CP E	BB
CP H	BC
CP L	BD
CP n	FEn
CPD	EDA9
CPDR	EDB9
CPI	EDA1
CPIR	EDB1
CPL	2F
DAA	27
DEC (HL)	35
DEC (IX + d)	DD35d
DEC (IY + d)	FD35d
DEC A	3D
DEC B	05
DEC BC	0B
DEC C	0D
DEC D	15
DEC DE	1B
DEC E	1D
DEC H	25
DEC HL	2B
DEC IX	DD2B
DEC IY	FD2B
DEC L	2D
DEC SP	3B
DI	F3
DJNZ, d	10d
E1	FB
EX (SP), HL	E3
EX (SP), IX	DDE3
EX (SP), IY	FDE3
EX AF, AF	08
EX DE, HL	EB
EXX	D9
HALT	76

IM 0	ED46
IM 1	ED56
IM 2	ED5E
IN A, (C)	ED78
IN A, (n)	DBn
IN B, (C)	ED40
IN C, (C)	ED48
IN D, (C)	ED50
IN E, (C)	ED58
IN H, (C)	ED60
IN L, (C)	ED68
INC (HL)	34
INC (IX + d)	DD34d
INC (IY + d)	FD34d
INC A	3C
INC B	04
INC BC	03
INC C	0C
INC D	14
INC DE	13
INC E	1C
INC H	24
INC HL	23
INC IX	DD23
INC IY	FD23
INC L	2C
INC SP	33
IND	EDAA
INDR	EDBA
INI	EDA2
INIR	EDB2
JP (HL)	E9
JP (IX)	DDE9
JP (IY)	FDE9
JP C, nn	DAnn
JP M, nn	FAnn
JP NC, nn	D2nn
JP nn	C3nn
JP NZ, nn	C2nn
JP P, nn	F2nn
JP PE, nn	EAnn
JP PO, nn	E2nn
JP Z, nn	CAnn
JR C, d	38d
JR, d	18d
JR NC, d	30d
JR NZ, d	20d
JR Z, d	28d
LD (BC), A	02
LD (DE), A	12
LD (HL), A	77
LD (HL), B	70
LD (HL), C	71
LD (HL), D	72
LD (HL), E	73
LD (HL), H	74
LD (HL), L	75
LD (HL), n	36n
LD (IX + d), A	DD77d
LD (IX + d), B	DD70d
LD (IX + d), C	DD71d
LD (IX + d), D	DD72d
LD (IX + d), E	DD73d
LD (IX + d), H	DD74d
LD (IX + d), L	DD75d
LD (IX + d), n	DD36dn
LD (IY + d), A	FD77d
LD (IY + d), B	FD70d
LD (IY + d), C	FD71d
LD (IY + d), D	FD72d
LD (IY + d), E	FD73d
LD (IY + d), H	FD74d
LD (IY + d), L	FD75d
LD (IY + d), n	FD36dn
LD (nn), A	32nn
LD (nn), BC	ED43nn
LD (nn), DE	ED53nn
LD (nn), HL	22nn
LD (nn), IX	DD22nn
LD (nn), IY	FD22nn
LD (nn), SP	ED73nn
LD A, (BC)	0A
LD A, (DE)	1A
LD A, (HL)	7E
LD A, (IX + d)	DD7Ed
LD A, (IY + d)	FD7Ed
LD A, (nn)	3Ann
LD A, A	7F
LD A, B	78
LD A, C	79
LD A, D	7A
LD A, E	7B
LD A, H	7C
LD A, I	ED57
LD A, L	7D
LD A, n	3En
LD B, (HL)	46
LD B, (IX + d)	DD46d
LD B, (IY + d)	FD46d
LD B, A	47

LD B, B	40
LD B, C	41
LD B, D	42
LD B, E	43
LD B, H	44
LD B, L	45
LD B, n	06n
LD B, C (nn)	ED4Bnn
LD BC (nn)	01nn
LD C, (HL)	4E
LD C, (IX + d)	DD4Ed
LD C, (IY + d)	FD4Ed
LD C, A	4F
LD C, B	48
LD C, C	49
LD C, D	4A
LD C, E	4B
LD C, H	4C
LD C, L	4D
LD C, n	0En
LD D, (HL)	56
LD D, (IX + d)	DD56d
LD D, (IY + d)	FD56d
LD D, A	57
LD D, B	50
LD D, C	51
LD D, D	52
LD D, E	53
LD D, H	54
LD D, L	55
LD D, n	16n
LD DE, (nn)	ED58nn
LD DE, nn	11nn
LD E, (HL)	5E
LD E, (IX + d)	DD5Ed
LD E, (IY + d)	FD5Ed
LD E, A	5F
LD E, B	58
LD E, C	59
LD E, D	5A
LD E, E	5B
LD E, H	5C
LD E, L	5D
LD E, n	1En
LD H, (HL)	66
LD H, (IX + d)	DD66d
LD H, (IY + d)	FF66d
LD H, A	67
LD H, B	60
LD H, C	61
LD H, D	62
LD H, E	63
LD H, H	64
LD H, L	65
LD H, n	26n
LD HL, (nn)	2Ann
LD HL, nn	21nn
LD I, A	ED47
LD IX, (nn)	DD2Ann
LD IX, nn	DD21nn
LD IY, (nn)	FD2Ann
LD IY, nn	FD21nn
LD L, (HL)	6E
LD L, (IX + d)	DD6Ed
LD L, (IY + d)	FD6Ed
LD L, A	6F
LD L, B	68
LD L, C	69
LD L, D	6A
LD L, E	6B
LD L, H	6C
LD L, L	6D
LD L, n	2En
LD SP, (nn)	ED7Bnn
LD SP, HL	F9
LD SP, IX	DDF9
LD SP, IY	FDF9
LD SP, nn	31nn
LDD	EDA8
LDDR	EDB8
LDI	EDAO
LDIR	EDB0
NEG	ED44
NOP	00
OR (HL)	B6
OR (IX + d)	DDB6d
OR (IY + d)	FDB6d
OR A	B7
OR B	B0
OR C	B1
OR D	B2
OR E	B3
OR H	B4
OR L	B5
OR n	F6n
OTDR	EDBB
OTIR	EDB3
OUT (C), A	ED79
OUT (C), B	ED41
OUT (C), C	ED49

OUT (C), D	ED51
OUT (C), E	ED59
OUT (C), H	ED61
OUT (C), L	ED69
OUT (n), A	D3n
OUTD	EDAB
OUTI	EDA3
POPAF	F1
POP BC	C1
POP DE	D1
POP HL	E1
POP IX	DDE1
POP IY	FDE1
PUSH AF	F5
PUSH BC	C5
PUSH DE	D5
PUSH HL	E5
PUSH IX	DDE5
PUSH IY	FDE5
RES 0, (HL)	CB86
RES 0, (IX + d)	DDCBd86
RES 0, (IY + d)	FDCBd86
RES 0, A	CB87
RES 0, B	CB80
RES 0, C	CB81
RES 0, D	CB82
RES 0, E	CB83
RES 0, H	CB84
RES 0, L	CB85
RES 1, (HL)	CB8E
RES 1, (IX + d)	DDCBd8E
RES 1, (IY + d)	FDCBd8E
RES 1, A	CB8F
RES 1, B	CB88
RES 1, C	CB89
RES 1, D	CB8A
RES 1, E	CB8B
RES 1, H	CB8C
RES 1, L	CB8D
RES 2, (HL)	CB96
RES 2, (IX + d)	DDCBd96
RES 2, (IY + d)	FDCBd96
RES 2, A	CB97
RES 2, B	CB90
RES 2, C	CB91
RES 2, D	CB92
RES 2, E	CB93
RES 2, H	CB94
RES 2, L	CB95
RES 3, (HL)	CB9E
RES 3, (IX + d)	DDCBd9E
RES 3, (IY + d)	FDCBd9E
RES 3, A	CB9F
RES 3, B	CB98
RES 3, C	CB99
RES 3, D	CB9A
RES 3, E	CB9B
RES 3, H	CB9C
RES 3, L	CB9D
RES 4, (HL)	CBA6
RES 4, (IX + d)	DDCBdA6
RES 4, (IY + d)	FDCBdA6
RES 4, A	CBA7
RES 4, B	CBA0
RES 4, C	CBA1
RES 4, D	CBA2
RES 4, E	CBA3
RES 4, H	CBA4
RES 4, L	CBA5
RES 5, (HL)	CBAE
RES 5, (IX + d)	DDCBdAE
RES 5, (IY + d)	FDCBdAE
RES 5, A	CBAF
RES 5, B	CBA8
RES 5, C	CBA9
RES 5, D	CBAA
RES 5, E	CBAB
RES 5, H	CBAC
RES 5, L	CBAD
RES 6, (HL)	CBB6
RES 6, (IX + d)	DDCBdB6
RES 6, (IY + d)	FDCBdB6
RES 6, A	CBB7
RES 6, B	CBB0
RES 6, C	CBB1
RES 6, D	CBB2
RES 6, E	CBB3
RES 6, H	CBB4
RES 6, L	CBB5
RES 7, (HL)	CBBE
RES 7, (IX + d)	DDCBdBE
RES 7, (IY + d)	FDCBdBE
RES 7, A	CBBF
RES 7, B	CBB8
RES 7, C	CBB9
RES 7, D	CBBA
RES 7, E	CBBB
RES 7, H	CBBC
RES 7, L	CBBD
RET	C9

RET C	D8	RRC D	CB0A
RET M	F8	RRC E	CB0B
RET NC	D0	RRC H	CB0C
RET NZ	C0	RRC L	CB0D
RET P	F0	RRC A	0F
RET PE	E8	RRD	ED67
RET P0	E0	RST 0	C7
RET Z	C8	RST10H	D7
RETI	ED4D	RST 18H	DF
RETN	ED45	RST 20H	E7
RL (HL)	CB16	RST 28H	EF
RL (IX + d)	DDCBd16	RST 30H	F7
RL (IY + d)	FDCBd16	RST 38H	FF
RL A	CB17	RST 8	CF
RL B	CB10	SBC A, (HL)	9E
RL C	CB11	SBC A, (IX + d)	DD9Ed
RL D	CB12	SBC A, (IY + d)	FD9Ed
RL E	CB13	SBC A, A	9F
RL H	CB14	SBC A, B	98
RL L	CB15	SBC A, C	99
RLA	17	SBC A, D	9A
RLC (HL)	CB06	SBC A, E	9B
RLC (IX + d)	DDCBd06	SBC A, H	9C
RLC (IY + d)	FDCBd06	SBC A, L	9D
RLC A	CB07	SBC A, n	DEn
RLC B	CB00	SBC HL, BC	ED42
RLC C	CB01	SBC HL, DE	ED52
RLC D	CB02	SBC HL, HL	ED62
RLC E	CB03	SBC HL, SP	ED72
RLC H	CB04	SCF	37
RLC L	CB05	SET 0, (HL)	CBC6
RLCA	07	SET 0, (IX + d)	DDCBdC6
RLD	ED6F	SET 0, (IY + d)	FDCBdC6
RR (HL)	CB1E	SET 0, A	CBC7
RR (IX + d)	DDC8d1E	SET 0, B	CBC0
RR (IY + d)	FDCBd1E	SET 0, C	CBC1
RR A	CB1F	SET 0, D	CBC2
RR B	CB18	SET 0, E	CBC3
RR C	CB19	SET 0, H	CBC4
RR D	CB1A	SET 0, L	CBC5
RR E	CB1B	SET 1, (HL)	CBCE
RR H	CB1C	SET 1, (IX + d)	DDCBdCE
RR L	CB1D	SET 1, (IY + d)	FDCBdCE
RRA	1F	SET 1, A	CBCF
RRC (HL)	CB0E	SET 1, B	CBC8
RRC (IX + d)	DDCBd0E	SET 1, C	CBC9
RRC (IY + d)	FDCBd0E	SET 1, D	CBCA
RRC A	CB0F	SET 1, E	CBCB
RRC B	CB08	SET 1, H	CBCC
RRC C	CB09	SET 1, L	CBCD

SET 2, (HL)	CBD6
SET 2, (IX + d)	DDCBdD6
SET 2, (IY + d)	FDCBdD6
SET 2, A	CBD7
SET 2, B	CBD0
SET 2, C	CBD1
SET 2, D	CBD2
SET 2, E	CBD3
SET 2, H	CBD4
SET 2, L	CBD5
SET 3, (HL)	CBDE
SET 3, (IX + d)	DDCBdDE
SET 3, (IY + d)	FDCBdDE
SET 3, A	CBDF
SET 3, B	CBD8
SET 3, C	CBD9
SET 3, D	CBDA
SET 3, E	CBDB
SET 3, H	CBDC
SET 3, L	CBDD
SET 4, (HL)	CBE6
SET 4, (IX + d)	DDCbE6
SET 4, (IY + d)	FDCBdE6
SET 4, A	CBE7
SET 4, B	CBE0
SET 4, C	CBE1
SET 4, D	CBE2
SET 4, E	CBE2
SET 4, H	CBE4
SET 4, L	CBE5
SET 5, (HL)	CBEE
SET 5, (IX + d)	DDCBdEE
SET 5, (IY + d)	FDCBdEE
SET 5, A	CBEF
SET 5, B	CBE8
SET 5, C	CBE9
SET 5, D	CBEA
SET 5, E	CBEB
SET 5, H	CBEC
SET 5, L	CBED
SET 6, (HL)	CBF6
SET 6, (IX + d)	DDCBdF6
SET 6, (IY + d)	FDCBdF6
SET 6, A	CBF7
SET 6, B	CBF0
SET 6, C	CBF1
SET 6, D	CBF2
SET 6, E	CBF3
SET 6, H	CBF4
SET 6, L	CBF5
SET 7, (HL)	CBFE
SET 7, (IX + d)	DDCBdFE
SET 7, (IY + d)	FDCBdFE
SET 7, A	CBFF
SET 7, B	CBF8
SET 7, C	CBF9
SET 7, D	CBFA
SET 7, E	CBFB
SET 7, H	CBFC
SET 7, L	CBFD
SLA (HL)	CB26
SLA (IX + d)	DDCBd26
SLA (IY + d)	FDCBd26
SLA A	CB27
SLA B	CB20
SLA C	CB21
SLA D	CB22
SLA E	CB23
SLA H	CB24
SLA L	CB25
SRA (HL)	CB2E
SRA (IX + d)	DDCBd2E
SRA (IY + d)	FDCBd2E
SRA A	CB2F
SRA B	CB28
SRA C	CB29
SRA D	CB2A
SRA E	CB2B
SRA H	CB2C
SRA L	CB2D
SRL (HL)	CB3E
SRL (IX + d)	DDCBd3E
SRL (IY + d)	FDCBd3E
SRL A	CB3F
SRL B	CB38
SRL C	CB39
SRL D	CB3A
SRL E	CB3B
SRL H	CB3C
SRL L	CB3D
SUB (HL)	96
SUB (IX + d)	DD96d
SUB (IY + d)	FD96d
SUB A	97
SUB B	90
SUB C	91
SUB D	92
SUB E	93
SUB H	94
SUB L	95

SUB n	D6n	XOR C	A9
XOR (HL)	AE	XOR D	AA
XOR (IX + d)	DDAEd	XOR E	AB
XOR (IY + d)	FDAEd	XOR H	AC
XOR A	AF	XOR L	AD
XOR B	A8	XOR n	EEn

APÊNDICE 2

QUADRO DE CONVERSÃO HEXADECIMAL/DECIMAL

	0	1	2	3	4	5	6	7	8	9	0A	0B	0C	0D	0E	0F
0	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
2	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
3	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
4	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79
5	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95
6	96	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111
7	112	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127
8	128	129	130	131	132	133	134	135	136	137	138	139	140	141	142	143
9	144	145	146	147	148	149	150	151	152	153	154	155	156	157	158	159
A	160	161	162	163	164	165	166	167	168	169	170	171	172	173	174	175
B	176	177	178	179	180	181	182	183	184	185	186	187	188	189	190	191
C	192	193	194	195	196	197	198	199	200	201	202	203	204	205	206	207
D	208	209	210	211	212	213	214	215	216	217	218	219	220	221	222	223
E	224	225	226	227	228	229	230	231	232	233	234	235	236	237	238	239
F	240	241	242	243	244	245	246	247	248	249	250	251	252	253	254	255

APÊNDICE 3

O SISTEMA BINÁRIO

Ainda que o conhecimento do sistema binário não seja indispensável à aprendizagem da programação em linguagem de máquina, pode revelar-se útil para a resolução de certos problemas específicos. O princípio da notação binária é, aliás, relativamente fácil de compreender. No capítulo consagrado ao modo de armazenar os números, foi indicado que cada local de memória (isto é, cada byte) podia receber um número cujo valor decimal está compreendido entre 0 e 255. Isto resulta da própria definição do byte. Um byte é formado da associação de oito unidades elementares chamadas bits (a palavra bit é a contração da expressão anglo-saxônica *binary digit* que significa algarismo binário). Os bits que constituem um byte são numerados da seguinte maneira:

Oito bits formam um byte

7	6	5	4	3	2	1	0

Estes oito bits podem ser considerados como se representassem interruptores. Um interruptor pode ser aberto ou fechado. Quando ele está aberto, a corrente não passa e o bit correspondente tem valor zero. Quando ele está fechado ou, por analogia, quando a corrente passa, o bit correspondente tem valor 1 (diz-se, neste caso, que o bit está "posicionado"). Entretanto, do mesmo modo que no caso do número decimal 33, por exemplo, os dois algarismos 3 não têm a mesma importância (eles não têm o mesmo "peso"), o número do bit em um byte determina seu peso no valor dado ao byte. O diagrama abaixo dá o valor decimal de um bit posicionado em um byte:

Valor do bit	128	64	32	16	8	4	2	1
Número do bit	7	6	5	4	3	2	1	0

Assim, quando o bit 4 está posicionado, 16 é o valor decimal do byte. Em outros termos, se os bits 1 e 3 de um byte estão posicionados enquanto todos os outros não estão, este byte terá por valor o decimal 10. De fato, o bit 1, quando está posicionado, corresponde a 2 (decimal) e o bit 3 a 8. Quando todos os bits de um byte estão posicionados, o valor decimal é 128+64+32+16+8+4+2+1 = 255. No caso contrário, em que todos os bits são zero, o byte vale naturalmente zero. É por isso que é possível dar a um byte qualquer valor decimal compreendido entre 0 e 255. Quando se atribui um valor decimal a um byte, por meio das instruções PEEK e POKE, ou de seu

equivalente em linguagem de máquina, o computador efetua, ele próprio, a conversão entre os sistemas decimal e binário, a fim de saber quais bits do byte devem ser posicionados para representar o valor decimal que se deseja armazenar.

Se os bits 1, 4 e 6 de um byte estão posicionados, qual é o valor decimal deste byte?

Quais bits devem ser posicionados para que o valor decimal de um byte corresponda a 67?

APÊNDICE 4

DESENHO DOS CARACTERES E DOS SPRITES

O programa abaixo permite desenhar muito facilmente os caracteres ou os sprites, que podem ser, em seguida, utilizados em outros programas. Foi mencionado anteriormente que a forma de cada caractere de jogo MSX era definida por oito bytes. Este programa permite atribuir os valores de sua escolha a cada um destes oito bytes de forma. Ele se destina a uma utilização posterior no modo vídeo 1. O programa pode ser entrado por meio do ENTHEX e deve ser verificado como foi indicado anteriormente.

Endereço inicial 40000
Endereço final 40702
Total hexadecimal 85482

```
011800  110 START:  LD    BC,24
11F803  120         LD    DE,1016
215F9D  130         LD    HL,IXSTR
CD5C00  140         CALL  92
3E01    150         LD    A,1
32E3D6  160         LD    (CURCHR),A
32E8D6  170         LD    (VER),A
32E9D6  180         LD    (HOR),A
210218  190         LD    HL,6146
22E6D6  200         LD    (CURSOR),HL
21A08C  210         LD    HL,SETSTR
22E4D6  220         LD    (CHRADD),HL
3E80    230         LD    A,128
32E2D6  240         LD    (CHBIT),A
21D8D6  250         LD    HL,GRID
22E0D6  260         LD    (BYTE),HL
CD849D  270         CALL  CHRGRD
CD909D  280         CALL  DISCH
3E7F    290         LD    A,127
210218  300         LD    HL,6146
CD4D00  310         CALL  77
3E31    320         LD    A,49
210C18  330         LD    HL,6156
CD4D00  340         CALL  77
```

```
3E30      350          LD    A,48
210B18    360          LD    HL,6155
CD4D00    370          CALL  77
3E30      380          LD    A,48
210A18    390          LD    HL,6154
CD4D00    400          CALL  77
CD779D    410          CALL  RAM2V
3E82      420          LD    A,130
218B18    430          LD    HL,6283
CD4D00    440          CALL  77
00        450 LOOP:    NOP
3E07      460          LD    A,7
CD4101    470          CALL  321
CB57      480          BIT   2,A
C8        490          RET   Z
3E08      500          LD    A,8
CD4101    510          CALL  321
CB47      520          BIT   0,A
CAEC9D    530          JP    Z,SWITCH
CB7F      540          BIT   7,A
CA3D9D    550          JP    Z,RIGHT
CB77      560          BIT   6,A
CAFB9C    570          JP    Z,CDOWN
CB6F      580          BIT   5,A
CAD89C    590          JP    Z,CUP
CB67      600          BIT   4,A
CA1F9D    610          JP    Z,LEFT
3E04      620          LD    A,4
CD4101    630          CALL  321
CB6F      640          BIT   5,A
CA8B9E    650          JP    Z,PREV
CB5F      660          BIT   3,A
CA329E    670          JP    Z,NEXT
C39F9C    680          JP    LOOP
3AE8D6    690 CUP:     LD    A, (VER)
3D        700          DEC   A
CA9F9C    710          JP    Z,LOOP
32E8D6    720          LD    (VER),A
2AE0D6    730          LD    HL,(BYTE)
2B        740          DEC   HL
22E0D6    750          LD    (BYTE),HL
2AE6D6    760          LD    HL,(CURSOR)
012000    770          LD    BC,32
A7        780          AND   A
ED42      790          SBC   HL,BC
```

```
22E6D6   800          LD    (CURSOR),HL
CD909D   810          CALL  DISCH
C39F9C   820          JP    LOOP
3AE8D6   830 CDOWN:   LD    A,(VER)
D608     840          SUB   8
CA9F9C   850          JP    Z,LOOP
C609     860          ADD   A,9
32E8D6   870          LD    (VER),A
2AE0D6   880          LD    HL,(BYTE)
23       890          INC   HL
22E0D6   900          LD    (BYTE),HL
2AE6D6   910          LD    HL,(CURSOR)
012000   920          LD    BC.32
09       930          ADD   HL,BC
22E6D6   940          LD    (CURSOR),HL
CD909D   950          CALL  DISCH
C39F9C   960          JP    LOOP
3AE9D6   970 LEFT:    LD    A,(HOR)
3D       980          DEC   A
CA9F9C   990          JP    Z,LOOP
32E9D6   1000         LD    (HOR),A
3AE2D6   1010         LD    A,(CHBIT)
87       1020         ADD   A,A
32E2D6   1030         LD    (CHBIT),A
2AE6D6   1040         LD    HL,(CURSOR)
2B       1050         DEC   HL
22E6D6   1060         LD    (CURSOR),HL
CD909D   1070         CALL  DISCH
C39F9C   1080         JP    LOOP
3AE9D6   1090 RIGHT:  LD    A,(HOR)
D608     1100         SUB   8
CA9F9C   1110         JP    Z,LOOP
C609     1120         ADD   A,9
32E9D6   1130         LD    (HOR),A
3AE2D6   1140         LD    A,(CHBIT)
CB0F     1150         RRC   A
32E2D6   1160         LD    (CHBIT),A
2AE6D6   1170         LD    HL,(CURSOR)
23       1180         INC   HL
22E6D6   1190         LD    (CURSOR),HL
CD909D   1200         CALL  DISCH
C39F9C   1210         JP    LOOP
FFC3A599 1220 IXSTR:  DEFB  255,195,165,153
99A5C3FF 1330         DEFB  153,165,195,255
FF81BDBD 1240         DEFB  255,129,189,189
BDBD81FF 1250         DEFB  189,189,129,255
```

```
FF818181  1260          DEFB  255,129,129,129
818181FF  1270          DEFB  129,129,129,255
010800    1280 RAM2V:   LD    BC,8
111004    1290          LD    DE,1040
2AE4D6    1300          LD    HL,(CHRADD)
CD5C00    1310          CALL  92
C9        1320          RET
010800    1330 CHRGRD   LD    BC,8
11D8D6    1340          LD    DE,GRID
2AE4D6    1350          LD    HL,(CHRADD)
EDB0      1360          LDIR
C9        1370          RET
210218    1380 DISCH:   LD    HL,6146
11D8D6    1390          LD    DE,GRID
3E08      1400          LD    A,8
F5        1410 DISCH1   PUSH  AF
1A        1420          LD    A,(DE)
CB7F      1430          BIT   7,A
CDDB9D    1440          CALL  BIT
CB77      1450          BIT   6,A
CDDB9D    1460          CALL  BIT
CB6F      1470          BIT   5,A
CDDB9D    1480          CALL  BIT
CB67      1490          BIT   4,A
CDDB9D    1500          CALL  BIT
CB5F      1510          BIT   3,A
CDDB9D    1520          CALL  BIT
CB57      1530          BIT   2,A
CDDB9D    1540          CALL  BIT
CB4F      1550          BIT   1,A
CDDB9D    1560          CALL  BIT
CB47      1570          BIT   0,A
CDDB9D    1580          CALL  BIT
13        1590          INC   DE
011800    1600          LD    BC,24
09        1610          ADD   HL,BC
F1        1620          POP   AF
3D        1630          DEC   A
C2989D    1640          JP    NZ,DISCH1
CD779D    1650          CALL  RAM2V
3E7F      1660          LD    A,127
2AE6D6    1670          LD    HL,(CURSOR)
CD4D00    1680          CALL  77
CDE39E    1690          CALL  DELAY
C9        1700          RET
F5        1710 BIT:     PUSH  AF
```

```
CAE49D  1720          JP    Z,BIT1
3E80    1730          LD    A,128
C3E69D  1740          JP    BIT2
3E81    1750 BIT1:    LD    A,129
CD4D00  1760 BIT2:    CALL  77
F1      1770          POP   AF
23      1780          INC   HL
C9      1790          RET
00      1800 SWITCH   NOP
2AE0D6  1810          LD    HL,(BYTE)
7E      1820          LD    A,(HL)
47      1830          LD    B,A
3AE2D6  1840          LD    A,(CHBIT)
4F      1850          LD    C,A
3E01    1860          LD    A,1
CB79    1870 SWIT1:   BIT   7,C
C2059E  1880          JP    NZ,SWIT2
CB00    1890          RLC   B
CB01    1900          RLC   C
3C      1910          INC   A
C3F89D  1920          JP    SWIT1
CB78    1930 SWIT2:   BIT   7,B
CA0F9E  1940          JP    Z,SWIT3
CBB8    1950          RES   7,B
C3119E  1960          JP    SWIT4
CBFF    1970 SWIT3:   SET   7,A
3D      1980 SWIT4:   DEC   A
CA1A9E  1990          JP    Z,SWIT5
CB08    2000          RRC   B
C3119E  2010          JP    SWIT4
78      2020 SWIT5:   LD    A,B
77      2030          LD    (HL),A
CDF29E  2040          CALL  GRDCHR
CD909D  2050          CALL  DISCH
CDE39E  2060 WAIT:    CALL  DELAY
3E08    2070          LD    A,8
CD4101  2080          CALL  321
CB47    2090          BIT   0,A
CA229E  2100          JP    Z,WAIT
C39F9C  2110          JP    LOOP
00      2120 NEXT:    NOP
3AE3D6  2130          LD    A,(CURCHR)
D67E    2140          SUB   126
CA9F9C  2150          JP    Z,LOOP
C67F    2160          ADD   A,127
32E3D6  2170          LD    (CURCHR),A
```

```
2AE4D6    2180          LD    HL,(CHRADD)
010800    2190          LD    BC,8
09        2200          ADD   HL,BC
22E4D6    2210          LD    (CHRADD),HL
CD849D    2220          CALL  CHRGRD
CD909D    2230          CALL  DISCH
210C18    2240          LD    HL,6156
CD4A00    2250          CALL  74
D639      2260          SUB   57
CA639E    2270          JP    Z,NEXT1
C63A      2280          ADD   A,58
CD4D00    2290          CALL  77
C39F9C    2300          JP    LOOP
3E30      2310 NEXT1:   LD    A,48
CD4D00    2320          CALL  77
210B18    2330          LD    HL,6155
CD4A00    2340          CALL  74
D639      2350          SUB   57
CA7B9E    2360          JP    Z,NEXT2
C63A      2370          ADD   A,58
CD4D00    2380          CALL  77
C39F9C    2390          JP    LOOP
3E30      2400 NEXT2:   LD    A,48
CD4D00    2410          CALL  77
210A18    2420          LD    HL,6154
3E31      2430          LD    A,49
CD4D00    2440          CALL  77
C39F9C    2450          JP    LOOP
00        2460 PREV:    NOP
3AE3D6    2470          LD    A,(CURCHR)
3D        2480          DEC   A
CA9F9C    2490          JP    Z,LOOP
32E3D6    2500          LD    (CURCHR),A
2AE4D6    2510          LD    HL,(CHRADD)
010800    2520          LD    BC,8
A7        2530          AND   A
ED42      2540          SBC   HL,BC
22E4D6    2550          LD    (CHRADD),HL
CD849D    2560          CALL  CHRGRD
CD909D    2570          CALL  DISCH
210C18    2580          LD    HL,6156
CD4A00    2590          CALL  74
D630      2600          SUB   48
CABB9E    2610          JP    Z,PREV1
C62F      2620          ADD   A,47
CD4D00    2630          CALL  77
```

```
C39F9C    2640          JP    LOOP
3E39      2650 PREV1:   LD    A,57
CD4D00    2660          CALL  77
210B18    2670          LD    HL,6155
CD4A00    2680          CALL  74
D630      2690          SUB   48
CAD39E    2700          JP    Z,PREV2
C62F      2710          ADD   A,47
CD4D00    2720          CALL  77
C39F9C    2730          JP    LOOP
3E39      2740 PREV2:   LD    A,57
CD4D00    2750          CALL  77
210A18    2760          LD    HL,6154
3E30      2770          LD    A,48
CD4D00    2780          CALL  77
C39F9C    2790          JP    LOOP
3E32      2800 DELAY:   LD    A,50
F5        2810 DEL:     PUSH  AF
3EFF      2820          LD    A,255
3D        2830 DEL1:    DEC   A
C2E89E    2840          JP    NZ,DEL1
F1        2850          POP   AF
3D        2860          DEC   A
C2E59E    2870          JP    NZ,DEL
C9        2880          RET
010800    2890 GRDCHR   LD    BC,8
ED5BE4D6  2900          LD    DE,(CHRADD)
21D8D6    2910          LD    HL,GRID
EDB0      2920          LDIR
C9        2930          RET
```

Uma vez digitado e conferido, este programa pode ser guardado numa fita por meio da instrução:

```
BSAVE "CAS:DESENHO",40000,40800
```

Ele será testado, em seguida, por meio das duas linhas BASIC seguintes:

```
1000 DEF USR=40000
1005 A=USR (1)
```

Uma rede de 8 X 8 deve ser afixada no vídeo. Em cima e à esquerda aparece a letra X. Esta letra pode ser deslocada na rede por meio das teclas de deslocamento do cursor. Cada um dos 64 quadrados que compõem a rede pode ser sucessivamente "iluminado" ou "apagado" (ver a este respeito o apêndice consagrado ao sistema binário). Para modificar a situação de um quadrado, é suficiente colocar o cursor (isto é, a letra X) sobre ele e, em seguida, apertar a barra de espaço. Se este quadrado estava apagado, ele se ilumina, o inverso também é verdadeiro. À direita da rede aparece o caractere a ser desenhado. O desenho toma forma à medida que o cursor é deslocado na tela.

Este programa pode ser utilizado para desenhar 126 caracteres. Um número é afixado no alto do vídeo e corresponde ao caractere que se encontra sobre a rede. Para passar para um outro caractere é suficiente apertar a tecla N. Esta tecla permite o aparecimento dos caracteres de número 1 a número 126, enquanto que a tecla P permite o seu aparecimento em sentido contrário. Naturalmente, a rede permanece intacta enquanto os caracteres correspondentes não tenham sido desenhados.

Logo que um certo número de caracteres tenham sido, deste modo, definidos e que se deseja guardá-los em fita, a tecla ESC deve ser pressionada. É suficiente em seguida, digitar:

```
BSAVE "CAS:CARACT",41000,42007
```

É aconselhável guardar duas cópias (em duas fitas diferentes) do novo conjunto de caracteres, a fim de evitar-se qualquer problema que possa resultar da alteração de uma fita magnética.

O procedimento completo para criar um novo jogo de caracteres é o seguinte:

1. Digitar:

```
CLEAR 200,35999
```

2. Digitar:

```
BLOAD "CAS:"
```

Estas instruções permitem carregar na memória central do computador o programa de desenho.

3. Neste estágio, é possível carregar um jogo de caracteres que se tenha começado a definir e que tenha sido armazenado em fita; para isto basta digitar:

```
BLOAD "CAS:"
```

4. Digitar em seguida as duas linhas a seguir:

```
1000 DEF USR=40000
1005 A=USR (1)
```

5. Desenhar os caracteres de sua escolha.

6. Apertar a tecla ESC.

7. Guardar em fita o novo jogo de caracteres digitando:

```
BSAVE "CAS:CARACT",36000,38007
```

Este jogo de caracteres pode agora ser utilizado em um programa, da seguinte maneira:

1. Teclar:

```
CLEAR 200,35999
```

2. Teclar:

```
SCREEN 1
```

3. Carregar o novo jogo de caracteres, digitando:

```
BLOAD "CAS:"
```

Os 126 caracteres encontram-se, assim, carregados na memória RAM. Entretanto, para que eles possam ser utilizados, uma cópia deve ser transferida para a memória VRAM. A transferência pode ser realizada por meio das seguintes instruções BASIC:

```
10 FOR A=0 TO 2007
20 VPOKE(INICIOCAR+A),PEEK(36000+A)
30 NEXT A
40 STOP
```

Este programa carrega o novo jogo de caracteres na memória VRAM, de modo que o primeiro caractere deste jogo tem por número 126 e que 126º corresponde ao número 251. O modo vídeo não deve ser modificado uma vez que o novo jogo de caracteres foi carregado na memória VRAM. Tal modificação arriscaria perder estes novos caracteres em favor daqueles do padrão MSX.

As seguintes linhas permitem testar se os novos caracteres foram corretamente carregados na memória VRAM:

```
10 FOR A=126 TO 251
20 VPOKE(INICIOVIDEO)+A),A
30 NEXT A
40 STOP
```

O novo jogo de caracteres pode assim ser mais útil nos programas BASIC que nos programas escritos em linguagem de máquina.

Ainda que o programa proposto não permita, em princípio, definir um jogo de 126 caracteres, o procedimento descrito abaixo pode ser utilizado para criar até 252 novos caracteres:

1. **CLEAR 200,35999**

2. Carregar o programa de desenho por meio da instrução:

```
BLOAD "CAS:"
```

3. Entrar e executar as duas linhas seguintes:

```
1000 DEF USF=40000
1005 A=USR(1)
```

4. Desenhar 126 caracteres.

5. Apertar a tecla ESC.

6. Guardar estes caracteres em fita, digitando:

```
BSAVE "CAS:JOGO1",36000,38007
```

7. Digitar o comando RUN

8. Desenhar um novo jogo de 126 caracteres.

9. Teclar ESC.

10. Guardar estes caracteres em fita, digitando:

```
BSAVE "CAS:JOGO2",36000,38007
```

Para poder utilizar os 252 caracteres em um programa, o seguinte procedimento deve ser empregado:

1. **CLEAR 200,35999**

2. **BLOAD "CAS:JOGO1"**

3. Entrar e executar o seguinte programa:

```
10 FOR A=0 TO 2007
20 VPOKE(INICIOCAR+A+1008),PEEK(36000+A)
30 NEXT A
40 STOP
```

Os 126 caracteres do primeiro jogo serão, então, carregados na memória VRAM. Seus números estarão compreendidos entre 126 e 251.

4. **BLOAD "CAS:JOGO2"**

5. Trocar a linha 20 do programa abaixo e substituí-la por:

```
20 VPOKE(INICIOCAR+A),PEEK(A+36000)
```

6. Digitar o comando RUN. O jogo de caracteres n.º 2 é deste modo transferido para a memória VRAM, os caracteres correspondentes tendo um número compreendido entre 0 e 125.

Para questões relativas à clareza da afixação, é preferível, quando se escreve um programa que utiliza um grande número de caracteres redesenhados, conservar as formas originais enquanto a escrita do programa não esteja terminada.

DESENHO DOS SPRITES

O programa anterior pode igualmente ser utilizado para desenhar os sprites. O procedimento para criar 32 sprites é o seguinte:

1. **CLEAR 200,35999**

2. Carregar o programa de desenho, digitando a instrução;

   BLOAD "CAS:"

3. Entrar e executar as duas linhas seguintes:

```
1000 DEF USR=40000
1005 A=USR(1)
```

4. Desenhar as formas dos 32 sprites, fazendo-as corresponder aos números normalmente reservados aos caracteres 1 a 32.

5. Teclar ESC.

6. Guardar em fita as formas desenhadas, digitando:

   BSAVE "CAS:SP32",36000,38007

Estes sprites poderão ser, em seguida, utilizados em um programa da seguinte maneira:

1. Digitar:

   CLEAR 200,35999

2. Carregar os sprites na memória central, digitando a instrução:

   BLOAD "CAS:SP32"

3. Transferir estes dados na memória executando o seguinte programa BASIC:

```
10 FOR A=0 TO 255
20 VPOKE(SPRITEFORMA+A),PEEK(A+36000)
30 NEXT A
40 STOP
```

Em seguida, é suficiente definir os parâmetros de afixação (atributos) de alguns sprites e carregá-los na memória VRAM utilizando a instrução VPOKE para fazê-los aparecer no vídeo.

RESPOSTAS ÀS QUESTÕES

1. A parte fixa do número decimal 45621 é 178, e sua parte flutuante é 53.

2. O número decimal que tem 64 como parte fixa e 31 como parte flutuante, é 16415.

3. Se os bytes dos endereços 40000 e 40001 contêm respectivamente os valores 5d e 15d, o resgistrador HL conterá o valor 3845 após a execução da instrução LD HL,(40000).

4. Se o registrador HL contém o valor 35621 e a instrução LD(40000), HL for executada, o valor 37 é então carregado para o endereço 40000 e o valor 139 para o endereço 40001.

5. O equivalente decimal de E3h é 227d.

6. Se FBh é a parte fixa de um número e CBh sua parte flutuante, este número vale 64459d.

Outros titulos publicados pela Editora Manole :

- **BASIC Basico**
  David Munro

- **Voce Conhece os Computadores ?**
  Karen Billings

- **Entendendo os Computadores**
  Myles Walsh

- **MSX Jogos em Assembler**
  Eric Ravis

- **MSX Jogos de Acao**
  Pierre Monsaut

- **Tecnicas de Programacao de Jogos em Assembler** (Incluindo 15 superjogos!!!)
  Georges Fagot-Barraly

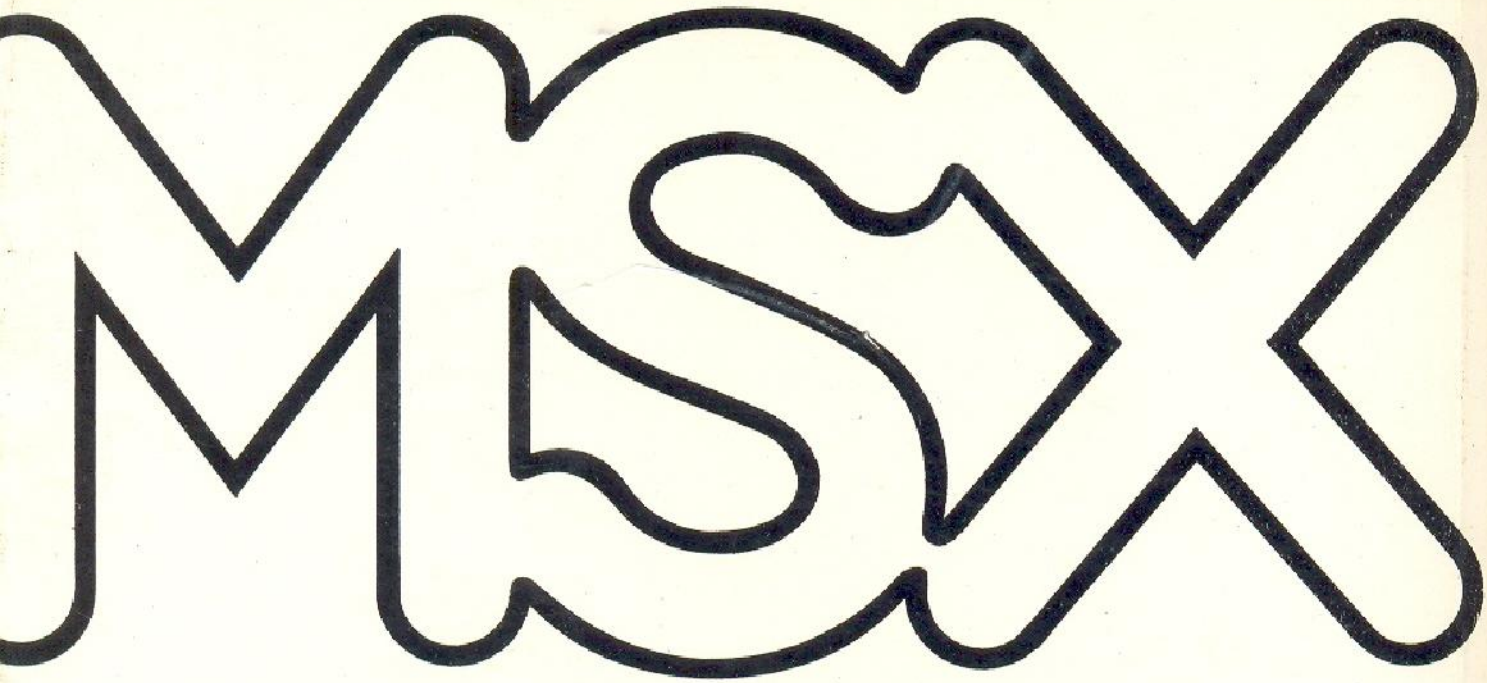
- **MSX Rotinas Graficas em Assembler**
  Steve Webb

- **Mais Jogos e Graficos para o MSX**
  Graham Carter

- **SPECTRUM Jogos de Acao**
  Pierre Monsaut

- **Jogos de Acao para o APPLE**
  Faca seus proprios jogos
  Zimmerman&Zimmerman

Caso nao encontrar os livros na sua livraria, procurar na editora que fica na Rua 13 de Maio , 1026 - Bela Vista - SP
(travessa da Av. Brig. Luis Antonio)
Fone: 287-0746



Os livros sobre BASIC são numerosos e a pessoa que tem um microcomputador encontra todas as informações necessárias para começar a programar. Porém as realizações dos programas requerem uma grande velocidade de execução que entra em choque com um problema maior: a lentidão do interpretador BASIC. Este livro permite ir mais longe abordando a programação em linguagem de máquina. A maneira de programar equivalente às instruções BASIC PRINT, GOTO, GOSUB, FOR/NEXT, etc., é estudada primeiro para que depois essas noções sejam aplicadas na realização dos jogos de ação.
Numerosos subprogramas poderão ser reutilizados pelo leitor nos seus próprios programas.