

OUTROS LIVROS NA ÁREA

Guias do Usuário

- | | | |
|----------|---|--|
| Carvalho | — | Basic para o TK 90X |
| Hurley | — | Programas para jovens programadores TK-82, 83, 85 e CP 200 |
| Hurley | — | TK 90X — Programas para jovens programadores |

0-07-450227-1



ASSEMBLER

Para o

TK 90X

TK 95X

JOSÉ EDUARDO M. DE CARVALHO

CARVALHO

ASSEMBLER

Para o

TK 90X



McGraw-Hill





**Assembler para
o TK 90X**

Assembler para o TK 90X

JOSÉ EDUARDO MALUF DE CARVALHO

McGraw-Hill
São Paulo
Rua Tabapuã, 1.105, Itaim-Bibi
CEP 04533
(011) 881-8604 e (011) 881-8528

*Rio de Janeiro • Lisboa • Porto • Bogotá • Buenos Aires •
Guatemala • Madrid • México • New York • Panamá • San
Juan • Santiago*

*Auckland • Hamburg • Kuala Lumpur • London •
Milan • Montreal • New Delhi • Paris • Singapore •
Sydney • Tokyo • Toronto*

Nenhuma parte desta publicação poderá ser reproduzida, guardada pelo sistema "retrieval" ou transmitida de qualquer modo ou por qualquer meio, seja este eletrônico, mecânico, de fotocópia, de gravação, ou outros, sem prévia autorização, por escrito, da Editora.

Coordenadora de Revisão: Daisy Pereira Daniel

Supervisor de Produção: Edson Sant'Anna

Capa: Layout: Cyro Giordano

Dados para Catalogação-na-Publicação (CIP) Internacional
(Câmara Brasileira do Livro, SP, Brasil)

C324a	Carvalho, José Eduardo Maluf de. Assembler para o TK 90X / José Eduardo Maluf de Carvalho. — São Paulo: McGraw-Hill, 1986.
	1. Assembler (Linguagem de programação para computadores) 2. TK 90X (Computador) — Programação I. Título.
86-0250	CDD-001.6424 -001.642

Índices para catálogo sistemático:

1. Assembler: Linguagem de programação : Computadores : Processamento de dados 001.6424
2. TK 90X : Computadores : Programação : Processamento de dados 001.642

Agradeço a todos os que me incentivaram e encorajaram a escrever este livro, em particular a Jorge dos Santos, e mais particularmente ainda a minha adorável esposa Vivian e a meus queridos filhos Felipe e Marina, que se contentaram apenas em me olhar, enquanto escrevia este volume.

José Eduardo Maluf de Carvalho, 32 anos, arquiteto, atuando em Planejamento Urbano, na Secretaria Municipal de Planejamento de São Paulo e em projetos de edificações e construção civil em seu escritório próprio, sentiu a necessidade de agilizar seu trabalho particular, nas tarefas mais repetitivas do arquiteto, desde uma consulta às leis de zoneamento, que regem o uso e aproveitamento do solo no Município de São Paulo, até as fases finais de detalhamento, quantificação e custo de um projeto de arquitetura.

Para isso, adquiriu, como um curioso, em março de 1982, um TK 82-C, e imediatamente conseguiu fazer deste pequeno micro uma poderosa ferramenta de trabalho.

Sentindo falta de uma melhor resolução gráfica, bem como de uma maior capacidade de armazenamento, partiu para um micro que foi um caso de "amor a primeira vista". Era o ZX Spectrum da Sinclair.

Desde setembro de 1983, o domínio sobre essa máquina foi num crescente cada vez maior, incluindo aí os periféricos ZX Microdrive, que nada mais é que um micro acionador de cartuchos (em vez de discos) e a Interface One (de compatibilização com o Microdrive, saída RC 232 C e rede local).

Trabalhando principalmente em linguagem Basic, com algumas rotinas em código de máquina, conseguiu elaborar um pacote de programas aplicativos para arquitetura, que vão desde a análise do zoneamento, otimizando a ocupação do terreno, definição do programa do projeto de arquitetura, elaboração da planta baixa, vegetação, detalhes construtivos, perspectivas, culminando com o orçamento e o cronograma da obra. Ou seja, não se sai do micro sem ter o projeto de arquitetura completo.

Hoje, após o lançamento do TK 90X e do TK 95, da Microdigital, compatíveis com o Spectrum, e até com algumas inovações e levando em consideração o ineditismo do seu trabalho, o autor fundou a empresa ARQUITRON INFORMÁTICA LTDA., de comércio e prestação de serviços em informática.

Mas, nem por isso abandona os deliciosos momentos de lazer que tem com este programa parceiro de *video games*.

Atualmente gerencia o Departamento de Desenvolvimento de Software da Empresa Tropic Informática Ltda., onde desenvolve programas para as linhas Spectrum e MSX.

Introdução	XI
Capítulo 1 — O que é o TK 90X	1
Capítulo 2 — Variáveis que controlam o sistema	23
Capítulo 3 — Algo sobre os comandos IN e OUT de Basic	43
Capítulo 4 — As cores pelo teclado	46
Capítulo 5 — O modo gráfico	49
Capítulo 6 — O microprocessador Z80A	52
Capítulo 7 — Estrutura de um programa em código de máquina	63
Capítulo 8 — A matemática na programação em código de máquina	69
Capítulo 9 — Operações lógicas	82
Capítulo 10 — As instruções em código de máquina do Z80	86
Capítulo 11 — As rotinas da ROM de 16K	129
Capítulo 12 — Como utilizar rotinas do programa monitor	135
Capítulo 13 — Programas e rotinas	151
Apêndice A — Conversões de valores decimais, binários e hexadecimais	215
Apêndice B — Códigos de operação do Z80 ordenados por mnemônicos	222

Meus parabéns!

Se você realmente deseja aprender a linguagem de máquina deste microprocessador de 8 bits, muito poderoso, denominado Z80A, você tem muita coragem, e só lhe dando os parabéns!

Você provavelmente já aprendeu e já domina todo o potencial da linguagem Basic deste microcomputador (já deve inclusive ter lido meu livro de Basic e seus segredos), mas quer ir um pouco mais adiante e dominar agora diretamente a máquina, através de seus códigos de processamento.

Pois vá em frente! Não existe nenhuma linguagem de alto nível que explore todo o potencial de uma máquina. Somente através de sua própria linguagem é que exploramos todo o seu potencial.

Esqueça todos os princípios da estrutura de um programa em linguagem Basic. Lá, você trabalha com linhas de programa, comando a comando, que serão posteriormente executados, um a um, seqüencialmente.

Aqui não — a estrutura da linguagem de máquina é completamente diferente. Evidente que a seqüência lógica de execução permanece, mas aqui nós manipulamos bytes diretamente armazenados em endereços especificados da memória. Isto é uma relação biunívoca: a cada endereço corresponde um e somente um byte de 8 bits.

Você vai precisar de muita paciência, precisará ser “adivinho” em algumas ocasiões, e principalmente deverá tomar muito cuidado com os valores numéricos que vai manipular. E exercitar muito.

Eu, particularmente, não conheço ninguém que faça um programa inteiro em código de máquina.

Você já imaginou, por exemplo, o programa inglês VU 3D, da Psion Software, que é um mini CAD (*computer aided design* — projetos assistidos por computador) para elaboração de perspectivas, que possui uma parte de códigos, num total de 32183 endereços, ser feito todo manualmente, e de cabeça? É praticamente impossível; isso é coisa de loucos.

O mais normal é a elaboração de programas em partes, em linguagem de alto nível, residente, que são “compiladas” uma a uma, ou seja, através de um programa-ferramenta, chamado “compilador”, são traduzidos automaticamente para a linguagem de máquina.

Você quer, por exemplo, fazer a tela rodar em diversas direções (scroll variado). Faça uma rotina de execução em linguagem de máquina para cada direção. Cada uma delas será posteriormente acessada através das senhas ou condições que você impuser, e poderá ser utilizada em qualquer outro programa.

Mas, vamos lá, então, à programação em código de máquina.

Você já sabe que a única coisa que seu computador entende são sinais, respectivamente, “com voltagem” e “sem voltagem”, ou ainda “voltagem alta” e “voltagem baixa”. No caso do Z80, a voltagem alta equivale a +5 volts, e a voltagem baixa equivale a 0 volts, que convencionou-se padronizar através dos dígitos 0, para voltagem baixa, e 1, para voltagem alta, dando origem ao sistema binário, diretamente manipulado pelo microcomputador.

Esses algarismos são mais conhecidos por bits, e o agrupamento de 8 desses bits dá origem a um byte, que é a menor unidade de armazenamento de memória de um computador, cujos valores vão de 0 a 255, no sistema decimal, totalizando 256 valores, ou, no sistema hexadecimal, de valores que iniciam em 00 e terminam em FF.

Na linguagem Basic, chegamos a manipular bytes de memória, mas, veja a grande dificuldade da linguagem de máquina — aqui, manipularemos até apenas um bit de um determinado byte, para ordenar algo ao microcomputador. E, por esse caminho, você percebe que, se errar ou esquecer apenas 1 bit de 1 byte, põe a perder todo o seu trabalho de elaboração de uma rotina em linguagem de máquina, provocando um “crash” no sistema, ou seja, o não retorno do micro para a linguagem Basic. Nestes casos, não se preocupe — basta desligar e ligar novamente o computador.

Não tenha medo de digitar nada, que você nunca, através do teclado, vai provocar algum dano no computador. O máximo que pode acontecer é a situação descrita acima.

Mas, como então programar em linguagem de máquina?

Vamos lá — primeiramente, você deve ter em mente, muito bem definida, a concepção geral do seu programa, sobre como ele vai funcionar, como vai responder a determinadas condições etc.

Conheço casos de programadores fanáticos em código de máquina, que chegam a pintar, num papel apropriado, a idéia que têm sobre o desenho desejado, incluindo pequenos caracteres gráficos que devam se movimentar, para em seguida, através de um papel transparente onde constam todas as locações da tela, passarem a escrever a rotina que irá executar aquele desenho ou figura. O resultado é, na verdade, um desenho magnífico, mas, na minha modesta opinião, o processo é muito sofisticado.

Bem, de posse da concepção do programa, evidentemente adequado à lógica do computador, você deve em seguida elaborar um fluxograma dele, ou seja, as suas etapas de execução.

Definidas estas etapas, com uma listagem das mais de 700 instruções em código de máquina, você passa então a escrever a sua rotina, através das “mnemônicas”, ou símbolos (nomes) das instruções, sempre levando em consideração os endereços, ou locações iniciais ou intermediárias da memória, que também podem ser rotulados (por você é claro) para não se perder em números mais tarde.

Neste fase, você tem duas opções para continuar o trabalho, que são:

1 — usar um programa-ferramenta denominado ASSEMBLER, ou montador, que permite que você digite a sua listagem ASSEMBLY, ou montada, a partir das mnemônicas, endereço por endereço, byte a byte, para que ele converta automaticamente estes seus símbolos em códigos, números que serão entendidos pelo computador para serem processados posteriormente.

Este é o método mais simples, visto que a primeira etapa de elaboração de um programa, é a sua listagem por instruções, mas, na minha opinião, não é o método mais indicado, pois está sujeito a muitos erros. E, como já vimos, basta uma unidade de um byte estar com o valor errado. . .

2 — esta opção é praticamente continuação da anterior. De posse da listagem das mnemônicas do programa, você deve converter estes símbolos para os seus respectivos códigos no sistema hexadecimal. Apesar do trabalho, isto já é uma vantagem, pois é uma conferência do trabalho.

Convertida a listagem para códigos hexadecimais, a última etapa desta opção é a conversão destes códigos para números do sistema decimal, que serão diretamente armazenados nos endereços da memória RAM do seu mi-

cro que você determinar. Nesta etapa, nada lhe impede que use um "Editor Assembler", outro programa-ferramenta que permite que você digite os códigos em hexadecimal, para que ele os converta automaticamente no sistema decimal.

Como você deve ter notado, o trabalho de programação em código de máquina é muito exaustivo e muito sujeito a erros, mas também muito compensador.

A prática desta linguagem é um fator muito importante para seu perfeito domínio, além do pleno conhecimento de todo o conjunto das instruções do microprocessador Z80A.

Praticando, e muito, você vai guardar os códigos, tanto em hexadecimal quanto em decimal, das instruções mais utilizadas na programação desta linguagem.

Bem, você já elaborou então a sua rotina em linguagem de máquina. Para um leigo no assunto, todos aqueles números não quer dizer nada. Mas, para quem entende, significa o resultado de horas de pesquisa e trabalho árduo!

Cuidado — não rode esse programa ainda! Você pode pôr tudo a perder se ele estiver errado! Como prevenção, grave essa rotina na fita cassete ou, se você tiver, no microdrive (que aparelho fantástico!).

Agora sim, de posse de uma cópia dessa rotina gravada, você pode processá-la, e saber, caso não aconteça o que você esperava, onde está o erro conferindo naquele papel rascunho onde você escreveu a listagem.

Carregue aquela rotina que foi gravada e, através de "POKE endereço, byte", corrija-a para gravá-la novamente, agora na segunda versão. Teste-a, e continue com esse processo até atingir o seu objetivo. Assim que atingi-lo, dê-se parabéns! Agora você também programa em linguagem de máquina.

Este é o objetivo deste livro. Ensiná-lo a programar em linguagem de máquina. Alguns autores falam superficialmente e profundamente (para mim, quanto mais se pratica, mais se aprofunda no assunto), de uma maneira muito didática, considerando que você não entende nada desse assunto, e que obrigatoriamente deve partir dos princípios mais rudimentares do processo (foi assim que eu comecei — sem curso algum).

Por isso começo o livro com uma recordação da linguagem Basic, cujos conceitos também serão utilizados em linguagem de máquina.

Pena que a teoria desta linguagem seja muito extensa; não tenha pressa, assimile bem os conceitos antes de passar adiante.

Nos capítulos finais do livro coloquei diversas rotinas em código de máquina, que, além de úteis e divertidas, vão mostrar-lhe como o computador trabalha, além de fazê-lo entender essa linguagem. Também inseri nesses capítulos alguns programas híbridos, ou seja, que são escritos em Basic, mas que possuem algum processamento diretamente em código de máquina.

Observe a execução dessas rotinas e desses programas, onde a velocidade de processamento do micro aumentou, chegando a ser, em certos casos, cerca de 100 vezes mais rápida que a linguagem Basic.

Essa é uma das grandes vantagens da linguagem de máquina — a velocidade de processamento muito alta, já que não existe nenhum interpretador no meio do caminho.

A outra grande vantagem é a economia de memória, por motivos óbvios (números são armazenados mais facilmente que a linguagem Basic; veja seu manual, não existem indicadores de início e final de linha, indicadores de final de comando etc.).

Pois bem, chega de escrever, e mãos à obra!

Divirta-se!

O AUTOR

Nota: Todos os programas, bem como as rotinas em linguagem de máquina listados neste livro foram testadas num TK 90X de 48K de RAM, n.º 7544/CQF/264/11015/9, e rodaram sem problemas. No entanto, mais uma vez ficou constatado que este micro não é 100% compatível com seu irmão inglês, o ZX Spectrum da Sinclair.

Portanto, muito cuidado com os programas que você adquirir, pois alguns deles não rodam no TK 90X.

Não se preocupe com o comando "BEEP" que você encontrar nas listagens Basic. É um comando do ZX Spectrum, equivalente ao "SOUND" do TK 90X. (É que eu prefiro o Spectrum!)

Na última Feira da Informática, vi e testei o novo TK 95, compatível com o TK 90.

Pelo contato que tive durante poucos minutos com esta máquina, devo parabenizar seu fabricante, pois parece (parece...) que é uma grande máquina, com um ótimo teclado.

O que é o TK 90X

Podemos descrever qualquer sistema de computação sob três aspectos diferentes:

O primeiro aspecto é relativo à descrição sucinta do sistema de um modo geral, a partir de sua CPU e seus periféricos.

O segundo aspecto refere-se ao hardware (partes físicas) do sistema propriamente dito.

E o terceiro aspecto é sobre o desempenho lógico do micro.

O SISTEMA

O microcomputador TK 90X é uma pequena caixa preta, de plástico, medindo 236 mm de largura, por 146 mm de profundidade e 44 mm de altura. Na superfície horizontal superior estão dispostas as quarenta teclas de um composto de silicone, com as legendas impressas pelo processo chamado *hot-stamping*, ou seja, *silk screen* à quente, para maior durabilidade, que compõe o seu teclado.

Na traseira da caixa estão dispostos, da esquerda para a direita, o plug que conecta com a entrada da TV (plug da antena externa), o soquete de entrada que conecta com a saída do gravador cassete, o soquete de saída que conecta com a entrada do gravador cassete, o plug de conexão para joysticks ti-

po Atari, o conector de expansão da máquina, onde se conectam diversos periféricos de entrada/saída, tais como uma impressora (ZX Printer ou Alpha-com 32) uma interface RS 232, a interface de compatibilização com o microdrive etc. e finalmente o plug de voltagem.

A placa de circuitos, incluindo o microprocessador Z80 e outros componentes eletrônicos, encontra-se dentro dessa caixa preta, tomando toda a sua dimensão, e é separada do teclado por dois cabos de ligação.

O HARDWARE (OS COMPONENTES FÍSICOS)

A placa de circuitos do TK 90X pode facilmente ser acessada, retirando-se os quatro parafusos inferiores que prendem a tampa, onde está instalado o teclado, à parte inferior da caixa onde se encontra a placa de circuitos. Se o seu micro estiver na garantia não convém abri-lo, pois irá destruir o lacre da garantia, anulando-a. Se não, e se a curiosidade for muito grande, retire os parafusos, segurando as duas partes da caixa, para que não se separem, e com o micro em posição de funcionamento levante o teclado vagarosamente, procurando não desconectar os seus dois cabos de ligação com a placa, pois são muito frágeis e suas pontas se desgastam muito facilmente. Estes cabos podem ser desconectados puxando-os suavemente dos seus soquetes, mas procure não fazê-lo: use duas hastes (canetas) de igual comprimento para sustentar a tampa, tal qual o capô de um carro.

Os componentes principais da placa de circuito estão desenhados na Figura 1.

Cada um desses componentes principais será descrito a seguir:

— O MICROPROCESSADOR Z80

Este chip, muito conhecido entre os aficionados de informática, visto que é o coração de muitos sistemas que existem por aí, é o mais importante do TK 90X, e o mais sofisticado (também o mais complicado, em termos de código de máquina) dos microprocessadores de 8 bits. É, também, o coração dos micros linha MSX (Expert e Hotbit).

Como microprocessador, tem capacidade de trabalhar como um "computador", de uma maneira genérica, capaz de executar instruções armazenadas (programa). Os programas para o TK 90X, quaisquer que sejam eles, serão sempre convertidos internamente em instruções em código de máquina do Z80.

No TK 90X, o *clock* (relógio — medidor de tempo interno de execução de instruções) é de 3,75 MHz, e, a esta velocidade, ele é capaz de processar cerca de 900.000 das mais simples instruções em código de máquina. É interessante notar que — a partir do momento que se liga a fonte de energia do mi-

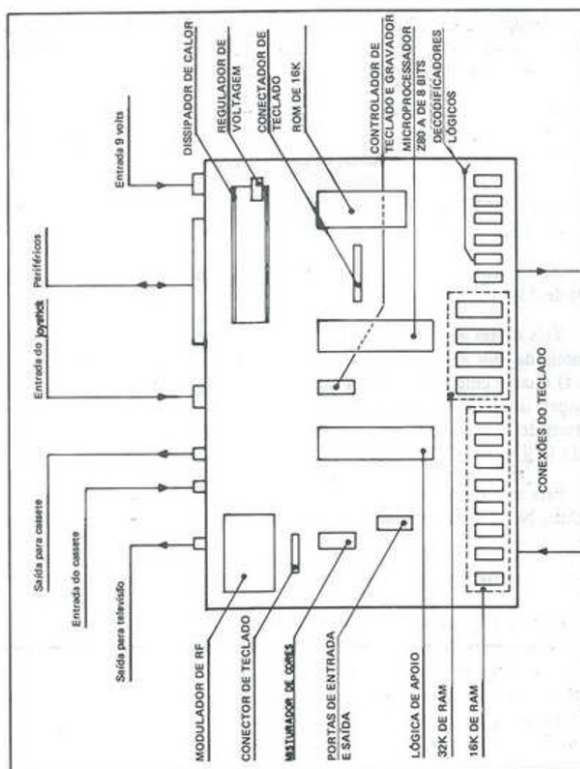


Figura 1 — Placa de circuitos eletrônicos e componentes principais do TK 90X.

cro, o seu microprocessador imediatamente começa a funcionar, apesar de não mostrar nenhum resultado. Ele simplesmente fica aguardando uma ordem. E são exatamente estas ordens que estudaremos neste livro.

— A ROM DE 16 KBYTES (ROM = READ ONLY MEMORY — OU MEMÓRIA APENAS DE LEITURA)

O programa em código de máquina que é normalmente executado pelo microprocessador Z80 está contido num único chip de ROM, que manipula 128 kbits ou 16 kbytes de informação.

Nestes 16 k de "programa monitor" do TK 90X, os aproximadamente primeiros 7 k estão dedicados ao sistema operacional, os 8 k seguintes, ao interpretador Basic e o 1 k restante, ao gerador de caracteres.

— A RAM (RANDOM ACCESS MEMORY — MEMÓRIA DE ACESSO RÁNDOMICO)

Na versão standard de 16 k do TK 90X existem 8 chips de memória RAM de 2 k cada ou 16 kbits.

Três destes 8 chips formam o "arquivo da imagem" ou mapeamento da memória da tela, e podem normalmente ser usados somente com este propósito. O quarto chip é dedicado à manipulação dos bytes dos atributos (cores de papel, tinta, flash, bright etc.) das 768 posições de caracteres da tela, e as variáveis do sistema. Um pouco mais de 8 k está disponível ao usuário na versão de 16 k.

Para a RAM total de 48 k, são adicionados mais 4 chips de 8 k cada, ou 64 kbits. Nessa memória estão disponíveis para o usuário um pouco mais de 40 k.

— A LA (LÓGICA DE APOIO)

Este chip pode ser considerado como sendo um "grande chip feito de diversos pequenos chips". É um dos chips conhecidos como "customizado", ou seja, só serve para esse computador. Ele atua como um "centro de comunicação", verificando se tudo o que o microprocessador requer ou ordena está sendo efetuado; ele também "lê" a memória para ver em que consiste a imagem da TV, e envia os sinais apropriados para a interface da TV.

— MISTURADOR DE CORES

Este chip recebe as informações sobre cores da LA e converte-as para os sinais apropriados a serem enviados ao modulador VHF. O sinal produzido pelo modulador é ajustado para o canal 3.

— Em adição a estes componentes principais, existem ainda a interface que produz som no alto-falante da TV; a interface para o gravador cassete; o dissipador de calor; o regulador de voltagem, que converte a tensão de entrada em 5 volts absolutos, sem oscilação; alguns decodificadores de endereços e outros componentes menores.

A LÓGICA DO SISTEMA

Neste aspecto, todas as ligações entre os vários componentes internos do microcomputador são consideradas. Essas ligações possuem uma existência real — são "caminhos" impressos na placa de circuitos, ou eventualmente fios de ligação.

O microprocessador Z80 pode gerar um endereçamento individual de até 65536 endereços diferentes de memória ($65536/1024 = 64$ kb) de cada vez. No TK 90X, versão 16 k, apenas os endereços 0 a 32767 (32 kb, sendo 16 k de ROM e 16 k de RAM) estão disponíveis. Na versão de 48 K todos os endereços são utilizados.

No TK 90X, os endereços são produzidos na forma de 16 sinais binários. O endereço 0 é, portanto, 00000000 00000000, e o endereço 65535 é 11111111 11111111. Isto porque o computador trabalha com o sistema numérico binário, com os algarismos 0 e 1.

Vamos ver:

No nosso sistema decimal, temos que:

$$\begin{array}{r}
 1985 \\
 \begin{array}{l}
 1 \times 10^3 = 1 \times 1000 = 1000 \\
 9 \times 10^2 = 9 \times 100 = 900 \\
 8 \times 10^1 = 8 \times 10 = 80 \\
 5 \times 10^0 = 5 \times 1 = 5
 \end{array} \\
 \hline
 1985
 \end{array}$$

ou

$$\begin{array}{r}
 4096 \\
 \begin{array}{l}
 4 \times 10^3 = 4 \times 1000 = 4000 \\
 0 \times 10^2 = 0 \times 100 = 0 \\
 9 \times 10^1 = 9 \times 10 = 90 \\
 6 \times 10^0 = 6 \times 1 = 6
 \end{array} \\
 \hline
 4096
 \end{array}$$

Analogamente, no sistema binário, que utiliza somente os algarismos 0 e 1, ou bits, temos para 1 byte (conjunto de 8 bits)

1111 1111	
↓	$1 \times 2^0 = 1 \times 1 = 1$
↓	$1 \times 2^1 = 1 \times 2 = 2$
↓	$1 \times 2^2 = 1 \times 4 = 4$
↓	$1 \times 2^3 = 1 \times 8 = 8$
↓	$1 \times 2^4 = 1 \times 16 = 16$
↓	$1 \times 2^5 = 1 \times 32 = 32$
↓	$1 \times 2^6 = 1 \times 64 = 64$
↓	$1 \times 2^7 = 1 \times 128 = 128$
	<hr/> 255

Ou seja, 256 valores diferentes para um byte, que começam em 0 e terminam em 255. Portanto, $2^8 = 256$

$$256 \times 256 = 2^8 \times 2^8 = 2^{16} = 65536 \text{ ou } 11111111 \ 11111111$$

Da mesma forma, no sistema hexadecimal, que utiliza os algarismos 0 a 9 para valores decimais de 0 a 9, e letras de A até F para valores de 10 a 15, temos:

HEXADECIMAL	DECIMAL
0	0
1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8
9	9
A	10
B	11
C	12
D	13
E	14
F	15

Portanto, para o valor decimal 255, temos em hexa FF e em binário 11111111

$$\begin{array}{r} \text{FF} \\ \downarrow \\ 15 \times 16^0 = 15 + \\ 15 \times 16^1 = 240 + \\ \hline 255 \end{array}$$

O programa a seguir faz as conversões automaticamente entre as bases numéricas decimal, binária e hexadecimal.

```

1 REM *****
2 REM * Programa de conversão *
3 REM * de bases numéricas *
4 REM * *****
10 CLS : PRINT "Este programa
faz a conversão de qualquer base
para qualquer outra base <=16"
15 INPUT "Digite a base anteri
or "; X$
20 LET A=0
25 IF VAL X$ > 16 OR X$="" THEN
GO TO 10
30 GO SUB 185
35 LET B=N
40 IF N < 2 OR N > 16 THEN GO TO 1
0
45 INPUT "Qual é o número "; X$
50 IF X$="" THEN GO TO 45
55 GO SUB 205
60 IF A=1 THEN PRINT "Erro ":
LET A=0: GO TO 45
65 LET N1=N
70 PRINT X$;" na base 10 é ";N
1
75 IF N1 < 1000000 THEN GO TO 85
80 PRINT "O número na base 10
é >= 1000000 podendo por isto oco
rrerem erros"
85 INPUT "Qual é a nova base "
;X$
90 IF X$="" THEN GO TO 85
95 GO SUB 185
100 LET B1=N: IF N < 2 OR N > 16 TH
EN GO TO 85
105 LET B$=""
110 LET V=INT (N1/B1)
115 LET R=N1-V*B1
120 IF R > 0 THEN GO TO 140
125 LET B$=B$+CHR$ (R+48)
130 LET N1=V: IF V=0 THEN GO TO
145

```



```

135 GO TO 110
140 LET r=r+55: LET b$=b$+CHR$(r): LET n1=v: IF v<>0 THEN GO TO 110
145 PRINT "O numero na base "; b$
150 FOR j=LEN b$ TO 1 STEP -1
155 PRINT b$(j);: NEXT j
160 PRINT
165 INPUT "Mais numeros (s/n) "; x$
170 IF x$="s" THEN GO TO 10
175 IF x$="n" THEN NEW
180 GO TO 165
185 LET n=0
190 FOR j=1 TO LEN (x$): LET d=CODE x$(j)
195 LET n=n*10+d-48: NEXT j
200 RETURN
205 LET n=0
210 FOR j=1 TO LEN x$: LET d=CODE x$(j)
215 IF d>47 AND d<58 THEN LET d=d-48: GO TO 230
220 IF d>64 AND d<71 THEN LET d=d-65: GO TO 230
225 LET e=1: RETURN
230 IF d>=b THEN LET e=1: RETURN
235 LET n=n*b+d
240 NEXT j
245 RETURN

```

Os endereços são gerados pelo microprocessador Z80 e transportados pelo computador através das Vias de Endereçamento (Address Bus). São no total 16 vias, nas quais um endereço pode ser especificado considerando-se uma via possuindo uma "alta" voltagem (5 volts) ou uma "baixa" voltagem (0 volts), ou ainda, respectivamente, 1 para alta voltagem e 0 para baixa voltagem.

Enquanto as vias de endereço possuem 16 contatos, as vias de dados (Data Bus) do TK 90X possuem somente 8 contatos, pelo fato do seu microprocessador ser de 8 bits. Portanto, qualquer dado que estiver sendo manipulado pelo sistema, seja ele uma instrução em código de máquina, um valor qualquer, ou uma instrução em Basic, deve ser considerado como sendo um número decimal na faixa de 0 a 255 (valor máximo de 1 byte), ou na faixa binária de 00000000 até 11111111.

A figura a seguir mostra esquematicamente como as vias de endereçamento e as vias de dados são interligadas aos outros componentes do TK 90X.

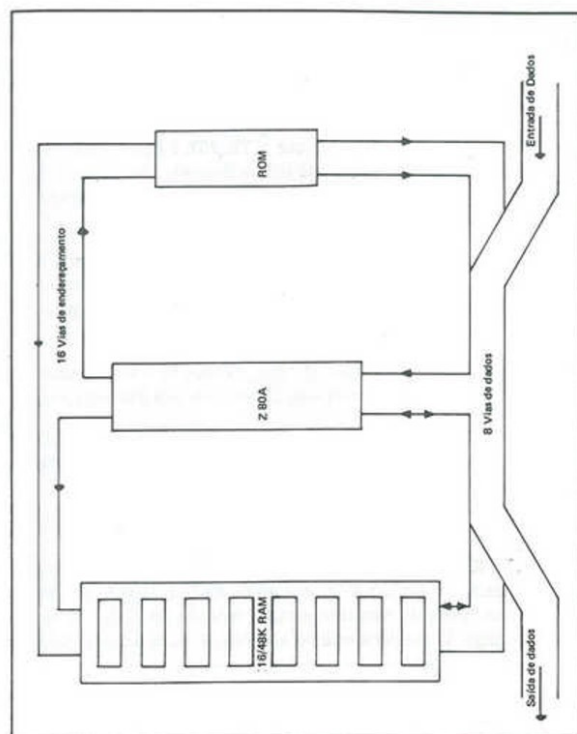


Figura 2 – Vias de endereçamento e de dados do TK 90X

O TK 90X é fornecido pela Microdigital com um programa monitor de 16 K, iniciando no endereço 0 e terminando no endereço 16383, que proporciona ao usuário o emprego de um sistema operacional bem como de um interpretador Basic. Também é possível "abandonar" este programa monitor e permitir que o microprocessador Z80 execute um programa em código de máquina escrito pelo usuário. Em modo normal de operação, o sistema operacional do TK 90X não requer nenhuma ação por parte do usuário, e todas as ações desse programa monitor são "transparentes" para ele.

Entretanto, isso acontece sempre que o TK 90X é ligado, mas não em operação; o seu interpretador Basic está constantemente "lendo" o teclado, através de uma rotina do programa monitor, aguardando uma declaração em Basic do usuário, seja em modo imediato ou em modo programado.

Note que de nenhuma maneira o microprocessador Z80 executará um programa em Basic — mas, sim o programa monitor, que é uma rotina em código de máquina do Z80. A única exceção a este procedimento é quando o próprio usuário insere no micro uma rotina em código de máquina.

O mapeamento da memória do TK 90X, versões 16 e 48 K é mostrado esquematicamente na Figura 3 e cada uma dessas áreas será discutida a seguir.

Nota: A partir daqui, todos os endereços que contiverem apenas algarismos serão da base decimal; se contiverem após os algarismos a letra "h" serão da base hexadecimal, e se contiverem a letra "b" serão da base binária.

— ÁREA DE ROM

Os 16 K de ROM contêm o sistema operacional, o interpretador Basic e o gerador de caracteres, ocupando os endereços de 0 a 16383, ou 0000 a 3FFFh. Como em qualquer microcomputador baseado no Z80, o início do programa em código de máquina está no endereço 0. Mais adiante, veremos esta área com mais detalhes.

— ÁREA DE MAPEAMENTO DA MEMÓRIA DA TELA — ARQUIVO DA IMAGEM

Os 6 K de memória, do endereço 16384 até 22527, ou 4000 a 57FFh, formam a área de alta resolução gráfica da tela. É importante notar que essa área está fixada nessa dimensão pelo hardware do TK 90X e não pode ser alterada por intermédio do software.

Existe uma relação de um para um entre todos os bits dessa área de memória e os pixels (pixels = *picture elements* — menor ponto de impressão) que aparecem na tela, e o cálculo seguinte mostra que o número de bits em 6 K de memória é igual ao número de pixels que podem ser mostrados na tela.

			MEMAVLB
		Caracteres gráficos	UDG
		3E h	RAMTOP
		?	
		Pilha de GOSUB	
		Pilha da máquina	SP
		Espaço livre	ADSPFREE
		Pilha de cálculo	STKEND
		Espaço temporário de trabalho	
		NL	
		Dados de INPUT	WORKPT
		80 h	
		NL	
		Área de edição de linhas	INADD
		80 h	
			VARADD
		Área de programas Basic	
		80 h	PROGBAS
		Informação de canais	CHCADD
23.734	5CB6h	Mapa do microdrive	
23.552	5C00h	Variáveis do sistema	
23.296	5B00h	Buffer da impressora	
22.528	5800h	Área de atributos	
16.384	4000h	Arquivo de imagem	
		Memória ROM	
0			

Figura 3 — Mapeamento da memória do TK 90X

Número de bytes em 6 K de memória = $1.024 \times 6 = 6.144$
 Número de bits em 6 K de memória = $6.144 \times 8 = 49.152$
 Número de pixels em 32 colunas por
 24 linhas da tela, com 64 pixels por
 caractere (matriz de 8×8) = $32 \times 24 \times 64 = 49.152$

Agora vamos ver isso em termos práticos, ou seja, como são dispostos todos esses elementos na tela.

Inicialmente, vamos considerar a tela da TV dividida em três blocos horizontais iguais. O bloco mais alto, começando na linha 0 e terminando na linha 7 é produzido através dos conteúdos dos endereços 16.384 a 18.431, ou 4000 a 47FFh. O bloco central, das linhas 8 a 15, utiliza os conteúdos dos endereços 18.432 a 20.479, ou 4800 a 4FFFh; e o bloco inferior, ou seja, o restante da tela do TK 90X, das linhas 16 a 23, utiliza os endereços 20.480 a 22.527, ou 5000 a 57FFh.

Preste atenção nos números em hexadecimal, que sugerem uma fácil manipulação do arquivo de imagem. Vamos analisar o bloco superior (e, por analogia, os outros dois blocos).

Cada posição PRINT necessita de 8×8 bits (64) e que haja 32 dessas posições em cada linha da tela ($32h = 20d$). Por outro lado, cada bloco da tela tem $32 \times 8 \times 8 = 2.048$ bytes, que, em hexadecimal, equivalem a 0800h. Vimos que os bytes correspondem à ordem: todos os primeiros bytes do primeiro bloco, os segundos bytes, os terceiros etc., o que nos dá um salto de 8! Isso significa, por exemplo, que o primeiro byte da primeira linha PRINT da tela dista da segunda de $32 \times 8 = 256$, ou 0100h.

Conclusão:

— Um byte de uma posição PRINT dista 0100h do seu byte análogo na posição PRINT imediatamente abaixo ou acima, num mesmo bloco.

— Um byte de um bloco da tela dista 0800h ou 1600h do byte com a mesma posição relativa noutro bloco, dependendo do bloco de referência e do bloco a que se refere.

Se você já fez algumas experiências em Basic com seu TK 90X e gostou, não imagine o quanto isto pode ser útil em programas de manipulação de imagens, tanto em Basic como (e principalmente) em Assembler.

— ÁREA DE ATRIBUTOS

O arquivo de imagens possui ainda 768 áreas de caracteres, que vão do endereço 22.528 a 23.295, ou 5800 a 5AFFh, onde cada uma delas contém

os códigos para cor de papel, cor da tinta, flash e bright, e são usados para armazenar valores que determinam os atributos dos 768 caracteres que podem ser impressos na tela.

Os valores (menores que 255) dos atributos podem ser considerados a partir da expressão:

$ATTR(\text{lin.}, \text{col.}) = INK + PAPER * 8 + BRIGHT * 64 + FLASH * 128$

Essa função $ATTR(\text{lin.}, \text{col.})$ é equivalente a $PEEK(22.528 + 32 * \text{lin.} + \text{col.})$

Ou seja, nos bytes dos atributos os bits 0, 1 e 2 determinam a cor da tinta; os bits 3, 4 e 5 determinam a cor do papel; o bit 6, se a posição está em modo bright ou não; e o bit 7, se está em modo flash ou não.

Por exemplo:	INK azul	= 1	-----1
	PAPER branco	= 7 ($7 \times 8 = 56$)	56
	BRIGHT 0	= 0	-----0
	FLASH 1	= 1 (1×128)	128
			185

A seguir, uma tabela de todos os valores possíveis para os atributos de qualquer posição. (Tabela 1).

Podemos imaginar cada pixel da tela da televisão (colorida) como sendo um pequeno triângulo equilátero, cujos vértices são iluminados. Para cada vértice há um pixel vermelho, um azul e um verde, e o arquivo de atributos é usado para controlar a iluminação (acesa ou não) dessas três cores diferentes. O arquivo da imagem vai indicar se um determinado pixel deve ser plotado com uma cor de tinta em particular. Os três bits mais baixos (0, 1 e 2) do valor do atributo daquela área são usados para decidir quais vértices do nosso triângulo imaginário, se o verde, o azul ou o vermelho, devem ser acesos. Nossos olhos contêm apenas três tipos de sensores de cores (verde, vermelho e azul). Nosso cérebro recebe sinal dos três vértices coloridos e combina-os em um simples pixel de cor composta. Por exemplo, se os três últimos bits de atributo são 111, equivalente à cor 7, nós temos verde+azul+vermelho, o que corresponde à luz branca. Os outros códigos de cores, quando escritos na forma binária, podem ser interpretados conforme tabela 2.

— BUFFER DA IMPRESSORA

Os endereços entre 23.296 e 23.551, ou 5B00 a 5BFFh são usados como o buffer da impressora (ZX Printer, ALphacom 32, Timex 2040 ou TK 500S quando e se for produzida). Esses 256 bytes são suficientes para

Tabela 1
TABELA DE ATRIBUTOS

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

armazenar 32 caracteres em sua forma de alta resolução, com os primeiros 32 bytes armazenando os bits da primeira linha dos caracteres, os próximos 32 bytes armazenando os bits da segunda linha, e assim por diante.

Se não houver impressora ligada ao seu micro, esta área pode ser usada como área de armazenamento de rotinas em código de máquina, elaboradas pelo usuário. ... (boa dica essa !) Não há interferência com nenhum programa que esteja no micro, mesmo que ele ocupe toda a memória.

Tabela 2

COR	CÓDIGO	BINÁRIO	VÉRTICES ACESOS
Black	0	000	
Blue	1	001	Blue
Red	2	010	Red
Magenta	3	011	Red Blue
Green	4	100	Green
Cyan	5	101	Green Blue
Yellow	6	110	GreenRed
White	7	111	GreenRed Blue

Para os interessados, aí está uma boa dica para se obter mais que 8 cores do nosso TK 90X.

– VARIÁVEIS DE SISTEMA

As 182 posições de memória entre 23.552 e 23.733, ou 5C00 a 5CB5h, armazenam as diferentes variáveis do sistema do TK 90X. Se estiver conectada ao seu micro uma Interface One da Sinclair, e um Microdrive, haverá uma ampliação dessa área, com o surgimento de novas variáveis de sistema que cuidarão exclusivamente desses periféricos. No capítulo seguinte estas variáveis serão melhor estudadas.

– MAPEAMENTO DO MICRODRIVE

Esta área da memória começa no endereço 23.734, ou 5CB6h, e tem apenas uma existência teórica na versão standard do TK 90X, ou seja, é uma área que não é usada com esse propósito, a menos que um Microdrive seja ligado a ela. (Como é, Microdigital, sai ou não sai esse periférico ?)

Enquanto sem o periférico, esta área também pode ser usada para armazenar rotinas em código de máquina, elaboradas pelo usuário.

– ÁREA DE INFORMAÇÃO DE CANAIS

Esta área especial de memória tem início no endereço apontado pela variável de sistema CHCADD, armazenada nos endereços 23.631 e 23.632, ou 5C4F e 5C50h. Esta área tem tamanho variável, mas termina num endereço que armazena um marcador final de valor 128 ou 80h.

Na versão padrão do TK 90X, ou seja, o micro sem periféricos conectados, existem detalhes de entrada/saída para quatro canais. Esses canais são:

- Canal "K" (*keyboard*) — permite entradas via teclado e saídas para a parte inferior da tela.
- Canal "S" (*screen*) — não permite nenhuma entrada, mas somente saídas normais para a tela.
- Canal "R" (*RS 232?*) — também não permite entradas, mas permite saídas para a parte inferior da tela. O tamanho desta área de vídeo pode ser expandido, se necessário.
- Canal "P" (*printer*) — também não permite entrada, mas somente saída para a impressora, via conector traseiro.

As informações de canais consistem, por canal, em 5 bytes de dados. Esses bytes fornecem o endereço da rotina de saída, que toma dois bytes; o endereço da rotina de entrada, que toma outros dois bytes; e o nome do arquivo, que é um simples código de letra correspondente.

Como existem quatro canais-padrão e um indicador de final, esta área de informação de canais no TK 90X standard ocupa os 21 endereços entre 23.734 e 23.754, ou 5CB6 e 5CCAh.

Os percursos ou caminhos que os dados percorrem para chegar a ou vir desses canais é chamado de fluxo. Associados a esses quatro canais-padrão existem quatro fluxos:

- Fluxo #0 — imprime dados na parte inferior da tela que
- Fluxo #1 — são lidos do teclado.
- Fluxo #2 — escreve dados na parte superior da tela, mas não pode ler dados.
- Fluxo #3 — envia dados de saída para a impressora, mas também não lê dados.

Cada instrução de entrada/saída de dados usa automaticamente um desses fluxos. O comando PRINT usa o fluxo #2 e o comando LPRINT usa o fluxo #3.

Portanto, PRINT "TK 90X" é uma abreviação de PRINT #2; "TK 90X"

Se você escrever LPRINT "TK 90X", o micro manda a mensagem para a impressora; mas, se você digitar LPRINT #2; "TK 90X", essa mensagem é desviada para a tela (isso funciona com os diversos comandos de saída, tais como LIST, LLIST etc.).

Analogamente, se você escrever PRINT #1; AT O, O; "TK 90X" (ou PRINT #0), o micro imprimirá a mensagem na linha 0, coluna 0, da parte inferior da tela (ou seja, a partir da linha 23). Experimente, pois isso é muito interessante na impressão de mensagens, sem interferir no arquivo de imagens!

— ÁREA DE PROGRAMAS BASIC

Esta área da memória armazena as linhas de programas em Basic, se existirem. O seu tamanho depende justamente de quantas linhas existirem.

O início da área de programa é sempre dado pelo valor armazenado na variável de sistema PROGBAS, que ocupa os endereços 23635 e 23636, ou 5C53 e 5C54h.

Note que na versão standard do TK 90X, a variável de sistema PROGBAS vai indicar que esta área de programas Basic começa no endereço 23755 ou 5CCBh, e isto sempre vai acontecer, a menos que haja um microdrive conectado ao micro, ou seja, a área de mapeamento do microdrive está sendo usada ou endereços extras foram reservados para informações adicionais de canais.

Na área de programas Basic, as linhas são armazenadas no seguinte formato:

Os primeiros dois bytes de qualquer linha armazenam o seu número, sendo o primeiro byte o mais significativo, e o outro, o menos significativo.

O terceiro e o quarto bytes armazenam o comprimento da linha. Aqui, o byte menos significativo vem antes que o byte mais significativo. O comprimento da linha significa o número de bytes, a partir do quinto byte até o byte final, o ENTER (indicador de final de linha), inclusive.

Agora começa a linha de Basic propriamente dita. Os códigos de TK 90X são usados para símbolos, palavras chaves, outros caracteres, e os códigos ASCII para caracteres alfanuméricos standard.

O último byte de qualquer linha é sempre o caractere ENTER.

Dentro de uma linha em Basic, múltiplas declarações são separadas entre si por dois pontos, caractere 58, ou 3AH. Note que se numa linha Basic existe um número, em decimal, ele é armazenado em caracteres ASCII e seguido pelo caractere NÚMERO — código 14 ou OEh, e em seguida a forma de ponto flutuante para números quaisquer ou a forma inteira para números inteiros da faixa — 65.535 a +65.535, onde, em ambos os casos, são consumidos cinco bytes. Isto significa que são reservados seis bytes de RAM para cada número decimal que estiver incluído num programa Basic !!!

O programa-demonstração a seguir, mostra o seu próprio conteúdo armazenado na área de programas.

[illegible]

Diagram illustrating the structure of a floating-point representation (REPRESENTAÇÃO EM PONTO FLUTUANTE) and its associated indicator (INDICADOR DE NÚMERO). The diagram shows a sequence of bits (0s and 1s) organized into fields. The fields are labeled as follows:

- INDICADOR DE NÚMERO (Number Indicator)
- REPRESENTAÇÃO EM PONTO FLUTUANTE (Floating-Point Representation)
- EXPO (Exponent)
- MANT (Mantissa)
- NEXT

The diagram shows the bit patterns for these fields, with the mantissa field being the largest and containing the most bits.

– ÁREA DE VARIÁVEIS

O endereço inicial desta área que armazena as variáveis correntes de um programa Basic é sempre dado pelo valor contido na variável de sistema VARADD, que ocupa os endereços 23.627 e 23.628, ou 5C4B e 5C4Ch.

No TK 90X, o tamanho dessa área não vai mudar durante a execução de um programa Basic, mas sim quando nela for introduzida uma nova variável.

O último endereço desta área sempre contém o marcador de final de área, caractere 128, ou 80h.

O programa a seguir mostra o conteúdo da área de variáveis, que contém apenas a variável de controle do loop.

Programa-demonstração da área de variáveis:

```

0>REM nao digite as linhas "R
EM" - somente a linha 10
1 REM primeiro digite em modo
direto:
2 REM PRINT PEEK 23627+256*PE
3 REM
4 REM O resultado sera 23818
5 REM que por sua vez e o val
or inicial da variavel de contro
le (<a) do loop
6 REM
7 REM
8 REM
9 REM
10 FOR a=23804 TO 23823: PRINT
a;TAB 9;PEEK a;TAB 15;CHR$ PEEK
a;NEXT a

```

— ÁREA DE EDIÇÃO DE LINHAS

O início desta área que armazena a linha Basic que está sendo digitada, ou editada, é sempre dado pelo valor armazenado na variável de sistema INADD, que ocupa os endereços 23.641 e 23.642, ou 5C59 e 5C5ah.

Quando a parte inferior da tela mostra apenas o cursor intermitente, a área de edição de linhas tem três bytes reservados para ela. O primeiro, cujo endereço é dado pelo conteúdo da variável de sistema CURADD, armazena um caractere ENTER e o segundo armazena um marcador de final — caractere 128, ou 80h.

Então, os caracteres digitados a partir do teclado são transferidos para esta área que é expandida para armazená-los.

Um procedimento similar é obtido quando se digita a tecla EDIT (CAPS SHIFT e tecla 1), para corrigir uma linha de programa. Em primeiro lugar, a área de edição é expandida para o comprimento igual ao da linha a ser editada. Então esta linha é copiada sobre o conteúdo anterior da área. (Um modo de se "limpar" esta área.)

— INPUT DADOS E ÁREA DE TRABALHO TEMPORÁRIO

Esta área da memória é usada para muitos propósitos, como, por exemplo, dados de entrada, concatenação de cadeia de caracteres etc. O endereço inicial da área é dado pelo valor armazenado na variável de sistema WORKPT, de endereços 23.649 e 23.650, ou 5C61 e 5C62h. Quando se faz necessária uma maior área de trabalho, esta área é expandida. Após o uso, ela é então esvaziada.

— ÁREA DA PILHA DE CÁLCULO

Esta é uma área muito importante da memória, e começa no endereço apontado pela variável de sistema STKCEND, de endereços 23.651 e 23.652, ou 5C63 e 5C64h, e vai até o endereço apontado pela variável de sistema ADSPFREE, de endereços 23.653 e 23.654, ou 5C65 e 5C66h.

Esta área é usada para armazenar os números em ponto flutuante em cinco bytes e, quando manipulando séries de caracteres, utiliza grupos de cinco bytes como parâmetros delas.

A manipulação desta pilha de cálculo é feita sob a forma conhecida como "LIFO", do inglês "Last-in First-out", ou seja, o último valor depositado na pilha é o primeiro a ser retirado.

— ESPAÇO LIVRE

A área de memória entre a pilha de cálculo e a pilha da máquina representa a quantidade de memória que é disponível ao usuário. Na versão do TK 90X de 16 K existem nominalmente 8.839 endereços nesta área, quando se liga o computador. Para a versão de 48 K de RAM, existem esses 8.839 mais os adicionais 32 K. Entretanto, é interessante notar que o valor mais baixo aceitável para CLEAR é 23.821, que traz a RAMTOP 8.878 bytes para baixo, ou na RAM de 48 K, 8.878 mais 32 K (41646 K).

— ÁREA DA PILHA DA MÁQUINA

O microprocessador Z80 deve ter uma área de trabalho disponível apenas para ele, que é chamada a pilha da máquina, onde ele vai manipular valores dos programas armazenados na memória.

— ÁREA DA PILHA DE GOSUB

Sempre que existir um loop de GOSUB, o número da linha desse loop será armazenado nesta área.

Cada número de linha de GOSUB requer três endereços. O mais alto armazena o número da declaração dentro de uma linha de programa Basic, para onde o retorno deverá ser feito. O próximo endereço armazena a parte menos significativa do número da linha de programa, e o último armazena a parte mais significativa desse número.

A variável de sistema RAMTOP, que ocupa os endereços 23.730 e 23.731, ou 5CB2 e 5CB3h, armazena o endereço que contém o valor 62, considerado como sendo o último endereço da área de Basic.

— ÁREA DE DEFINIÇÃO DE GRÁFICOS DO USUÁRIO

Os últimos 168 endereços da memória RAM do TK 90X são reservados para armazenar as representações dos bits dos caracteres gráficos definidos pelo usuário, a menos que tenha sido invadida pelo sistema Basic; quer pela modificação de endereços, através do uso de CLEAR, ou por um programa Basic muito extenso.

Como parte do procedimento de inicialização do TK 90X, a representação bit a bit das letras de "A" até "U" são copiadas nesta área. O usuário, ao definir alguns ou todos os 21 caracteres gráficos, muda esta representação.

O último endereço da memória é sempre fornecido pela variável de sistema MEMAVLB, que ocupa os endereços 23.732 e 23.733, ou SCB4 e SCB5h.

Na versão de 16 K do TK 90X, este endereço deve ser 32.767; e na de 48K, deve ser 65535, valor esse obtido pela expressão Basic, em modo direto:

```
PRINT PEEK 23732+256 * PEEK 23733
```

Esses nomes complicados para estas variáveis não querem dizer absolutamente nada, bem como não têm nada a ver com a linguagem Basic nem com a linguagem de máquina. Não há, portanto, necessidade de se guardá-los.

Variáveis que controlam o sistema

Os bytes na memória, do endereço 23.552 até o endereço 23.733, são reservados para uso específico do sistema. Você pode ver o conteúdo de qualquer desses endereços; convém até anotar esses valores, pois podem ser úteis mais tarde. Para ver esses conteúdos, você simplesmente digita:

```
PRINT PEEK endereço
```

Podemos também mudar esses valores iniciais, sem perigo algum de alterar o hardware standard da máquina. É somente praticando, e “fufutando”, que se descobre como ocorrem as coisas dentro dessa pequena maravilha. O máximo que pode acontecer, quando se altera uma variável de sistema, é um *crash*, e o passo seguinte nesse caso é desligar a máquina e tornar a ligá-la. Os nomes dessas variáveis não têm nada a ver com a linguagem Basic, e o computador não as reconhece pelo nome (felizmente...). Você pode usar em seus programas nomes semelhantes de variáveis, que o computador não vai confundir a sua variável com uma variável de sistema.

As abreviações utilizadas na coluna 1 da tabela a seguir tem o seguinte significado:

x – alterando o valor inicial da variável, provavelmente ocorrerá um *crash* no sistema.

n – não haverá efeito nenhum, alterando o valor inicial.

O número da coluna 1 é o número de bytes daquela variável.

Para dois bytes, o primeiro é o menos significativo – o contrário do que normalmente se espera. Portanto, para se dar um POKE de um valor v numa variável de sistema de dois bytes, de endereço n, use

POKE n, v-256 * INT (V/256)

POKE n + 1, INT (v/256)

E, para ver o valor, use a expressão:

PRINT PEEK n + 256 * PEEK (n + 1)

Se você quiser alterar algum endereço n, como no caso de aumentar a área de caracteres gráficos definidos pelo usuário, visto no capítulo anterior, deve efetuar alguns cálculos:

Chamemos de bms o byte menos significativo

BMS o byte mais significativo

VARISIT a variável de sistema que se queira alterar.

Os cálculos são:

XMS = INT (n/256)

bms = n - BMS * 256

E, a seguir:

POKE VARISIT, bms

POKE POKE VARISIT + 1, BMS

Cuidado com as maiúsculas e minúsculas!

VARIÁVEIS DO SISTEMA – TABELA DE DADOS

Nota	Endereço	Nome	Conteúdo
n8	23552/5COO	KBDWORK	Esta variável consiste em 8 bytes, cada um deles contendo informação sobre a tecla pressionada, ou o período de repetição; o código em modo <i>extended</i> etc.
n1	23560/5CO8	KEYPRS	Esta variável contém sempre o código da última tecla pressionada, tendo em conta o modo. Só é alterada quando se pressiona outra tecla. A repetição automática afeta a variável. Passando-se a zero, pode-se verificar se foi pressionada alguma tecla.
1	23561/5CO9	RPTDLAY	Tempo, em 60 ciclos por segundo, durante o qual é necessário pressionar uma tecla para acionar o auto-repeat (repetição automática). Possui o valor inicial 35, podendo ser alterado para mais ou para menos. (Cui-

1 23562/5COA RPTCCLE

n2 23563/5COB PT DEF

n1 23565/5COD K CLR

n2 23566/5COE TVCLR

x38 23568/5C10 PSTRM

dado com valores muito baixos – a repetição é muito rápida...) Tente.

Atraso, em 60 ciclos por segundo, entre as sucessivas repetições de uma tecla. Inicialmente contém o valor 5. Pode-se diminuir para o mínimo de 1, a fim de tornar mais rápida a repetição.

Endereço dos argumentos das funções definidas pelo usuário. Valor inicial 0.

Quando é escrito diretamente no teclado um código de comando, o segundo byte, ou seja, a cor e o uso ou não de FLASH é guardado aqui, enquanto o código de INK, PAPER, FLASH ou INVERSE é impresso. Depois disso, a ROM consulta este byte para impressão a seguir do código de comando.

Esta variável é usada pela rotina de impressão para guardar AT, TAB e os comandos de cor dirigidos para a tela.

Esta variável é usada para guardar os deslocamentos de CHCADD. Para um total de 19 arquivos, sendo 16 do usuário e 3 da máquina, existe um deslocamento. Quando este é somado a CHCADD, aponta para um endereço que constitui o início da rotina de tratamento desse arquivo.

2	23606/5C36	PTBL CHR	Indica o endereço do conjunto de caracteres, menos 256 (incluindo, no conjunto, todos os caracteres de código entre 32 e 127, padrão ASCII). A variável contém normalmente o endereço 15360 ou 4C00 (conjunto de caracteres da ROM), mas este valor pode ser alterado, levando-a a apontar para um novo conjunto de caracteres definido pelo usuário.
1	23608/5C38	BUZCCLE	Som produzido pelo computador quando uma linha de programa em edição ultrapassa o comprimento de 23 linhas da tela (tela cheia). Para alterar este comprimento, modifique o conteúdo desta variável, inicialmente em 64.
1	23609/5C39	KCLICK	Duração do som produzido ao se pressionar uma tecla. Pode-se aumentar seu valor inicial, a fim de tornar mais "simpático" o som produzido.
1	23610/5C3A	ERRCD	Indica o código de mensagens menos 1. Inicialmente em 255. Se for alterada por uma listagem de programa Basic, quando este programa terminar, a máquina imprime a mensagem de erro desejada.
x1	23611/5C3B	SFLAG O	Esta variável contém um byte cujos valores contém bandeiras (indicadores) que controlam o sistema Basic.

x1	23612/5C3C	SFLAG 1	Bandeiras associadas à impressão na tela e na impressora.
x2	23613/5C3D	P ERR	Esta variável aponta para um elemento da pilha da máquina. Quando ocorre um erro, é para este endereço que a execução salta depois da pilha ser limpa por RST 08 (instrução em código de máquina). Alterando este elemento, é possível escrever novas rotinas de tratamento de erro, ou provocar um <i>crash</i> no sistema.
n2	23615/5C3F	PLIST	Esta variável aponta para o endereço de retorno da pilha da máquina, para o qual salta a execução após uma listagem automática.
n1	23617/5C41	CURSOR	Esta variável especifica o cursor em uso (K, L, C, E ou G).
2	23618/5C42	LNJMP	Linha para onde deve saltar a execução do programa. Usada nas instruções GOTO e GOSUB.
1	23620/5C44	INSTRNR	Número da declaração da linha para onde deve saltar a execução. Alterando primeiro a variável LNJMP e depois esta, força-se um salto para uma determinada declaração da linha.
2	23621/5C45	EXCLINE	Número da linha onde se encontra a instrução em execução.
1	23623/5C47	SUBLEXC	Número da instrução em execução dentro de uma linha.

1	23624/5C48	BORCLR	Esta variável contém os atributos da parte inferior da tela. É uma experiência muito interessante alterar seu valor e imprimir mensagens com diversos valores.
2	23625/5C49	CURLINE	Número da linha onde se encontra o cursor.
x2	23627/5C4B	VARADD	Indica o endereço inicial da área de variáveis Basic, quando da execução de um programa.
n2	23629/5C4D	XVARADD	Endereço da variável em atribuição.
x2	23631/5C4F	CHCADD	Indica tabela de endereços usada por PSTRM.
x2	23633/5C51	IOADD	Indica endereço da tabela anterior (de endereços de tratamentos de arquivos) que está sendo usado pela rotina de tratamento.
x2	23635/5C53	PROGBAS	Indica o endereço de início da área de programas Basic.
x2	23637/5C55	NEXEXC	Indica o endereço da linha de programa Basic seguinte a que está sendo executada.
x2	23639/5C57	ENDDATA	Aponta para o separador final do último elemento de uma listagem DATA. Se não existir DATA no programa, aponta para 80h, no fim das informações de canal.
x2	23641/5C59	INADD	Endereço da instrução a ser digitada no teclado.

2	23643/5C5B	CURADD	Endereço do cursor no interior da linha avaliada.
x2	23645/5C5D	CHNXADD	Endereço do próximo caractere a ser interpretado.
2	23647/5C5F	SYCHADD	Endereço do caractere que se encontra depois do sinal "?".
x2	23649/5C61	WORK PT	Endereço da área de trabalho temporário.
x2	23651/5C63	STKEND	Endereço inferior da pilha de cálculo.
x2	23653/5C65	ADSPFREE	Endereço do início da memória livre.
n1	23655/5C67	BREGCAL	Registro de cálculo usado para fins diversos (contagem).
n2	23656/5C68	MEMCADD	Endereço usado para as seis memórias de cálculo (normalmente MENSPCAL, mas nem sempre).
1	23658/5C6A	SFLAG 2	Mais bandeiras controladoras do sistema.
x1	23659/5C6B	SIZE	Número de linhas (incluindo uma em branco) da parte inferior da tela. Ela pode ser alterada fazendo com que o programa ocupe toda a tela, mas cuidado: antes de cada instrução que ocupe esta área (por exemplo, INPUT), a variável deverá conter o valor original(2), ou seja, reabrir espaço na parte inferior.
2	23660/5C6C	LIST NR	Número da linha superior em listagem automática.
2	23662/5C6E	CONTJMP	Número da linha para onde salta o comando CONT.

1	23664/5C70	CONTNR	Número da instrução da linha destino após o comando CONT.
n1	23665/5C71	SFLAG 3	Mais bandeiras controladoras do sistema.
1	23681/5C81		Não usada pelo sistema. Alguns programas em linguagem de máquina têm a instrução RET (201) armazenada nesta posição, para evitar que, ao se dar o comando LOAD "" CODE, se tenha acesso aos seus códigos. . .
2	23682/5C82	HVBFFIN	Número de colunas (32) e linhas (24) do final da área de entrada.
2	23684/5C84	DFPSPRT	Endereço da posição PRINT no arquivo de imagens. Pode ser dirigida para outra posição.
2	23686/5C86	DFPSPRTL	Idêntica à anterior, mas para a parte inferior da tela.
x1	23688/5C88	HVPOS	Número de colunas para a posição PRINT.
x1	23689/5C89		Número de linhas para a posição PRINT.
x2	23690/5C8A	HVPSL	Idêntica à anterior, mas para a parte inferior da tela.
1	23692/5C8C	SCRINC	Contador de "scroll" da imagem: contém um a menos que o número de scrolls realizados antes de a imagem parar imprimindo a mensagem "scroll?". Se for colocado um valor maior aqui, a imagem rolará ininterruptamente.
1	23693/5C8D	ATCLRP	Atributos permanentes definidos por declarações INK, PAPER etc.

1	23694/5C8E	MASKCLRP	Usado para atributos transparentes. Qualquer bit igual a 1 indica que o atributo correspondente não é retirado de ATCLRP, mas do valor que já está na tela.
n1	23695/5C8F	ATCLRT	Atributos provisórios em uso, em instruções tipo PLOT, DRAW etc.
n1	23696/5C90	MASKCLRT	Como MASKCLRT, mas provisório.
1	23697/5C91	SFLAG 4	Outras bandeiras controladoras do sistema.
n30	23698/5C92	MEMSPCAL	É nesta área que a unidade de cálculo aritmético pode guardar até 6 números diferentes, em notação de ponto flutuante, com 5 bytes cada, utilizando memórias especiais.
2	23728/5CBO	NMIVCT	Ex-veter de interrupção, não usado devido às características de programação da ROM. Pode ser usada sem restrições pelo usuário.
2	23730/5CB2	RAMTOP	Esta variável contém o endereço do último byte da área Basic.
2	23732/5CB4	MEMAVLB	Endereço do último byte físico da RAM.
n2	23666/5C72	STRVLEN	Comprimento da série de caracteres em atribuição.
n2	23668/5C74	SYTADD	Endereço do elemento seguinte na tabela de sintaxe da ROM. Esta tabela define o local onde se encontra a rotina correspondente a cada comando e o modo de obter a informação necessária.

2	23670/5C76	INITRND	O primeiro número para a instrução RND. Esta variável é inicializada por RAND.
3	23672/5C78	TVCOUNT	Contador de imagens de 3 bytes, incrementado a cada ciclo de alimentação.
2	23675/5C7B	UDGRAPH	Endereço do primeiro caractere gráfico definido pelo usuário. RAMTOP não afeta esta área.
1	23677/5C7D	LSTPLOT	Usada para armazenamento provisório da coordenada x quando são realizados cálculos para definição da posição de PLOT.
1	23678/5C7E	COORDS	Idêntica à anterior, mas para a coordenada y.
1	23679/5C7F	POSIMPR	Número da posição de impressão, em 32 colunas.
1	23680/5C80	PRTADD	Byte menos significativo do endereço da posição que se segue para LPRINT (no buffer da impressora).

Alguns comentários sobre as variáveis de sistema

1 - KBDWORK

Esta variável armazena:

- a) 255 se nenhuma tecla foi pressionada ou
- b) o código do maior caractere em branco à esquerda impresso em uma tecla.

No último caso, o código pode ser considerado como sendo aquele caractere que INKEY\$ produziria se CAPS LOCK (tecla 2) estivesse acionado.

Esta propriedade pode ser usada com vantagens, quando se usa INKEY\$ em um programa. Digite o programa a seguir e veja qual o efeito que CAPS SHIFT ou SYMBOL SHIFT produzem quando pressionados com outras teclas:

```
10 PRINT AT 0,0; INKEY$; "  ": REM 4 espaços dentro das aspas
20 GOTO 10
```

Como você pode notar, o caractere produzido não depende apenas de qual tecla foi pressionada, mas também em que modo se encontrava o computador. Isto conduz a linhas meio "complicadas" quando se usa INKEY\$, por exemplo, ao final de um programa:

```
9000 PRINT "Quer rodar outra vez? (s/n)": PAUSE 0:
      IF INKEY$="s" OR INKEY$="S" OR INKEY$="NOT"
      THEN RUN
9010 NEW
```

Podemos utilizar também o byte do endereço 23552:

```
10 PRINT AT 0,0; CHR$ PEEK 23552; "  ";
20 GOTO 10
```

Você vai perceber que qualquer que seja a tecla pressionada, o caractere produzido será aquele que aparece em branco no lado esquerdo da tecla que está sendo pressionada. A solução para o problema:

```
9000 PRINT "Quer rodar outra vez? (s/n)": PAUSE 0:
      IF CHR$ PEEK 23552="S" THEN RUN
9010 NEW
```

Se você pretende usar esta técnica em seus programas, faça da seguinte maneira:

```
10 LET kbd = 23552
```

E mais adiante,

```
... IF CHR$ PEEK kbd = ... THEN ...
```

Agora experimente o programa abaixo, para ver qual endereço desta variável utilizar em seus programas:

```
10 FOR f = 23552 TO 23559
20 PRINT INKEY$; "  "; CHR$ PEEK f;
30 GOTO 20
```

Quando quiser mudar para o próximo endereço, ou para o próximo f, basta interromper o programa e digitar NEXT f.

Notou que a penúltima tecla pressionada é impressa com a última, através de CHR\$ PEEK f? Retire este comando da linha 20 e preste atenção nas diferenças.

2 - KEYPRS

O conteúdo deste endereço vai produzir o código da última tecla pressionada, esteja você pressionando outra ou não. Como no caso da variável anterior, se mais de uma tecla é pressionada, então o código da primeira a fazer contato tem prioridade sobre a outra. É importante notar, que embora CAPS SHIFT ou SYMBOL SHIFT sozinhas não alterem o conteúdo de KBDWORK, ou mesmo de INKEY\$, juntas produzem código 14, que normalmente é usado para representação de números em listagens Basic do TK 90X.

Existem outras quatro combinações de teclas que produzem valores para bytes de 23560 e códigos de INKEY\$, que não aparecem no manual do TK 90X. São elas:

Teclas normalmente usadas para produzir:	Valor do código INKEY\$
GRÁFICOS (CAPS SHIFT & 9)	15
TRUE VIDEO (CAPS SHIFT & 3)	4
VÍDEO INVERSO (CAPS SHIFT & 4)	5
CAPS LOCK (CAPS SHIFT & 2)	6

3 - RPTDLAY E RPTCCLE

Experimente rodar o programa abaixo, que é auto-explicativo, para ver o que acontece com o dispositivo de repetição automática de teclado, e o tempo de leitura do teclado:

```

10 PRINT "RPTDLAY - 23561 = "; PEEK 23561 "RPTCCLE - 23562 = ";
  PEEK 23562
20 LET a = PEEK 23561: LET b = PEEK 23562
30 INPUT "Digite valor para mudar RPTDLAY"; m
40 INPUT "Digite valor para mudar RPTCCLE"; n
50 POKE 23561, m: POKE 23562, n
60 INPUT "Experimente digitar qualquer coisa"; c$
70 PRINT c$
80 POKE 23561, a: POKE 23562, b
90 PRINT "Valor usado para RPTDLAY = "; m; "Valor usado para RPTCCLE
  = "; n
100 INPUT "Outra tentativa (s/n) ?"; a$
110 IF a$ = "s" THEN RUN
120 STOP

```

4 - MUDANDO MODOS DO CURSOR

A variável de sistema de endereço 23617 especifica o modo do cursor; através dela o programador pode forçar apenas o modo gráfico e o modo estendido, no próximo comando INPUT.

Para modo gráfico: POKE 23617,2: INPUT a\$: PRINT a\$

Para modo estendido: POKE 23617,1: INPUT a\$: PRINT a\$

Experimente o programa, alternando esses dois POKES:

```

10 FOR f = 0 TO 255
20 PRINT f; TAB 8;
30 INPUT a$: PRINT a$; INPUT a$: PRINT a$
40 NEXT f

```

Portanto, através de CURSOR, podemos apenas entrar em modo gráfico e modo estendido.

Através de uma variável "misteriosa", localizada no endereço 23658, podemos também entrar em modo maiúsculo (CAPS LOCK). Digitando-se POKE 23658,8, o endereço de SFLAG 2, automaticamente estamos com letras maiúsculas.

Isso pode facilitar em certos programas que possuam uma linha do tipo

```

100 INPUT "Quer rodar outra vez (s/n)"; s$: IF s$ = "S" OR s$ = "s" THEN
  RUN

```

Podemos passar esta linha para:

```

100 POKE 23658,8: INPUT "Quer rodar outra vez (s/n)"; s$: IF s$ = "S" THEN
  RUN: POKE 23658,0: REM entra em modo maiúsculo e sai desse modo
  através de POKE 23658,0.

```

Você pode, também, dependendo do tipo de programa que esteja fazendo, perguntar logo de início se deseja trabalhar somente em letras maiúsculas ou não.

Esta técnica é muito interessante quando existem nos programas diversos comandos tipo INPUT e INKEY\$.

5 - CORES DA TELA

BORCLR

Embora o TK 90X proteja o usuário contra acidentes que fazem INK e PAPER, em comandos INPUT, serem da mesma cor, você, se quiser, pode alterar somente as duas linhas da parte inferior do vídeo, alterando os valores da variável de sistema BORCLR, de endereço 23624. Isso pode ser particularmente interessante, quando você não quer de modo algum que alguém veja a listagem de seu programa, e, dessa forma, não é permitido imprimir qualquer coisa na parte inferior da tela.

É interessante notar que, através do uso dessa variável, podemos também fazer FLASH e BRIGHT nas linhas de INPUT, adicionando:

128 para FLASH 1 e
64 para BRIGHT 1

O novo valor de BORCLR permanece inalterado até ser executado um novo comando BORDER ou um NEW.

Por exemplo, para produzir uma borda magenta com tinta amarela e papel magenta, nas linhas inferiores, com FLASH 1 e BRIGHT 1, digite:

BORDER 3:POKE 23624, 128 + 64 + 3 * 8 + 6 :CLS

ATCLRP - MASKCLRP - ATCLRT - MASKCLRT

Estas variáveis do sistema simplesmente armazenam os valores que estão sendo usados para INK, PAPER, BRIGHT e FLASH. A diferença entre os nomes, ou seja, a letra P e T, significa Permanente ou Temporário (estas, quando incluídas em algum comando de saída de dados).

ATCLRP e ATCLRT são pouco utilizadas, mas para referência os valores a serem adicionados nelas, para tratamento de cores são

$ATCLR = 8 * (\text{cor de PAPER}) + \text{cor da INK} + (128 \text{ para FLASH } 1) + (64 \text{ para BRIGHT } 1)$

Já MASKCLRP e MASKCLRT são mais úteis. Ocupam respectivamente os endereços 23694 e 23696.

Qualquer bit de um byte de uma destas variáveis, que tiver o valor 1, mostra que o bit correspondente de atributos para impressão deve ser tomado da posição corrente de impressão da tela, como no caso de INK, PAPER, BRIGHT e FLASH 8.

O ponto mais interessante destas variáveis é que nós podemos limitar seu efeito a apenas cores primárias (azul, vermelho e verde) de códigos do TK, respectivamente 1, 2 e 4. Os valores para POKE nas variáveis são:

BIT	EFEITO	VALOR
0	azul INK 8	1
1	verm INK 8	2
2	verde INK 8	4
3	azul PAPER 8	8
4	verm PAPER 8	16
5	verde PAPER 8	32
6	BRIGHT 8	64
7	FLASH 8	128
		255

6 - O RELÓGIO DO TK 90X

Existe no conjunto das variáveis do sistema do TK 90X, uma variável, que constantemente altera seus valores, como que contando "tempo", chamada TVCOUNT, que possui seu valor inicial incrementado 60 vezes por segundo. Esta frequência é igual à frequência ou ciclagem da área, e também ao número de vezes por segundo que uma nova imagem é enviada à televisão para formar o que vemos.

TVCOUNT é inicializada em zero, quando o computador é ligado, e incrementada a cada 20 milissegundos aproximadamente, a menos que um comando SOUND, uma operação com fita cassete ou um outro periférico estejam ligados ao TK 90X.

Esta é a justificativa para o comando PAUSE n, que significa simplesmente "aguarde até que TVCOUNT seja incrementado de n".

TVCOUNT está armazenada em 3 bytes: 23672, 23673 e 23674. Cada byte tem 8 bits, e o valor máximo de TVCOUNT é 2 elevado a 24 menos 1, ou seja, 16777215, o que corresponde a aproximadamente 3 dias, 21 horas e alguns minutos, a partir do momento em que o computador for ligado (se você fizer esta experiência, faça-a antes que acabe a garantia). Atingindo esse valor, TVCOUNT é zerada e inicia nova contagem.

O valor de TVCOUNT pode ser encontrado através da linha:

PRINT PEEK 23672 + 256 * PEEK 23673 + 65536 * PEEK 23674

Experimente digitar o programa a seguir, que simula um relógio muito simples:

```

10 LET tvcount = 23672
20 LET min = 0
30 POKE tvcount,0: POKE tvcount + 1,0: POKE tvcount + 2,0
40 PRINT AT 1,7;"Segundos = ";
50 LET t = PEEK tvcount + 256 * PEEK (tvcount + 1) + 65536 * PEEK
  (tvcount + 2)
60 LET seg = INT (t/50): REM desconte um tempo de processamento
70 PRINT AT 1,1;seg
80 LET min = INT (seg/60)
90 PRINT AT 3,7;"Minutos = "; AT 3,18;min
100 GOTO 50

```

O potencial desta variável é muito grande: serve para contador de tempo em diversos tipos de programas, até mesmo para aqueles que pretendem dar assessoria por computador por hora!

7 – ROLANDO A IMAGEM

Um pequeno problema que surge para programadores em Basic do TK 90X é como não deixar o computador perguntar "Scroll?" a cada vez que a tela é preenchida com dados.

Existe uma variável de sistema, chamada SCRINC (de INCrementador de SCRoll), de endereço 23692 que possui o valor de 1 a menos que o número de linhas de tela que serão roladas para cima até a próxima pergunta "Scroll?" (normalmente, esse valor é menor que 23). Portanto, devemos colocar nessa variável um valor maior que o número de linhas que rolam para cima na tela, ou seja, de imediato, 255, que é o valor máximo.

Experimente:

```

10 PRINT AT 21,31'
20 PRINT PEEK 23692
30 GOTO 20

```

Rode o programa e introduza a seguir a linha 15 POKE 23692,255, e rode novamente.

8 – VARIÁVEIS DE CONTROLE DA MEMÓRIA

Existem algumas variáveis do sistema, que servem apenas para o computador saber o estado em que se encontra a sua memória, como, por exemplo, saber onde começa o programa em Basic, onde estão as variáveis desse programa etc. Muitas dessas variáveis do sistema têm pouco interesse para o programador. Veremos algumas delas.

Progbas

Os endereços 23635 e 23636 dizem ao computador onde começa o programa Basic.

PRINT PEEK 23635 + 256 * PEEK 23636 dá o valor de PROGBAS, ou seja, o início da área de programas Basic. Some cinco a este valor e você terá o endereço do primeiro caractere após a declaração REM da primeira linha de um programa qualquer (sem microdrive ou interface), cheia de códigos esquisitos, que são, na verdade, uma rotina em código de máquina.

Se você pretende criar uma rotina em linguagem de máquina e armazená-la dessa forma, não esqueça de deixar, após a declaração REM, tantos espaços quantos forem os bytes da sua rotina.

Para introduzir uma mensagem de autor de programa e deixá-la protegida na primeira linha, digite:

```
POKE (PROGBAS),0: POKE (PROGBAS + 1),0
```

Se o número anterior da primeira linha do seu programa for menor que 256, então, o primeiro dos dois comandos acima pode ser omitido. Este procedimento "renumera" a primeira linha do programa, atribuindo a ela o número 0, tornando impossível uma simples edição dessa linha.

Varadd

Esta variável armazena o endereço das variáveis Basic, através da expressão:

```
PRINT PEEK 23627 + 256 * PEEK 23628
```

As variáveis Basic estão situadas imediatamente após o programa Basic na memória do TK 90X; portanto, nós podemos encontrar o comprimento de um programa através da simples subtração de PROGBAS de VARADD:

```
PRINT "O programa tem"; 256 * ((PEEK 23628 - PEEK 23636) + (PEEK 23627 - PEEK 23635));
```

Ramtop e Adspfree

Estas duas variáveis podem ser utilizadas para medir a quantidade de memória disponível ao usuário. No mapeamento da memória do TK 90X, você percebe que entre ADSPFREE e RAMTOP existem apenas o espaço livre, a pilha da máquina e a pilha de GOSUB.

RAMTOP é dada por:

```
PRINT PEEK 23730 + 256 * PEEK 23731
```

e ADSPFREE por:

```
PRINT PEEK 23653 + 256 * PEEK 23654
```

Como alternativa da segunda expressão, existe na ROM do TK 90X uma rotina que faz isso, e que começa no endereço 7962. Em vez de toda essa expressão você pode simplesmente digitar PRINT USR 7962, o que dá no mesmo.

Para estimar a quantidade de memória disponível, subtraia ADSPFREE de RAMTOP.

Ainda sobre ADSPFREE, você provavelmente deve saber como parar aquele programa maravilhoso que quando carrega da fita, automaticamente já entra rodando. Você sabe que ele foi gravado com SAVE. . . LINE. . . Para interromper esse programa, você deve usar MERGE " ", em vez de LOAD " ", o que cria um grande problema para as software-houses. . . Pois bem, uma maneira de resolver esse problema é gravar o programa como sendo um bloco de bytes, através da inserção das seguintes linhas no final do programa:

```
9997 LET adspfree = PEEK 23653 + 256 * PEEK 23654
9998 SAVE "nome" CODE 23552, adspfree + 23500
9999 RUN
```

Dessa forma, toda a memória é enviada para a fita cassete, incluindo o programa, as variáveis Basic, as variáveis do sistema e a pilha do computador, como um bloco de códigos, de tal forma que, quando for carregado de volta ao computador, este vai iniciar o seu processamento a partir da linha em que parou, ou seja, a partir da última linha do programa (não esqueça que, para carregar o programa, você deve usar LOAD " " CODE).

9 — DIVERSAS VARIÁVEIS

Para saber qual o número da próxima linha a ser interpretada, digite:

```
PRINT PEEK 23662 + 256 * PEEK 23663 (SERÁ O COMANDO TRACE?)
```

Para saber qual o número da próxima declaração naquela linha, digite:

```
PRINT PEEK 23664
```

Os endereços 23677 e 23678 armazenam respectivamente as coordenadas X e Y do último ponto plotado (variáveis de sistema LSTPLOT e

COORDS). Você pode tratar isso criando duas variáveis em Basic, quando usando linhas com PLOT ou DRAW; se você começar o programa com

```
LET xo = 23677: LE yo = 23678
```

toda vez que precisar saber qual foi o último ponto plotado, PEEK xo e PEEK yo lhe darão as coordenadas desse ponto.

Para desenhar uma linha a partir do último ponto plotado DRAW (x - PEEK xo), (y - PEEK yo) ou, então, simplesmente, como você pode deduzir, dar POKEs nessas variáveis, para alterar a posição do último ponto plotado, ou então para desenhar pontos e retas sem utilizar os comandos PLOT e DRAW.

HVPOS — Armazena a posição corrente de impressão, mas não da maneira que você deve estar aguardando. Se você deu algum comando tipo PRINT AT lin, col, de linha e coluna, então,

endereço 23688 armazena 33 - col.

endereço 23689 armazena 24 - lin.

Portanto, para encontrar a posição corrente de impressão

```
LET lin. = 24 - PEEK 23689
```

```
LET col. = 33 - PEEK 23688
```

Se você pretende usar essa técnica em seus programas, até mesmo em conjunto com SCREEN\$, então faça-o da seguinte forma:

```
DEF FN y() = 24 - PEEK 23689
```

```
DEF FN x() = 33 - PEEK 23688
```

INITRND — Esta é a variável utilizada para gerar o último número randômico (aleatório), e está armazenada nos endereços 23670 e 23671. Experimente digitar:

```
PRINT RND, (PEEK 23670 + 256 * PEEK 23671) / 65536
```

e serão impressos dois valores iguais. Cada vez que RND é usada, INITRND é alterada pelo computador da seguinte maneira:

Novo INITRND = (75 (INITRND + 1)) mod. 65537 - 1, o que corresponde em Basic a:

```
LET INITRND = 75 * (INITRND + 1): LET INITRND = INITRND - 65537 * INT (INITRND/65537) - 1
```

O novo valor de INITRND é então armazenado e dividido por 65536, para produzir um valor para RND entre 0 e 1.

DFPSPRT e DFSPRTL — Estas duas variáveis de sistema, armazenadas nas locações 23684 e 23686, armazenam o endereço no arquivo de imagens das posições de impressão, tanto para a parte superior da tela, quanto para a parte inferior.

Normalmente não usamos PEEK e POKE no arquivo de imagens, pois temos três comandos que suprem qualquer necessidade, que são, POINT (x,y), PRINT e SCREEN\$ (x,y).

Para qualquer posição de impressão na tela, nas posições X e Y, temos que:

$$\text{DFPSPRT} = 2048 * \text{INT} (Y/8 + 8) + (Y - 8 * \text{INT} (Y/8)) * 32 + X$$

Não esqueça que cada caractere na tela é armazenado em 8 bytes na memória (um para cada linha de caractere). Os endereços desse grupo de 8 bytes de cada caractere da tela são separados de outro na mesma posição, em outra parte do arquivo de imagem, por 256. Portanto, se DFPSPRT é a primeira linha, na segunda é DFPSPRT + 256, na terceira é DFPSPRT + 512, e assim por diante.

Experimente digitar o programa a seguir, que ilustra bem essas experiências:

```
10 BORDER 0:PAPER 0:INK 6:CLS
20 PRINT AT INT (RND * 22), INT (RND * 32);
30 LET DF = PEEK 23684 + 256 * PEEK 23685
40 FOR c = 0 TO 1
50 FOR a = DF TO DF + 7 * 256 STEP 256
60 READ b:POKE a,b:NEXT a
70 NEXT c
80 RESTORE:GOTO 40
90 DATA 24, 60, 126, 25, 31, 254, 60, 24, 248, 60, 23, 15, 15, 23, 60, 248
```

Se você realmente quiser proteger seus programas contra aqueles "piratas profissionais", dê um POKE na variável de sistema P ERR, de endereço 23613, com valor 0

```
POKE 23613, 0
```

E coloque esse comando como sendo o primeiro da primeira linha do programa, para então experimentar dar um BREAK no programa e ver o que acontece...

Algo sobre os comandos IN e OUT de Basic

Uma falha gritante do manual do TK 90X!

Estes dois comandos são importantíssimos, tanto a nível de Basic, quanto a nível de linguagem de máquina.

Os microcomputadores, ou melhor, qualquer computador deve poder se comunicar com o exterior. Este mundo exterior inclui normalmente o leitor, a sua televisão, o seu gravador, uma impressora, microdrives etc., enfim, tudo que possa ser conectado à CPU do micro.

Dividimos os modos de comunicação de um computador, em duas categorias principais: as entradas e as saídas. As entradas são as informações recebidas pelo computador, através de qualquer periférico de entrada. As saídas, ao contrário, são as informações enviadas pelo computador para um periférico de saída. Para poder se comunicar com o exterior, o computador, qualquer que seja ele, utiliza o que chamamos de portas (*ports*, em inglês). O nome em si tem lógica.

O microprocessador pode ler e escrever dados na memória através das instruções PEEK e POKE. Mas, para ele não interessa se está lendo ou escrevendo na ROM ou na RAM; ele sabe somente que existem no máximo 65536 endereços disponíveis de uma vez, para sua pesquisa.

Numa total analogia, podemos dizer que existem 65536 portas de entrada ou saída (em inglês, *I/O ports* - de *Input* e *Output*). Estas, como já vimos, são usadas pelo sistema para comunicação com o meio exterior, e podem ser controladas pela linguagem Basic, através das instruções IN e OUT.

A instrução IN é similar a PEEK

IN endereço

Ela possui um argumento, o endereço da porta, e seu resultado é o byte lido daquela porta.

OUT é uma instrução similar a POKE

OUT endereço, valor

que escreve naquela porta especificada, o valor determinado.

A maneira como esse endereço é interpretado depende muito do estado do computador, ou seja, se existe um periférico conectado a ele. No TK 90X é preferível imaginarmos esses endereços escritos na sua forma binária, porque cada bit do endereço trabalha de maneira independente, ou seja, depende do estado da máquina, ou da finalidade do processo.

São no total, 16 bits, assim denominados:

A15, A14, ..., A10, ..., A2, A1, A0

onde A0 é o primeiro bit, A1 é o segundo, e assim por diante, até A15 que é o último. Normalmente esses bits têm o valor 1, mas se um deles tem valor zero, significa que o computador deve realizar uma tarefa específica. Os bits A0, A1, A2, A3 e A4 são os mais importantes. O computador não pode fazer mais que uma tarefa por vez, portanto apenas um destes cinco bits deve ser 0, de cada vez. Os bits A6 e A7 são ignorados. Os melhores endereços para serem utilizados são aqueles múltiplos de 32, menos 1.

Os bits de A8 em diante, algumas vezes, transmitem alguma informação extra ao computador.

O byte lido ou escrito tem 8 bits, que são chamados de D0 a D7.

Existe um grupo de endereços de entrada, que tanto lêem o teclado, como o soquete EAR do micro.

O teclado é dividido em oito meias linhas de cinco teclas cada:

- IN 65278 le de CAPS SHIFT até V
- IN 65022 le de A até G
- IN 64510 le de Q até T
- IN 63486 le de I até S
- IN 61438 le de O até 6
- IN 57342 le de P até \

- IN 49150 le de ENTER até H
 - IN 32766 le de SPACE até B
- (Estes endereços são $254 + 256 * (255 - 2^n)$, com n variando de 0 a 7).

No byte lido, os bits D0 a D4 associam-se às cinco teclas daquela meia linha — sendo D0 associado à tecla mais externa e D4 associado à tecla mais próxima ao meio do teclado. O bit será 0 se a tecla foi pressionada, e 1 em caso contrário. D6 é o valor no soquete EAR traseiro.

A porta de endereço 254, com relação a saídas, dirige o som para a TV (D4), o soquete MIC traseiro (D3) e também muda as cores da borda da tela (D2, D1, D0).

A porta de endereço 251 dirige a impressora ligada no conector traseiro, tanto em leitura, quanto em escrita, ou seja, sabendo se a impressora terminou ou não a mensagem já enviada.

As portas de endereço 254, 247 e 239 são usadas para periféricos, como microdrives, interfaces seriais etc.

A porta de endereço 63 é usada para sintetizadores de voz (nem todos — são características de cada um).

Experimente o programa:

```
10 FOR x = 0 TO 7
20 LET m = 254 + 256 * (255 - 2^x)
30 PRINT AT 0,0; IN m: GOTO 30
```

e vá pressionando as teclas para saber os valores respectivos de cada uma. Quando você completar a digitação de cada meia linha, pressione BREAK e dê, em modo direto, NEXT x, para passar para outra meia linha. Anote os valores encontrados.

NOTA: Infelizmente, nem todos os TK são iguais... Eu, particularmente, já estou no meu terceiro TK 90X, que foi o primeiro a aceitar o Microdrive, a Interface One e uma impressora Alphacon 32, todos juntos. Os outros TK que tive não reconheciam os comandos do Microdrive, assim que eu ligava a impressora neles. Particularidade muito interessante esta!

Capítulo

4

As cores pelo teclado

Apesar de tudo o que já foi dito tanto no Manual como neste livro sobre como se obter cores para INK e PAPER, ou como entrar em modo FLASH ou BRIGHT, existe outra maneira de se colorir mensagens, gráficos criados pelo usuário, ou texto; através de códigos que são obtidos pressionando-se diretamente certas teclas.

Por exemplo, para se digitar uma linha de programa, que seja com INK 1 e PAPER 1, você deve fazer o seguinte:

- 1- Digite o número da linha;
- 2- Entre em modo EXTENDED (cursor E), pressionando CAPS SHIFT junto com SYMBOL SHIFT;
- 3- Pressione a tecla 1 — pronto! Você já obteve papel azul;
- 4- Pressione novamente CAPS SHIFT (ainda em modo EXTENDED) e outra vez a tecla 1 — e você passará a escrever com tinta azul, ou seja, com a tinta da mesma cor que o papel, para que, por exemplo, nenhum pirata tenha acesso ao seu *copyright* da primeira linha do programa.

Essa sequência de comandos vale para qualquer tecla de 0 a 7, referente às cores.





















A tecla 8 pode tanto desligar o FLASH quanto o BRIGHT, dependendo de qual tecla você pressionar, antes de pressionar a 8.

A tecla 9, ao contrário da 8, pode tanto ligar o FLASH quanto o BRIGHT.

Portanto, nessa primeira fileira de teclas, de 1 a 0, cada uma delas pode ter até 8 funções (não existe teclado mais inteligente do que este), dependendo somente do modo em que se encontra o computador naquele momento, e, principalmente, da sua prática com o teclado.

A seguir, um quadro ilustrativo dos diversos modos de se utilizar essas teclas, que por omissão, creio eu, não saiu no manual do TK 90X.

QUADRO RESUMO DAS TECLAS 1 ATÉ 0

Teclas											
Modo	Shift	DEF FN	FN	LINE	OPEN	CLOSE	MOVE	ERASE	POINT	CAT	FORMAT
E	Symbol										
	Caps	INK BLUE	INK RED	INK MAGENTA	INK GREEN	INK CYAN	INK YELLOW	INK WHITE	FLASH 0	FLASH 1	INK BLACK
	Nenhum	PAPER BLUE	PAPER RED	PAPER MAGENTA	PAPER GREEN	PAPER CYAN	PAPER YELLOW	PAPER WHITE	BRIGHT 0	BRIGHT 1	PAPER BLACK
G	Caps ou Symbol									SAI DE GRAPHICS	DELETE
	Nenhum									SAI DE GRAPHICS	DELETE
K·L·C	Caps	EDIT	CAPS LOCK	TRUE VIDEO	INVERSE VIDEO					MODO GRAPHICS	DELETE
	Symbol		Ⓐ	#	\$	%	a	.	()	—
	Nenhum	1	2	3	4	5	6	7	8	9	0

O modo gráfico

Você já deve ter notado que este micro não possui página gráfica de alta resolução.

Ótimo. Porque essa página gráfica de alta resolução é apenas uma área da memória RAM reservada estritamente para desenhos, não se misturando com textos.

No TK 90X, em Basic mesmo, você já deve ter visto que pode misturar texto e desenhos como quiser. Não tem que dar nenhuma instrução especial, para entrar e sair de modo gráfico algum. É uma grande vantagem desta máquina.

Mas, a operação de desenhar um ponto na tela possui algumas diferenças do modo de impressão de textos.

Um comando PRINT AT 0,0 imprime uma mensagem, ou um caractere na primeira linha do vídeo, ou seja, a linha de topo (a primeira superior). Você sabe que existem 22 linhas disponíveis. Para texto, começando em zero e terminando em 21. Mas se você quiser mais duas linhas (lembra-se das variáveis do sistema?), basta dar POKE 23659,0, que faz sumir a parte inferior da tela — tome cuidado, pois estas linhas da parte inferior obrigatoriamente devem existir antes de um comando INPUT, por exemplo, ou antes de ser impressa uma mensagem lá (até mesmo SCROLL?)

Já com o comando PLOT, ou outro comando de desenho, ocorre o inverso.

Um comando PLOT 0,0 desenha um ponto no canto inferior esquerdo do vídeo. Um PLOT 255,0 desenha um ponto no canto inferior direito, enquanto PLOT 175,0 desenha um ponto no canto superior esquerdo da tela e PLOT 255,175 desenha um ponto no canto superior direito da tela. Lembre-se de que a capacidade gráfica do TK 90X é de 256 pontos na horizontal (eixo X), e 176 pontos na vertical (eixo Y). Portanto, em computação gráfica neste micro, o eixo Y tem sentido inverso à maneira de formatar comandos de saída de textos.

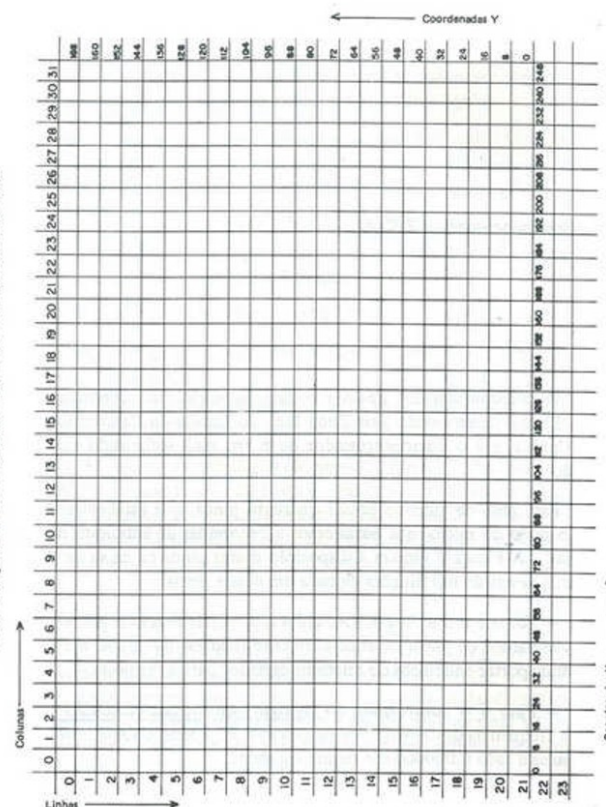
A seguir, uma tabela de orientação das posições gráficas para desenho na tela, bem como das posições de impressão de textos, que ajudará você na hora de converter gráficos em textos (coordenados), e vice-versa.

Por exemplo, você desenha, através de PLOT e DRAW, um retângulo na tela, e quer escrever alguma palavra bem no centro desse retângulo. Se os lados desse retângulo forem X e Y, teremos:

```
PRINT AT (175 - Y)/8; X/8; "..."
```

Note que as linhas (vertical) estão associadas ao eixo Y, e as colunas (horizontal) estão associadas ao eixo X. Veja também a inversão de valores na vertical.

DISPOSIÇÃO DOS ELEMENTOS DO VÍDEO DO TK 90X



O microprocessador Z80A

O microprocessador Z80A é o chip de silício mais importante do TK 90X. Ele foi desenvolvido pela Zilog Inc., do Estado da Califórnia, nos Estados Unidos, e é o microprocessador de 8 bits mais sofisticado existente no mundo.

Esse chip de silicone possui quarenta pinos, que estabelecem contato com o resto do micro, que passaremos a denominar de simplesmente "pinos ou vias". A Figura 1 mostra a disposição desses pinos na caixa do Z80, e, a seguir, a descrição das funções de cada um desses pinos:

— **Pinos 1 a 5 e 30 a 40 (A0 até A15)** Estes dezesseis pinos formam o que chamamos de pinos de endereçamento (*address bus*), que são utilizados para transportar endereços do microprocessador para a memória.

— **Pino 6 — controlador do "relógio" de entrada (CLOCK)** No TK 90X a frequência deste relógio é de cerca de 3,75 Mhz, ou seja, um "relógio" que pulsa a cada 0,000000306 de um segundo.

— **Pinos 7 a 10 e 12 a 15 (D0 até D7)** São oito pinos que formam as vias de dados (*data bus*), que manipulam bytes de dados de e para o microprocessador.

— **Pino 11 — pino de tensão (+ 5V)** Estabilizado em + 5 volts absolutos, requeridos pelo microprocessador.

52

— **Pino 16 — pino de "interrupções mascaradas" \overline{INT} (INTERRUPT REQUEST)** No TK 90X a rotina de leitura do teclado, ou seja, a rotina da memória ROM que reconhece se alguma tecla foi pressionada, é considerada uma interrupção mascarada. Isto significa que o hardware da máquina contém um relógio que ativa o pino INT a cada 1/60 seg. (frequência local), fazendo com que o microprocessador interrompa a sua atividade normal, para executar essa rotina de leitura de teclado. Essas interrupções podem ser controladas pelo programador com instruções especiais em código de máquina.

— **Pino 17 — pino de "interrupção não mascarada", NMI (NON MASKABLE INTERRUPT)** Quando este pino é ativado, ele faz com que o microprocessador pare a execução de um programa em código de máquina.

— **Pino 18 — pino \overline{HALT} , \overline{HALT}** Quando este pino está no nível 0, indica que o microprocessador está executando a instrução HALT, ou seja, entra em "estado de espera", aguardando alguma instrução.

Esta instrução é usada basicamente em dois casos:

1- No final de um programa, após todas as instruções em código de máquina terem sido executadas.

2- Quando é necessário permanecer com o Z80 parado, aguardando uma instrução ou uma interrupção.

— **Pino 19 — pino de solicitação de memória, \overline{MREQ}** Este pino é uma saída do Z80, que, quando está no nível 0, indica que existe um endereço de memória a ser utilizado nos pinos de endereçamento, para leitura ou para escrita da/na memória. O \overline{MREQ} faz parte da seleção do chip de memória, pois informa ao meio externo que o Z80 está realizando uma leitura ou escrita na memória.

— **Pino 20 — pino de entrada/saída — \overline{IORQ} (INPUT/OUTPUT REQUEST)** Este sinal indica que existe, na metade inferior dos pinos de endereçamento (bits A0 até A7 — menos significativos), um endereço válido de entrada/saída (input/output), para uma operação de entrada/saída, de leitura ou escrita na memória. Um sinal \overline{IORQ} também é gerado com um sinal \overline{MI} quando uma interrupção é reconhecida, para indicar que uma resposta de vetor de interrupção pode ser colocada nas vias de dados.

— **Pino 21 — p de leitura, \overline{RD} , (memory read)** Quando este pino está no nível 0 indica que o microprocessador quer ler dados da memória ou de algum periférico de entrada/saída.

— **Pino 22 — pino de escrita, \overline{WR} (memory write)** Quando está no nível 0, indica que existe nas vias de dados D0 a D7, um byte para ser armazenado no endereço da memória ou no periférico de entrada/saída.

– **Pino 23 – pino de reconhecimento – $\overline{BUSA\overline{K}}$ (bus acknowledge)** O microprocessador reconhece uma “requisição externa”, interrompendo a execução de qualquer instrução e ativando este pino.

– **Pino 24 – pino de espera – \overline{WAIT}** Este é um sinal de pedido de espera, com o objetivo de sincronizar memórias e periféricos de entrada/saída mais lentos que o Z80. Enquanto este sinal de entrada WAIT for mantido no nível 0, o microprocessador fica parado aguardando que o meio externo responda à sua solicitação de leitura ou escrita.

– **Pino 25 – pino de solicitação – \overline{BUSRQ} (bus request)** O Z80 permite que periféricos externos usem os pinos de endereçamento ou os pinos de dados, através da ativação deste pino.

– **Pino 26 – pino de inicialização – \overline{RESET}** Este pino é uma entrada usada para inicializar o Z80. Ele é acionado imediatamente após ligar o TK 90X, ou quando se deseja inicializar o Z80. Quando o \overline{RESET} vai para o nível 0, ocorre o seguinte:

- 1- os pinos de endereçamento e de dados são inicializados;
- 2- todos os sinais de controle ficam inativos;
- 3- os registros R e I ficam com valor 0;
- 4- o modo de interrupção é colocado em 0;
- 5- as interrupções provenientes da entrada INT são inibidas; e
- 6- o contador de programa é zerado.

– **Pino 27 – pino de “busca” da memória – $\overline{M\overline{I}}$ (machine cycle one)** Quando vai para o nível 0, indica que está sendo executado um *fetch* (ciclo de busca de instrução), da instrução corrente. Toda instrução, ao ser executada, exige que o Z80 primeiramente realize a busca do seu código de operação (OP CODE) que está armazenado na memória. Em seguida, o Z80 deposita este código no registro de instrução, para, então, interpretá-lo.

– **Pino 28 – pino de “restauração” da memória – (refresh) \overline{RFSH}** Não está diretamente relacionado com leitura ou escrita na memória. É utilizado em memórias RAM dinâmicas, como um seu refrescamento (restauração).

Basicamente, o \overline{RFSH} é uma operação de leitura em determinadas posições de memória, sem que haja efetivamente transferência de informações.

O Z80, através das vias de endereçamento (*address bus*), do registro R e do sinal \overline{RFSH} , implementa as funções de \overline{RFSH} , sem necessidade de controladores externos.

Quando $\overline{RFSH} = 0$ e $\overline{MREQ} = 0$, o conteúdo do registro R é colocado nos 7 bits menos significativos (A0 até A6) das vias de endereçamento, e a cada busca de instrução (*fetch*), o conteúdo do registro R é incrementado de uma unidade.

– **Pino 29 – pino terra.**

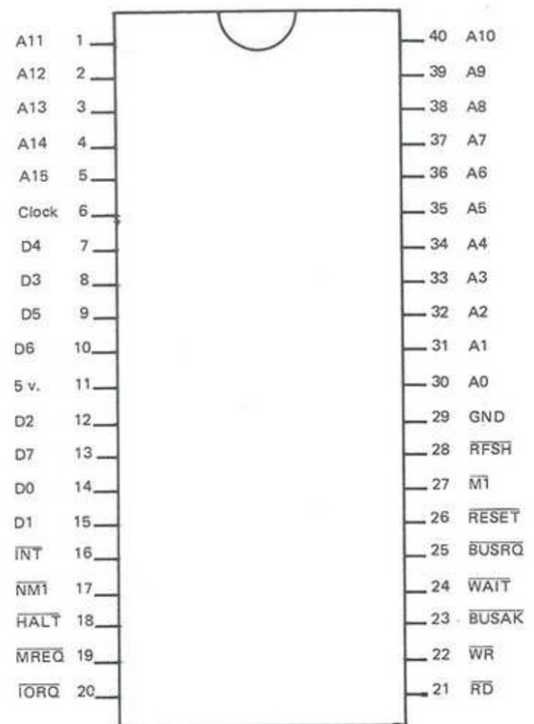


Figura 1 – Pinagem do microprocessador Z80

A ESTRUTURA INTERNA DO Z80

A estrutura interna desse chip é um tanto complicada, mas felizmente dividida em cinco partes que exercem funções diferentes. São elas:

- 1- Unidade de controle
- 2- Registro de instruções
- 3- Contador de programa
- 4- Registros disponíveis ao usuário
- 5- Unidade lógica e aritmética

A figura abaixo mostra esquematicamente a disposição dessas partes dentro do chip.



Partes funcionais do chip Z80

A seguir, a descrição dessas cinco partes:

UNIDADE DE CONTROLE

A unidade de controle do Z80 pode ser comparada a um "gerente de uma linha de produção de uma fábrica". É responsabilidade da unidade de controle a aquisição de matérias-primas (bytes de dados), que são transformadas pela fábrica (estrutura do Z80) em produtos finais acabados (também bytes de dados), que são enviados aos destinatários finais, garantindo assim sucesso na produção.

Essa unidade de controle produz um número muito grande de sinais de controle internos, que, através das vias de controle, vão para outras partes da estrutura interna do microprocessador, assim como esses sinais de controle vão para os pinos de controle RD, WR, MREQ etc.

Note, porém, que, da mesma maneira como um gerente de uma linha de produção, esta unidade de controle não é responsável sobre qual tarefa deva ser realizada, mas apenas sobre como fazer.

O Z80 tem condição de funcionar como computador porque tem a habilidade de seguir um programa armazenado. Este programa deve estar presente em algum lugar da memória, de modo que ele possa ler as instruções em código de máquina, uma a uma, para então executá-las.

REGISTRO DE INSTRUÇÕES

O termo "registro" é usado para descrever um dispositivo interno do Z80 que guarda temporariamente 8 bits de um byte qualquer, para poder manipulá-los.

Nos circuitos internos deste microprocessador existem vários registros, e o movimento de bytes entre esses registros é um dos recursos mais importantes em programação em linguagem de máquina.

O registro de instruções é um registro muito especial, que armazena o código de operação de uma instrução durante todo o tempo em que esta esteja sendo executada. Esse código de operação de instrução é quem determina o que deve ser feito pelo sistema durante uma instrução.

CONTADOR DE PROGRAMA

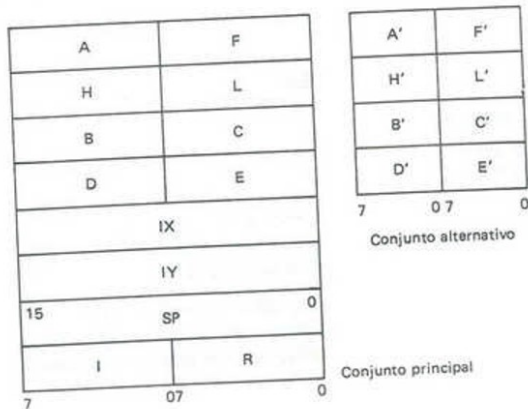
Este contador de programa não é um registro simples de 8 bits, mas sim a junção de dois registros de 8 bits, formando um par de registros, totalizando 16 bits, para serem usados juntos, no sentido de manter controle de endereçamentos de cada instrução guardada na memória. Sempre que uma instrução for lida na memória, a fim de ser executada, junto com ela deve ser fornecido um endereço.

O PC (Program Counter) terá então o endereçamento dessa instrução, sendo normalmente incrementado para, após tê-la executado, apontar para a instrução seguinte.

REGISTROS DISPONÍVEIS AO USUÁRIO (REGISTROS PRINCIPAIS)

Existem, no total, 24 registros disponíveis ao usuário no microprocessador Z80. Eles são assim denominados porque podem receber e armazenar bytes de dados especificados pelo usuário. Cada registro tem seu próprio nome, que não possui lógica na maioria dos casos, e alguns deles possuem funções específicas. São registros de 1 byte, ou seja, 8 bits (daí o microprocessador ser de 8 bits – manipular 1 byte por vez), que em determinadas situações são agrupados aos pares, formando um “par de registros”, capaz de manipular, então, 16 bits.

O esquema abaixo mostra os registros do Z80.



A seguir a descrição dos registros disponíveis no Z80, com as suas respectivas funções:

REGISTRO A

Este é o mais importante registro do Z80. Ele é mais conhecido por “acumulador”, porque na maioria das operações que dele se utilizam, usam-no para acumular seus resultados.

Ele é muito usado para desenvolver operações aritméticas e lógicas, e muitas delas se utilizam apenas deste registro para atingir o resultado.

Por conseguinte, existem diversos modos pelos quais um byte de dados pode ser armazenado pelo programador neste registro. Portanto existem muitas instruções em código de máquina que envolvem o registro A.

REGISTRO F

Também conhecido como *flag register*, ou seja, em tradução literal, “registros de bandeiras”, cujo significado é “registros de indicadores de estado”.

Normalmente ele é mais conhecido como uma coleção de 8 bits indicadores de algum estado específico de microprocessador, em vez de um registro propriamente dito.

O conceito destes bits indicadores de estado será visto com mais detalhes mais tarde, mas por enquanto basta saber que determinados indicadores querem dizer algo, quando estão com valor 1, e querem dizer outra coisa completamente diferente quando estão com valor 0.

REGISTROS H E L, FORMANDO O PAR HL

Normalmente, as instruções que endereçam bytes de dados na memória, o fazem através do par de registros HL, tornando então possível o endereçamento de até 65.536 posições de memória. Lembremos que um endereço de memória sempre se utiliza de 16 bits (2 bytes), e é subdividido em “parte alta” e “parte baixa”, dando origem aos nomes desses registros H (de HIGH) e L (de LOW), significando que a parte alta do endereço será armazenada no registro H e a parte baixa no registro L. Por exemplo, o endereço 7682h, para ser armazenado no par HL, é subdividido em 76, que vai para o registro H, e 82, que vai para o outro registro.

Uma memória de 65.536 posições de endereçamento pode ser considerada como sendo dividida em 256 páginas de 256 posições, e, neste caso, o valor armazenado no registro H serve como indicador de qual página da memória está sendo utilizada.

No microprocessador Z80, o par de registros HL é um dos três pares empregados no endereçamento de registros, porém é o mais importante. Ele pode ser usado para armazenar um número de 16 bits em vez de endereços, pois existe um grande número de operações aritméticas que podem ser realizadas com esses números.

REGISTROS B, C, D, E, OU PARES BC E DE

Estes pares de registros são usados principalmente como registros de endereçamento, para auxiliar o par HL, ou utilizados individualmente, como registros comuns, embora o nome DE seja abreviação de "DESTINATION" (destino), e o registro B, quando utilizado individualmente serve como contador de loop, na maioria das vezes.

CONJUNTO DE REGISTROS ALTERNATIVOS

O conjunto de registros existentes, A, B, C, D, E, F, H e L podem não ser suficientes para o programador em código de máquina. Para isso existem os registros alternativos, A', B', C', D', E', F', H' e L', que, através de instruções em código de máquina especiais, permitem que o conteúdo dos registros principais seja momentaneamente trocado com o conteúdo do seu equivalente alternativo.

Assim, os dados anteriores ficam guardados nos registros alternativos, enquanto se trabalha com os registros principais. A cada troca, todos os registros são envolvidos, com exceção dos registros A e F.

PARES DE REGISTROS IX E IY

Estes dois pares de registros são usados em operações que envolvem indexação, que é uma facilidade que permite manipulação de itens de listagens ou tabelas, a fim de serem pesquisadas. Estes registros mantêm um endereço base, e as posições desejadas são sempre em função desse endereço base, que deve estar obrigatoriamente armazenado no par IX ou no par IY.

No programa monitor de 16 K do TK 90X, o par de registros IY é normalmente utilizado para apontar o endereço 23.610, ou 5C3Ah, endereço este que é considerado como base da tabela de variáveis do sistema.

O par IX é utilizado como apontador das rotinas dos comandos Basic, LOAD, SAVE, VERIFY e MERGE.

Por isso, não é conveniente mexer no par IY, e muito cuidado ao utilizar o par IX.

REGISTRO APONTADOR DA PILHA SP (STACK POINTER)

Este registro ainda é um registro de endereçamento. Ele é usado para apontar posições na área da pilha da máquina na memória RAM, e é sempre considerado como sendo um registro simples, porém de 2 bytes.

Essa pilha da máquina (*machine stack*) é utilizada para se "empilhar" endereços, da seguinte maneira: último a ser colocado, primeiro a ser retirado (LIFO = Last In, First Out), e o apontador da pilha (SP) é usado para armazenar o endereço da última posição a ser executada. Entretanto, quando uma nova entrada está para ser efetivada, a Unidade de Controle do microprocessador reduz o valor armazenado no ponteiro da pilha antes de aceitar essa entrada.

REGISTRO I

Este é o registro vetor de interrupção. Em outros sistemas baseados no Z80, que não o TK 90X, este registro normalmente é usado para armazenar o endereço base de uma tabela de endereços para manipulação de diferentes dispositivos de entrada/saída. Entretanto, esta facilidade não é utilizada no TK 90X, e este registro é usado para auxiliar na geração de sinais para a televisão.

REGISTRO R

Este é o registro de refrescamento de memórias. Na realidade, ele é um simples contador que é incrementado a cada ciclo de busca de instrução. O valor no registro se altera diversas vezes entre valores de 0 a 255.

O registro R é usado para gerar parte do endereço requerido para memórias dinâmicas, de forma que possa ser "refrescado" (recarregado).

A UNIDADE LÓGICA E ARITMÉTICA (ULA)

Este é o quinto bloco funcional do microprocessador Z80 tendo como função específica o procedimento de operações lógicas e aritméticas, de propósitos bem reduzidos; em aritmética apenas operações de soma e subtração binárias são possíveis, e sempre de um byte contra outro byte.

O usuário deverá programar rotinas em linguagem de máquinas repetitivas, caso deseje trabalhar com campos maiores que um byte. Por esse motivo, após cada soma ou subtração de um byte, um indicador do registro F, aquele chamado de CARRY FLAG, ou bandeira de transporte, poderá ser ativado (conter valor 1), simulando aquela nossa conhecida operação de “vai um”, na operação aritmética dos próximos dois bytes.

Esta unidade também é capaz de desenvolver operações com bits, operações lógicas (AND, NOT, OR ou XOR — que veremos adiante) e operações de ativar ou desativar indicadores de estados (*flags*), a fim de mostrar determinados resultados.

Estrutura de um programa em código de máquina

Conforme já vimos, o microprocessador Z80 trabalha como um computador, já que é uma pequena máquina capaz de seguir instruções de um programa armazenado. Este programa obrigatoriamente sempre é um conjunto de instruções em código de máquina, associado a alguns dados, ou bytes, ou posições da memória; tanto ROM como RAM, ou informações genéricas que o programa necessita, sempre armazenadas em posições consecutivas na memória.

Em microcomputadores baseados no microprocessador Z80, estas posições da memória armazenam no máximo 8 bits, ou 1 byte de dado. Portanto, um programa em código de máquina consiste em um conjunto de dados que aparecem como uma série de números de 8 bits.

A descrição mais elementar de um programa em código de máquina é a sua representação binária. Por exemplo:

<i>Decimal</i>	<i>Binário</i>
Posição 0	11110011
Posição 1	10101111
Posição 2	00010001
Posição 3	11111111
Posição 4	11111111
Posição 5	11000011
Posição 6	11001011
Posição 7	00010001

Essa representação é uma maneira perfeitamente válida para mostrar um programa em código de máquina. Mas você há de convir que é difícilimo para nós programarmos assim, e é também muito sujeito a erros. (Imagine, dada essa estrutura de programa, se você trocar apenas um bit de um byte qualquer...)

A seguir, o mesmo programa em código de máquina, só que com outra representação, expressa em decimal e em hexadecimal. Mas, que é igualmente difícil e não muito útil, dado que não sabemos o seu significado.

<i>Decimal</i>	<i>Hex</i>	<i>Decimal</i>	<i>Hex</i>
Posição 0	0000	243	F3
Posição 1	0001	175	AF
Posição 2	0002	17	11
Posição 3	0003	255	FF
Posição 4	0004	255	FF
Posição 5	0005	195	C3
Posição 6	0006	203	CB
Posição 7	0007	17	11

Experimente digitar o programa Basic a seguir, que mostra em valores hexadecimal, os conteúdos das sucessivas posições da ROM.

[illegible]

E o resultado, para as 21 primeiras posições:

[illegible]

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100 101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118 119 120 121 122 123 124 125 126 127 128 129 130 131 132 133 134 135 136 137 138 139 140 141 142 143 144 145 146 147 148 149 150 151 152 153 154 155 156 157 158 159 160 161 162 163 164 165 166 167 168 169 170 171 172 173 174 175 176 177 178 179 180 181 182 183 184 185 186 187 188 189 190 191 192 193 194 195 196 197 198 199 200 201 202 203 204 205 206 207 208 209 210 211 212 213 214 215 216 217 218 219 220 221 222 223 224 225 226 227 228 229 230 231 232 233 234 235 236 237 238 239 240 241 242 243 244 245 246 247 248 249 250 251 252 253 254 255 256 257 258 259 260 261 262 263 264 265 266 267 268 269 270 271 272 273 274 275 276 277 278 279 280 281 282 283 284 285 286 287 288 289 290 291 292 293 294 295 296 297 298 299 300 301 302 303 304 305 306 307 308 309 310 311 312 313 314 315 316 317 318 319 320 321 322 323 324 325 326 327 328 329 330 331 332 333 334 335 336 337 338 339 340 341 342 343 344 345 346 347 348 349 350 351 352 353 354 355 356 357 358 359 360 361 362 363 364 365 366 367 368 369 370 371 372 373 374 375 376 377 378 379 380 381 382 383 384 385 386 387 388 389 390 391 392 393 394 395 396 397 398 399 400 401 402 403 404 405 406 407 408 409 410 411 412 413 414 415 416 417 418 419 420 421 422 423 424 425 426 427 428 429 430 431 432 433 434 435 436 437 438 439 440 441 442 443 444 445 446 447 448 449 450 451 452 453 454 455 456 457 458 459 460 461 462 463 464 465 466 467 468 469 470 471 472 473 474 475 476 477 478 479 480 481 482 483 484 485 486 487 488 489 490 491 492 493 494 495 496 497 498 499 500 501 502 503 504 505 506 507 508 509 510 511 512 513 514 515 516 517 518 519 520 521 522 523 524 525 526 527 528 529 530 531 532 533 534 535 536 537 538 539 540 541 542 543 544 545 546 547 548 549 550 551 552 553 554 555 556 557 558 559 560 561 562 563 564 565 566 567 568 569 570 571 572 573 574 575 576 577 578 579 580 581 582 583 584 585 586 587 588 589 590 591 592 593 594 595 596 597 598 599 600 601 602 603 604 605 606 607 608 609 610 611 612 613 614 615 616 617 618 619 620 621 622 623 624 625 626 627 628 629 630 631 632 633 634 635 636 637 638 639 640 641 642 643 644 645 646 647 648 649 650 651 652 653 654 655 656 657 658 659 660 661 662 663 664 665 666 667 668 669 670 671 672 673 674 675 676 677 678 679 680 681 682 683 684 685 686 687 688 689 690 691 692 693 694 695 696 697 698 699 700 701 702 703 704 705 706 707 708 709 710 711 712 713 714 715 716 717 718 719 720 721 722 723 724 725 726 727 728 729 730 731 732 733 734 735 736 737 738 739 740 741 742 743 744 745 746 747 748 749 750 751 752 753 754 755 756 757 758 759 760 761 762 763 764 765 766 767 768 769 770 771 772 773 774 775 776 777 778 779 780 781 782 783 784 785 786 787 788 789 790 791 792 793 794 795 796 797 798 799 800 801 802 803 804 805 806 807 808 809 810 811 812 813 814 815 816 817 818 819 820 821 822 823 824 825 826 827 828 829 830 831 832 833 834 835 836 837 838 839 840 841 842 843 844 845 846 847 848 849 850 851 852 853 854 855 856 857 858 859 860 861 862 863 864 865 866 867 868 869 870 871 872 873 874 875 876 877 878 879 880 881 882 883 884 885 886 887 888 889 890 891 892 893 894 895 896 897 898 899 900 901 902 903 904 905 906 907 908 909 910 911 912 913 914 915 916 917 918 919 920 921 922 923 924 925 926 927 928 929 930 931 932 933 934 935 936 937 938 939 940 941 942 943 944 945 946 947 948 949 950 951 952 953 954 955 956 957 958 959 960 961 962 963 964 965 966 967 968 969 970 971 972 973 974 975 976 977 978 979 980 981 982 983 984 985 986 987 988 989 990 991 992 993 994 995 996 997 998 999 1000 1001 1002 1003 1004 1005 1006 1007 1008 1009 1010 1011 1012 1013 1014 1015 1016 1017 1018 1019 1020 1021 1022 1023 1024 1025 1026 1027 1028 1029 1030 1031 1032 1033 1034 1035 1036 1037 1038 1039 104

Altere a linha 40 do programa, de forma que também seja mostrado o equivalente em decimal dos bytes da ROM.

Este programa foi incluído aqui para mostrar como usualmente se produz uma listagem com valores decimais ou hexadecimais de um programa em código de máquina.

Mas, o que querem dizer esses valores? Aparentemente não têm significado nenhum. Vamos ver isso:

Na tabela do final deste livro, que mostra todas as instruções do Z80, com seus respectivos códigos, ou mesmo no manual do TK 90X, você pode ver o que significam esses códigos:

Decimal	Mnemônica	Comentário
Posição 0	DI	Não permite interrupções
Posição 1	XOR A	Efetua a operação lógica XOR A com o acumulador (registro A).
Posição 2 - 4	LD DE, FFFF	Carrega o par de registros DE com o valor constante FFFFh.
Posição 5 - 7	JP 11CB	Efetua um salto relativo para o endereço CBI1

Nessa descrição, "mnemônica" quer dizer o nome das instruções em código de máquina, que é uma maneira estilizada de representar a instrução, de forma que seja facilmente compreendida.

Todas as instruções em código de máquina do conjunto de instruções do Z80 possuem suas próprias mnemônicas, e normalmente um programa em código de máquina é descrito utilizando-se delas em vez de valores binários, decimais ou hexadecimais.

Note na listagem acima, que duas linhas de instrução usam posições simples enquanto outras duas linhas ocupam três posições. No último caso, a primeira posição armazena o código da instrução propriamente dito, enquanto as outras duas armazenam os dados associados a essas instruções.

A forma usual de se listar um programa em código de máquina é mostrada a seguir:

Endereço	Código de Máquina	Mnemônica	Comentários
0000	F3	DI	Não permite interrupções
0001	AF	XOR A	Operação lógica XOR ou OR exclusivo
0002	11 FF FF	LD DE, FFFF	Endereço de topo da memória, armazenado no par DE
0005	C3 CB 11	JP 11CB	Equivalente ao Basic "GOTO"

Esta forma de listagem normalmente é chamada de formato *assembly* (montado), e possui os endereços das posições, armazenando o primeiro byte da linha da instrução, em hexadecimal; os códigos das instruções e seus dados associados, também em notação hexadecimal; as mnemônicas de cada instrução, e, finalmente, um campo para comentários, onde o usuário pode escrever o que deseja que aconteça naquele momento.

O exemplo acima mostra como o formato *assembly* para um dado bloco de códigos de Z80 pode ser derivado — uma operação que normalmente é denominada *disassembly*, e o programa de computador que desenvolve essas operações é conhecido por *disassembler*. Portanto, programas *disassembler* são muito úteis, que mostram programas em código de máquina, com as posições da memória ou endereços, em hexadecimal, seus conteúdos, também nessa base numérica, e as mnemônicas das instruções correspondentes.

Em outras palavras, um programa *assembler*, ou *montador*, permite-nos desenvolver listagens em código de máquina a partir de suas mnemônicas, para que ele faça a sua conversão para seus códigos equivalentes automaticamente, e os programas chamados de *disassembler*, ou *desmontadores*, permitem que *abramos* listagens em código de máquina, ou seja, mostrarmos-nos listagens de instruções do Z80, seus endereços e respectivos códigos hexadecimais com as mnemônicas.

Às vezes, alguns programadores em código de máquina incluem rótulos (*labels*) em determinadas posições ou endereços da memória, que são constantemente utilizadas durante toda a execução de uma determinada rotina, para facilitar o trabalho de localizar aquela posição.

Escrevendo a rotina acima dessa forma:

INÍCIO = 0000	Endereço/posição 0
NOVO INÍCIO = 11CB	Continua aqui
TOPO MEM = FFFF	Decimal 65.535 – topo da memória

Endereço	Rótulo	Mnemônica	Comentário
0000	INÍCIO	DI	Não permite interrupções
0001		XOR A	Operação XOR A
0002		LD DE, TOPO MEM	Carrega par DE com o topo da memória
0005		JP NOVO INÍCIO	Salta para aquela posição

A seguir, alguns exemplos de listagens, que, apesar de serem completamente diferentes, querem dizer a mesma coisa.

[illegible]

Este é o método mais comum, descrito anteriormente. A primeira coluna da esquerda mostra os endereços, em valores hexadecimais, seguida dos códigos hexa. e as mnemônicas correspondentes.

Outra listagem, mais simples:

[illegible]

Aqui também, a primeira coluna da esquerda significa endereços, em hexadecimal. Compare os códigos da primeira linha com a listagem anterior.

Outro tipo de listagem, abaixo, que mostra, quando possível, os caracteres correspondentes àqueles valores dos códigos hexadecimais.

```
Print 0
0000 0000 0000 0000 0000 0000 0000 0000
0001 0001 0001 0001 0001 0001 0001 0001
0010 0010 0010 0010 0010 0010 0010 0010
0011 0011 0011 0011 0011 0011 0011 0011
0012 0012 0012 0012 0012 0012 0012 0012
0013 0013 0013 0013 0013 0013 0013 0013
0014 0014 0014 0014 0014 0014 0014 0014
0015 0015 0015 0015 0015 0015 0015 0015
```

E aqui, uma curiosidade: o conteúdo dos registros do Z80 e o estado das bandeiras, ou indicadores, que veremos em detalhe mais adiante.

```
READY
Registers
BC DE HL IX IY A
0000 F3B2 5E2E 03D4 5C3A 55
BC DE HL SP (SP) (HL)
0001 0000 595B BF4D 52A0 30
BREAKPOINT : -
FLAGS = SZ-H-PNC
01100001
```

```
READY
Exit
```

Um programa para mostrar bytes de uma forma "tabulada":

```
20 LET a=5000 REM Coloca
30 PRINT "Primeiro endereço"
40 LET b=a
50 FOR i=0 TO 31 STEP 3
60 LET c=INT(PEEK a/16): LET
70 LET d=CHR$(48+h+7*(h>9)):
80 LET e=CHR$(48+l+7*(l>9)): REM
90 LET f=CHR$(48+i+7*(i>9)): REM
100 PRINT TAB 9; a; b; c; d; e; f;
110 LET a=a+1: NEXT i
120 PRINT "Fim da conversão do sistema
130 PRINT "decimal (dos bytes) para o siste
140 PRINT "hexadecimal"
150 PRINT TAB 9; a; b; c; d; e; f;
160 LET a=a+1: NEXT i
170 GO TO 10
```

A matemática na programação em código de máquina

Já vimos que num microcomputador baseado no microprocessador Z80, como o TK 90X, toda e qualquer manipulação de dados em sua memória envolve um byte de 8 bits. A melhor forma de se representar esse dado é a que utiliza a notação binária para cada bit envolvido na operação. Mas, nesse formato, os números são muito difíceis de serem manipulados, e, por esta razão, entre outras de menor importância, é que se usa a notação hexadecimal.

Neste capítulo veremos, por seções, os códigos hexadecimais, aritmética binária absoluta, complemento de dois aritmético, representação integral de um número e representação de um número em ponto flutuante. Os três primeiros tópicos se aplicam em qualquer microcomputador de 8 bits, enquanto os dois últimos são um pouco diferentes no TK 90X.

VALORES OU CÓDIGOS HEXADECIMAIS

O sistema hexadecimal, da mesma forma que um sistema decimal, ou outro sistema qualquer, deve ter uma base para representar valores numéricos. Conforme o próprio nome diz, a base desse sistema é dezesseis, enquanto no sistema decimal, com o qual estamos acostumados a lidar, emprega-se base dez, e no sistema binário emprega-se base dois.

No nosso sistema decimal, possuímos 10 valores distintos para representar quantidades diferentes. Analogamente, o sistema binário utiliza-se de dois valores, quais sejam, o zero e o um, e o sistema hexadecimal possui nada me-

nos que dezesseis valores para representação numérica. Nesta, os primeiros dez caracteres são os dígitos 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 e os seis caracteres adicionais são as letras A, B, C, D, E e F.

Mas, o que representa essa base decimal? Significa que todo e qualquer valor numérico pode ser representado através dos caracteres que compõem aquela base.

Por exemplo: na base decimal, o que significa 1985? Vamos decompor essa quantidade numérica:

$$\begin{array}{r}
 1985 \\
 \downarrow \\
 5 \times 10^0 = 5 \times 1 = 5 \\
 \downarrow \\
 8 \times 10^1 = 8 \times 10 = 80 \\
 \downarrow \\
 9 \times 10^2 = 9 \times 100 = 900 \\
 \downarrow \\
 1 \times 10^3 = 1 \times 1000 = 1000 \\
 \hline
 1985
 \end{array}$$

Ou, na base binária:

$$\begin{array}{r}
 111011110011 \\
 \downarrow \\
 1 \times 2^0 = 1 \times 1 = 1 \\
 \downarrow \\
 1 \times 2^1 = 1 \times 2 = 2 \\
 \downarrow \\
 0 \times 2^2 = 0 \times 4 = 0 \\
 \downarrow \\
 0 \times 2^3 = 0 \times 8 = 0 \\
 \downarrow \\
 1 \times 2^4 = 1 \times 16 = 16 \\
 \downarrow \\
 1 \times 2^5 = 1 \times 32 = 32 \\
 \downarrow \\
 1 \times 2^6 = 1 \times 64 = 64 \\
 \downarrow \\
 1 \times 2^7 = 1 \times 128 = 128 \\
 \downarrow \\
 0 \times 2^8 = 0 \times 256 = 0 \\
 \downarrow \\
 1 \times 2^9 = 1 \times 512 = 512 \\
 \downarrow \\
 1 \times 2^{10} = 1 \times 1024 = 1024 \\
 \downarrow \\
 1 \times 2^{11} = 1 \times 2048 = 2048 \\
 \hline
 3827
 \end{array}$$

E na base hexadecimal:

$$\begin{array}{r}
 FA8B \\
 \downarrow \\
 B \times 16^0 = 11 \times 1 = 11 \\
 \downarrow \\
 8 \times 16^1 = 8 \times 16 = 128 \\
 \downarrow \\
 A \times 16^2 = 10 \times 256 = 2560 \\
 \downarrow \\
 F \times 16^3 = 15 \times 4096 = 61440 \\
 \hline
 61440
 \end{array}$$

Aí estão, portanto, dois exemplos de conversão, respectivamente de base binária e hexadecimal para decimal.

A tabela abaixo mostra a representação decimal, binária e hexadecimal para números de 0 a 15.

Decimal	Binário	Hexadecimal
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
10	1010	A
11	1011	B
12	1100	C
13	1101	D
14	1110	E
15	1111	F

Como você pôde notar na tabela anterior, cada caractere hexadecimal forma uma representação binária de 4 bits. Isso significa que um byte de 8 bits é representado por um par de caracteres hexadecimais, e um número de 16 bits necessita de quatro caracteres hexadecimais.

Por exemplo:

```

0000 0000b = 00h
0100 1111b = 4Fh
0000 0000 0000 0000b = 0000h
0100 1100 1010 1111b = 4CAFh

```

O programa abaixo faz a conversão automática de números na base decimal para números na base hexadecimal:

```

1 REM Base decimal para hexad
2:1000
3 BORDER 0: PAPER 0: INK 0: C
4:10 INPUT "Digite um número men
5:QUE SEUSSEJA
6:20 IF >65535 THEN GO TO 10

```

```

30 PRINT d;" Dec. ";
40 DIM h(4): DIM h$(4)
50 LET h(1)=INT (d/4096): LET
d=d-h(1)*4096
60 LET h(2)=INT (d/256): LET d=
d-h(2)*256
70 LET h(3)=INT (d/16): LET d=
d-h(3)*16
80 LET h(4)=d
90 FOR a=1 TO 4
100 LET h$(a)=CHR$(h(a)+48+7*(
h(a)>9))
110 NEXT a
120 PRINT INK 5; BRIGHT 1;h$; I
NK 6; BRIGHT 0;" hexa "
130 GO TO 10

```

Repare nas linhas 50 até 80, as divisões sucessivas por, respectivamente, 16^3 , 16^2 e 16^1 , e o aproveitamento do resto dessas divisões.

Nas linhas 90 a 110 são feitas as conversões para caracteres ASCII.

Programa que converte números hexadecimais em decimais

```

1 REM Base hexadecimal para d
ecimal
2 POKE 23558,8
3 BORDER 0: PAPER 0: INK 6: C
LS
10 DIM h$(4)
20 INPUT "Digite valor hexadec
imal ";h$
30 IF CODE h$=32 THEN GO TO 20
40 LET d=0
50 FOR a=1 TO 4
60 IF h$(a)=CHR$(32) THEN GO TO
100
70 IF h$(a)<"0" OR h$(a)>"9" A
ND h$(a)<"A" OR h$(a)>"F" THEN G
O TO 120
80 LET d=d+16*(4-a)+(CODE h$(a
)-48+7*(CODE h$(a)>57))
90 NEXT a
100 PRINT "Hexadecimal ";TAB 15
h$;
110 PRINT "Decimal ";TAB 15;d
120 GO TO 10

```

Portanto, para se converter um número na base decimal, para a base hexadecimal, divide-se esse número pelo valor correspondente à potência de 16,

cujo expoente corresponde àquela posição do bit, e converte-se o resultado para o equivalente hexadecimal, até o final. Por exemplo, para se converter 65535d em hexa:

```

1- 65535 ÷ 16 = 4095,9375
2- 65535 | 4096
   24575  15
     4095
Resultado = 15 decimal ou F hexa

```

```

3- 15 x 4096 = 61440
4- 65535 - 61440 = 4095
5- 4095 ÷ 16 = 255,9375
6- 4095 | 256
   1535  15
     255
Resultado = 15 decimal ou F hexa

```

```

7- 15 x 256 = 3840
8- 4095 - 3840 = 255
9- 255 ÷ 16 = 15,9375
10- 255 | 16
    95  15
     15
Resultado = 15 decimal ou F hexa

```

11- Resto final = 15 decimal ou F hexa
Portanto o número 65535 d corresponde à FFFF h.

De posse de uma calculadora, podemos simplificar o processo. Por exemplo, vamos converter 40000d para hexa:

```

40000 | 4096
 36384  9,76 ..... equivale a 9 hexa
  3136 | 256
   3072 12,25 ..... equivale a C hexa
    64  16
     0   4 ..... equivale a 4 hexa
Resto final ..... 0

```

Portanto, 40000d = 9C40h

Se você pretende se aprofundar nos estudos da linguagem de máquina, aconselho-o a se exercitar, e muito, nessas conversões, principalmente entre base decimal e base hexadecimal.

ARITMÉTICA BINÁRIA

Agora vamos ver as regras para operações aritméticas binárias.

I – ADIÇÃO

$0 + 0 = 0$
 $0 + 1 = 1$
 $1 + 0 = 1$
 $1 + 1 = 0$ com transporte de 1
 $1 + 1 + 1 =$ com transporte de 1

Por exemplo:

8	1000	+
+ 7	0111	
15	1111	

Se a soma exceder o valor decimal 15, o resultado obrigatoriamente será expresso em cinco bits.

Por exemplo:

9	1001	+
+ 8	1000	
17	10001	

Se você pretende se aprofundar em código de máquina, meus parabéns! Mas, antes assimile perfeitamente essas regras iniciais de operações aritméticas básicas, porque, sem elas, não é possível se fazer nada, visto que teremos instruções em linguagem de máquina que manipulam apenas um bit de um byte. E se não soubermos o resultado...

II – SUBTRAÇÃO

$0 - 0 = 0$
 $1 - 0 = 1$
 $1 - 1 = 0$
 $0 - 1 = 1$ com transporte de -1
 $1 - 1 - 1 = 1$ com transporte de -1

Por exemplo:

14	1110	-
-3	0011	
11	1011	

Numa subtração, o resultado tanto pode ser positivo como negativo. A fim de distinguir o sinal de um número, usa-se o bit mais à esquerda, qual seja o bit mais significativo, como bit de sinal. Normalmente usa-se a convenção que se atribui o valor 1 ao sinal negativo e 0 ao sinal positivo. Consequentemente, o microprocessador Z80 que possui 8 bits de dados, utiliza 7 deles para armazenar o dado propriamente dito e o último para indicar o seu sinal. Muito cuidado deve ser tomado na expressão desses valores.

São necessários complexos circuitos eletrônicos para se subtrair diretamente os valores binários, e nos microcomputadores é usual realizar estas subtrações somando-se o "complemento de 2" do diminuidor ao diminuendo. Isto significa que se torna possível dispensar o circuito que executa a subtração, realizando essa operação através de circuitos de soma, inclusive quando se determina a forma complementar do diminuidor.

O complemento de 2 de um número binário é determinado invertendo-se cada um dos seus bits e somando-se 1 ao resultado. Por exemplo, o complemento de 2 de 76 em representação binária é determinado da seguinte maneira:

76 = 01001100

Invertendo-se os bits:

10110011	+
1	
10110100	= -76 em forma de complemento de 2

Para se realizar uma subtração usando-se a complementação de 2 é necessário obter o complemento de 2 do diminuidor, que é em seguida somado ao diminuendo. Como exemplo, consideremos 44 subtraído de 15.

15 = 00001111 e 44 = 00101100
 O complemento de 2 de 44 é -44 = 11010011

Então:

00001111	+
11010011	

11100011 = -29, na forma de complemento de 2, isto é, trata-se do complemento de 2 de +29. Nesse caso, o bit de sinal indica um número positivo que se encontra na forma complementar de 2.

III – MULTIPLICAÇÃO

Um método muito comum de realizar a multiplicação binária utiliza o princípio do deslocamento e soma dos algarismos. O multiplicando é multiplicado por cada bit do multiplicador, bit a bit, e o produto resultante obtém-se somando todos os produtos parciais convenientemente deslocados. Como os bits do multiplicador são 0 ou 1, os termos do produto parcial são iguais a 0 ou ao multiplicando, ou versões deslocadas deste.

Por exemplo, vamos multiplicar 193 por 21. Utilizemos para tanto um acumulador de 16 bits para guardar o resultado, e trabalhem com representações binárias de 8 bits para o multiplicando e o multiplicador.

```

Multiplicando ..... 193 = 11000001
Multiplicador ..... 21 = 00010101
      11000001 .. produtos parciais
      11000001 .. convenientemente
      11000001 .. deslocados
111111010101 .. soma dos termos
  
```

Quando se multiplicam dois números binários, o produto contém um número de bits igual à soma dos bits contidos nos dois valores envolvidos na operação. O número máximo de somas requerido para a multiplicação, usando o método de deslocamento e soma, é igual ao número de bits do multiplicador.

IV – DIVISÃO

A divisão binária é realizada usando-se um método de deslocamento e subtração, ao contrário da multiplicação que vimos acima. Diminui-se repetidamente o divisor do dividendo, depois deste ser convenientemente deslocado, e verifica-se o sinal do resto após cada subtração. Se o sinal do resto é positivo, o valor do quociente é 1, mas se o sinal é negativo, o valor é zero, e o dividendo é reconduzido ao seu valor anterior, somando-se de novo o divisor. Depois de a subtração dar um quociente positivo, ou, depois do tratamento anterior, no caso de ter dado um quociente negativo, o divisor é deslocado de uma posição, para a direita, sendo incluído o bit seguinte e repetindo-se a operação até todos os bits do divisor terem sido usados.

COMPLEMENTO DE DOIS ARITMÉTICO

Como vimos na subtração binária, o conceito de complemento de 2 aritmético é muito simples, mas, quando usado em programas em código de máquina, o resultado pode ser um pouco confuso.

O método permite que o programador considere valores na faixa binária entre 0000 0000 até 1111 1111, equivalentes a 0 até 127 do sistema decimal, e 000 0000 até 1111 1111 como equivalente à faixa decimal de -128 a -1.

O resultado dessa interpretação depende do bit 7 (o mais à esquerda na representação binária), atuando como bit de sinal. Ele valerá 0, em caso de números positivos, e 1, no caso de números negativos.

A tabela a seguir ilustra a aplicação do complemento de 2 aplicado a números de 8 bits.

	Binário	Decimal	Hexadecimal
Números positivos	0111 1111	+127	7F
	0111 1110	+126	7E
	0000 0010	+2	02
	0000 0001	+1	01
	0000 0000	0	00
	1111 1111	-1	FF
Números negativos	1111 1110	-2	FE
	1000 0001	-127	81
	1000 0000	-128	80

Note que é possível estender o conceito de complemento de dois aritmético a números que utilizam 16 bits na sua representação binária, atingindo a faixa decimal de -32.768 até +32.767.

Para se converter um número decimal negativo ao seu respectivo complemento de dois binário, realizam-se os seguintes passos:

- 1- Encontra-se a forma binária do valor absoluto decimal. Por exemplo: -54 equivale a 00110110
- 2- Inverte-se o número binário, isto é, todos os zeros passam a valer 1, e todos os 1 passam a valer 0. Portanto, 00110110 passa a 11001001

- 3- Soma-se 1 a esse valor em aritmética binária

```

11001001
  1
+
-----
11001010

```

- 4- Esta é a forma utilizada, tanto em binário, quanto em hexadecimal. Portanto, o complemento de dois de -54 é 11001010 ou CAh.

Essas operações podem ser realizadas em ordem inversa, quando convertendo-se números complemento de dois para números decimais.

REPRESENTAÇÃO INTEGRAL DE UM NÚMERO

O interpretador Basic do TK 90X utiliza cinco bytes para representar números inteiros entre -65535 e +65535, ao passo que números fracionários ou excedentes a essa faixa são tratados em cinco bytes, mas na forma de ponto flutuante.

Na forma integral, o primeiro byte é sempre zero. O segundo byte armazena zero se o inteiro é positivo e 255 se o inteiro é negativo. Os terceiro e quarto bytes armazenam o valor do número, na forma de complemento de dois de 16 bits, sem sinal, sendo o terceiro byte o menos significativo, e o quarto, o mais significativo. O quinto byte também não é usado e vale sempre zero.

O programa a seguir mostra como são manipulados os valores digitados pelo usuário, em decimal, na sua forma integral. Note as condições para somente números inteiros e contidos no limite dessa faixa, especificados na linha 20 do programa.

```

10 INPUT n
20 IF n <> INT n OR n < -65535 OR n > 65535 THEN GOTO 10
30 PRINT "Número escolhido"; n
40 LET v = PEEK 23627 + 256 * PEEK 23628
50 FOR a = 1 TO 5
60 PRINT a; " . "; TAB 5; PEEK (a + v)
70 NEXT a
80 GOTO 10

```

O programa fornece resultados na forma:

```

- Número escolhido: 0
1. 0
2. 0
3. 0
4. 0
5. 0

```

Portanto, zero é positivo e tem o valor $0 \times 1 + 0 \times 256$.

- Número escolhido: 1516

```

1. 0
2. 0
3. 236
4. 5
5. 0

```

Portanto, 1516 é positivo e tem o valor $236 \times 1 + 5 \times 256$.

- Número escolhido: -1

```

1. 0
2. 255
3. 255
4. 255
5. 0

```

Ou seja, -1 é negativo, e tem a forma em complemento de dois, FFFFh.

REPRESENTAÇÃO EM PONTO FLUTUANTE

A representação de cinco bytes de um número em ponto flutuante permite a manipulação de valores entre .29 E-38 a 1.7 E38.

O valor zero é sempre armazenado com todos os cinco bytes valendo zero. Todos os outros valores são armazenados com o expoente no primeiro byte e a mantissa nos outros quatro bytes.

Encontrar o expoente e a mantissa de um número decimal é, em essência, a conversão de números decimais para seus equivalentes em formato E.

Primeiramente, consideremos o número 1234.5, que pode ser expresso no formato E, como .1234 E+4 (repare que o ponto andou 4 casas para a esquerda). Para obter a mantissa de um número, o ponto decimal é deslocado para a esquerda até o último algarismo significativo. O expoente é o número de deslocamentos requeridos para a mantissa.

Como o TK 90X trabalha com números binários em vez de decimais, as mesmas operações serão agora aplicadas para simples valores binários.

Primeiro, consideremos o valor binário 00001 1111, que equivale a 31 decimal.

O ponto binário é colocado à direita dos 8 bits, e isso toma 5 deslocamentos para que ele fique à frente do bit mais significativo.

Nesta fase, o expoente e a mantissa são:

Exp. = + 5 e a mantissa = .1111 10000...

No TK 90X essas partes são manipuladas de uma maneira um pouco diferente.

Para o expoente O verdadeiro expoente, + 5 acima, é sempre incrementado pelo decimal 128, ou 80h, para fornecer o expoente aumentado. Neste exemplo, +5 + 128 = 133.

Para a mantissa O bit mais significativo da mantissa é sempre 1, e, como é redundante, é substituído pelo bit de sinal, ou seja, quando tratando com números positivos será 0, e quando tratando com números negativos, será sempre 1.

No nosso exemplo, a mantissa é .01111 1000... e agora precisamos expressar o número em cinco bytes. No caso da mantissa, os bytes que não são usados sempre valerão 0.

Portanto, a forma de representação em ponto flutuante do número 31 será 133, 120, 0,0,0; em hexadecimal será 85,78,00, 00,00 e em binário, 1000 0101 0111 1000 0000 0000 0000 0000 0000

As mesmas operações se aplicam a números negativos e fracionários, embora as etapas de conversão não sejam tão fáceis quanto para números inteiros e positivos.

O programa a seguir mostra a forma decimal de números em ponto flutuante:

```
10 INPUT n
20 IF n = 0 THEN GOTO 40
30 LET n = n + .2E -38
40 PRINT "Numero escolhido"; n
50 PRINT
60 PRINT "Exp."; TAB 9; "Mantissa"
70 LET v = PEEK 23627 + 256 * PEEK 23628
80 PRINT PEEK (v + 1); TAB 9;
90 FOR a = 2 TO 5
100 PRINT PEEK (v + a); CHR$ 32;
110 NEXT a
120 PRINT ""
130 GOTO 10
```

Esse programa mostra que para os números seguintes, a forma de ponto flutuante é:

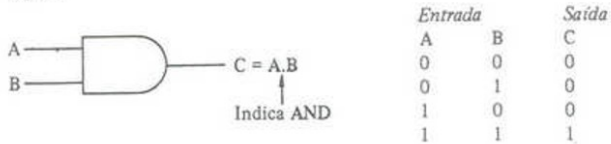
		Exp.	Mantissa
1.	=	129	0 0 0 0
2	=	130	0 0 0 0
35456	=	144	10 128 0 0
-1	=	129	128 0 0 0
-35456	=	144	138 128 0 0
6.333	=	131	74 167 239 158

Operações lógicas

O TK 90X utiliza circuitos eletrônicos lógicos que produzem uma saída que depende do valor de uma ou mais variáveis de entrada. Os valores de entrada e saída encontram-se no nível lógico 0 (0 volts) ou no nível lógico 1 (+ 5 volts).

OPERAÇÃO LÓGICA AND

A porta AND produz uma saída lógica 1 quando todas as entradas se encontram nesse mesmo nível lógico, ou seja, quando todas as entradas também valem 1; em qualquer outro caso, o resultado ou a saída é zero. O símbolo e a tabela verdade da porta AND de duas entradas são apresentadas a seguir:



Por exemplo:

A = 0110 0101
B = 1101 0010

Portanto, A.B resultará:

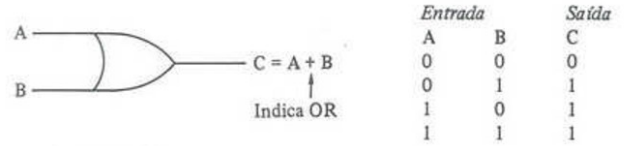
```

0110 0101
1101 0010
-----
0100 0000

```

OPERAÇÃO LÓGICA OR

A porta OR produz uma saída lógica 1 quando pelo menos uma das entradas se encontra no nível lógico 1. Isto significa que a saída só é 0 quando todas as entradas também valem 0.



Por exemplo:

A = 0110 0101
B = 1101 0010

Portanto, A + B resultará:

```

0110 0101
1101 0010
-----
1111 0111

```

OPERAÇÃO LÓGICA NOT

A porta NOT (porta inversora ou porta complementar) produz uma saída inversa (complementar) da entrada.

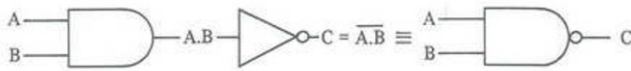


Por exemplo:

A = 0110 0101
 \bar{A} = 1001 1010

OPERAÇÕES LÓGICAS NAND E NOR

Podem-se executar estas operações lógicas usando-se combinações de portas lógicas. A porta *NAND* é obtida combinando-se uma porta AND e uma porta NOT. Isto é mostrado, a seguir, para o caso de um elemento NAND de duas entradas. Esta porta só fornece uma saída lógica 0 quando todas as entradas se encontram no nível lógico 1; em qualquer outro caso o resultado será sempre 1.



Entrada		Saída
A	B	C
0	0	1
0	1	1
1	0	1
1	1	0

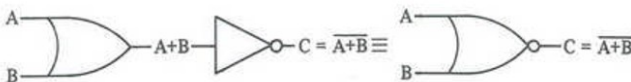
Por exemplo:

A = 0110 0101
B = 1101 0010
 $\overline{A.B} = 1011 1111$

A porta *NOR* é obtida combinando-se uma porta lógica OR e uma porta lógica NOT. Isto é mostrado a seguir, para o caso de um elemento NOR de duas entradas. Esta porta só produz uma saída lógica 1 quando todas as entradas possuem nível lógico 0; em todos os outros casos produz saída 0.

Utilizando-se os valores anteriores para A e B, teremos:

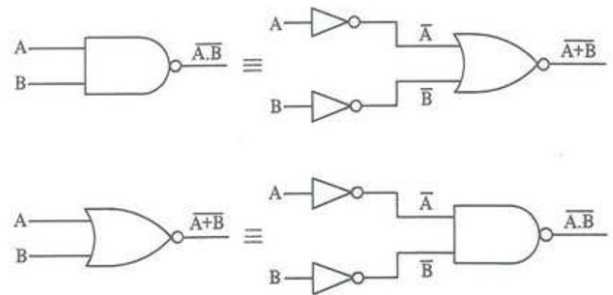
$\overline{A+B} = 0000 1000$



É igualmente interessante notar que as operações lógicas NAND e NOR sobre as variáveis numéricas A e B podem ser substituídas por uma forma equivalente:

NAND $\longrightarrow \overline{A.B} = A+B$
NOR $\longrightarrow \overline{A+B} = A.B$

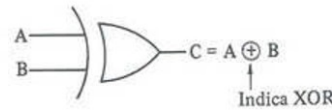
Ou através dos diagramas lógicos:



OPERAÇÃO LÓGICA OR EXCLUSIVO (XOR)

Pode-se executar esta operação utilizando-se um circuito de porta lógico. A porta *OR* exclusiva produz uma saída lógica 1 quando qualquer das entradas, mas não todas, estiver no nível lógico 1. Isto implica que a saída só estará no nível 0 quando todas as entradas tiverem o mesmo valor lógico, seja 0 ou 1.

Entrada		Saída
A	B	C
0	0	0
0	1	1
1	0	1
1	1	0



As instruções em código de máquina do Z80

Essas instruções, que a partir deste capítulo passaremos a discutir e posteriormente aplicar em exemplos práticos, são divididas em 18 grupos, onde cada instrução tem alguma semelhança com as outras. . . Porém, antes de discutí-las, devemos mencionar seis classes de dados que podem completar uma instrução do Z80.

Essas classes são:

1- Um número, na faixa de um byte simples, ou seja, na faixa de 0 até 255, ou 0h até 255h. As instruções que requeiram um byte terão sua indicação seguida de "dd".

Por exemplo, a mnemônica "LD D, dd".

2- Um número de dois bytes, na faixa de 0 a 65535, ou 0h a FFFFh, representado por "dddd", como na mnemônica LD BC, dddd.

3- Um endereço (2 bytes), na mesma faixa numérica que a anterior, porém com a representação junto com as mnemônicas de "end", como, por exemplo, a instrução "JP end".

4- Um deslocamento de um byte, ou seja, um número na faixa idêntica a de um byte, considerado obrigatoriamente em sua forma de complemento de dois aritmético, com representação após a mnemônica de "e", como, por exemplo, na instrução "JR e".

86

5- Um byte para deslocamento indexado, também na faixa numérica de um byte, e aqui nesta classe, também considerado em sua forma de complemento de dois aritmético, com representação nas instruções que requeiram este tipo de deslocamento de "d", como, por exemplo, "LD A, (IX + d)".

6- Um byte para deslocamento indexado e um simples byte na faixa numérica de -128d a +127d, para o primeiro byte, e na faixa numérica de 0d a 255d, para o segundo valor, com as respectivas representações de "d" e "dd", como na instrução "LD (IX + d), dd".

Passemos então aos grupos de instruções do Z80.

GRUPO 1 — AS INSTRUÇÕES DE NÃO-OPERAÇÃO

Mnemônica	Código hexa
NOP	00

Esta instrução NOP, quando executada pelo microprocessador, faz com que este interrompa suas atividades por cerca de 1,14 microssegundos. Nenhum dos registros ou indicadores é afetado por esta instrução.

Esta instrução é muito útil em casos de se querer uma pausa determinada em certas rotinas de linguagem de máquina, ou para cancelar, ou apagar instruções em código de máquina em rotinas já prontas (como aquela de seu programa predileto, que exibe a mensagem de copyright. . .).

GRUPO 2 — INSTRUÇÕES DE CARREGAR REGISTROS COM VALORES NUMÉRICOS

As instruções a seguir são desenvolvidas no sentido de carregar registros com simples bytes, constantes.

Mnemônica	Código hexa
LD A, dd	3Edd
LD H, dd	26dd
LD L, dd	2Edd
LD B, dd	06dd
LD C, dd	0Edd
LD D, dd	16dd
LD E, dd	1Edd

Como se pode notar pelos códigos hexa, estas instruções requerem duas locações da memória, uma para o código da instrução e outra para o valor envolvido. As instruções acima podem ser consideradas análogas às instruções de atribuição em Basic, onde se atribui um valor qualquer a uma variável de nome conhecido. No nosso caso, atribuímos a um registro determinado o valor de um byte especificado, ou, em outros termos, carregamos aquele registro com o conteúdo especificado. Este valor será então armazenado no registro, anulando o anterior.

As instruções a seguir são análogas às anteriores, porém envolvem pares de registros e valores de dois bytes.

<i>Mnemônica</i>	<i>Código hexa</i>
LD HL, dddd	21dddd
LD BC, dddd	01dddd
LD DE, dddd	11dddd
LD IX, dddd	DD21dddd
LD IY, dddd	FD21dddd
LD SP, dddd	31dddd

Estas instruções vão requerer três ou quatro locações na memória.

O primeiro byte do valor numérico da instrução vai sempre para o registro menos significativo do par de registros envolvido na instrução, enquanto o segundo byte, logicamente, vai para o outro registro envolvido, o registro mais significativo do par. Entende-se por registro menos significativo os registros L, C, E, X, Y ou P, e por registro mais significativo, os registros H, B, D, I, ou S.

Também estas instruções são análogas às instruções de atribuição de valores da linguagem em Basic. Aqui, elas carregam aquele par de registros com um valor especificado.

As instruções deste grupo não afetam as bandeiras.

GRUPO 3 – INSTRUÇÕES DE COPIAR E TROCAR CONTEÚDO DE REGISTROS

São ao todo 59 instruções do universo de instruções do microprocessador Z80, que tratam de copiar conteúdos de registros ou par de registros. Estas instruções podem ser subdivididas em quatro subgrupos.

SUBGRUPO 1 – INSTRUÇÕES DE COPIAR CONTEÚDOS DE REGISTROS SIMPLES

A tabela abaixo fornece o código das instruções que tratam da cópia de conteúdos de simples registros, genericamente denominados registros *r*, para outros registros especificados.

Registro <i>r</i>	LD A, <i>r</i>	LD H, <i>r</i>	LD L, <i>r</i>	LD B, <i>r</i>	LD C, <i>r</i>	LD D, <i>r</i>	LD E, <i>r</i>
A	7F	67	6F	47	4F	57	5F
H	7C	64	6C	44	4C	54	5C
L	7D	65	6D	45	4D	55	5D
B	78	60	68	40	48	50	58
C	79	61	69	41	49	51	59
D	7A	62	6A	42	4A	52	5A
E	7B	63	6B	43	4B	53	5B

Nenhuma das instruções contidas nessa tabela afetam as bandeiras indicadoras de estado. Existem ainda quatro instruções envolvendo os registros I e R:

<i>Mnemônica</i>	<i>Código hexa</i>
LD A, I	ED57
LD A, R	ED5F
LD I, A	ED47
LD R, A	ED4F

Estas instruções afetam a bandeira indicadora de paridade ou excesso (mais adiante você verá o que faz esta bandeira).

SUBGRUPO 2 – INSTRUÇÕES DE COPIAR CONTEÚDOS DE PAR DE REGISTROS

São apenas três instruções neste subgrupo, e todas envolvem o par de registro de função especial, chamado "ponteiro da pilha" (stack pointer).

<i>Mnemônica</i>	<i>Código hexa</i>
LD SP, HL	F9
LD SP, IX	DDF9
LD SP, IY	FDF9

Estas instruções não afetam as bandeiras indicadoras de estado.

Note que não existem operações de cópia de conteúdo de pares de registros genéricos e, portanto, as instruções acima não são apropriadas para o caso. Esta operação é desenvolvida com duas instruções de carregamento de registros simples e cópia de registros simples.

Por exemplo, para se desenvolver a operação: LD HL, DE, utilizamos primeiramente LD H, D e em seguida LD L, E.

Como alternativa, que consome mais memória e mais tempo de execução, o conteúdo do primeiro par de registros pode ser armazenado na pilha da máquina, para em seguida ser copiado no segundo registro (veja instruções da pilha).

SUBGRUPO 3 – INSTRUÇÃO EX DE,HL

Mnemônica	Código hexa
EX DE,HL	EB

Esta instrução, muito útil por sinal, permite que o programador troque o conteúdo do par de registros DE com aquele armazenado pelo par de registros HL, sem afetar qualquer bandeira indicadora de estado, com grande velocidade e economia de memória.

Esta instrução é normalmente utilizada quando um endereço ou um valor numérico que ocupa dois bytes deve ser movido do par de registros DE para o par HL, mas sem cancelar ou perder o conteúdo original do par HL.

SUBGRUPO 4 – INSTRUÇÕES COM O GRUPO ALTERNATIVO DE REGISTROS

Mnemônica	Código hexa
EXX	D9
EX AF, A'F'	08

A instrução EXX faz com que o conteúdo dos registros H, L, B, C, D, e E sejam trocados respectivamente com os conteúdos dos registros H', L', B', C', D' e E'.

A instrução EX AF, A'F', conforme o seu nome sugere, faz a troca entre os registros AF e A'F'.

Estes registros alternativos são sempre utilizados para armazenar endereços ou valores numéricos, protegendo estes valores contra qualquer erro ou acidente no decorrer do programa, podendo ser utilizados a qualquer momento, de um modo muito rápido e fácil.

GRUPO 4 – INSTRUÇÕES PARA CARREGAMENTO DE REGISTROS COM VALORES NUMÉRICOS COPIADOS DE LOCAÇÕES DA MEMÓRIA

O conjunto das instruções do Z80 possui muitas instruções que procuram dados em endereços da memória, para então carregá-los em determinados registros. Todas estas instruções requerem do programador a especificação do endereço, ou endereços, de onde os dados devam ser copiados, para, então, o registro, ou o par de registros receber estes dados.

As instruções deste grupo podem ser subdivididas em três subgrupos, dependendo da técnica de endereçamento selecionada pelo programador.

Estas técnicas de endereçamento são:

1- Endereçamento absoluto — o atual endereço de dois bytes é especificado segundo sua própria instrução.

2- Endereçamento indireto — o endereço de dois bytes está sempre disponível em algum par de registros de endereçamento.

3- Endereçamento indexado — o endereço da locação α deve ser computado pela adição de um valor de deslocamento, d , ao endereço base armazenado no par de registros IX ou IY.

SUBGRUPO 1 – INSTRUÇÕES UTILIZANDO ENDEREÇAMENTO ABSOLUTO

Mnemônica	Código hexa	
LD A, (end)	3A endereço	
LD HL, (end)	2A end	forma usual
	ED6B end	forma não usual
LD BC, (end)	ED4B end	
LD DE, (end)	ED5B end	
LD IX, (end)	DD2A end	
LD IY, (end)	FD2A end	
LD SP, (end)	ED7B end	

A instrução LD A, (end) é a única instrução do conjunto de instruções do Z80 que permite carregar, em endereçamento direto ou absoluto, o conteúdo especificado daquela locação da memória em um registro simples. Note que as instruções remanescentes deste grupo podem ser consideradas como sendo instruções duplas, ou seja, por exemplo, a instrução LD BC, (end) pode ser considerada como primeiramente LD C, (end) seguida de LD B, (end + 1).

Em qualquer caso, o conteúdo do endereço especificado é copiado no registro menos significativo, e o conteúdo do endereço seguinte é copiado no registro mais significativo.

SUBGRUPO 2 – INSTRUÇÕES QUE UTILIZAM ENDEREÇAMENTO IN-DIRETO

Mnemônica	Código hexa
LD A, (HL)	7E
LD A, (BC)	0A
LD A, (DE)	1A
LD A, (HL)	66
LD L, (HL)	6E
LD B, (HL)	46
LD C, (HL)	4E
LD D, (HL)	56
LD E, (HL)	5E

Em todos os casos, o endereço da locação de onde o byte deve ser copiado deve estar presente obrigatoriamente em algum par de registro entre HL, DE ou BC.

Note que, por exemplo, a instrução LD D, (BC) não existe, e, portanto, devem ser executadas outras instruções que efetuem o mesmo processamento:

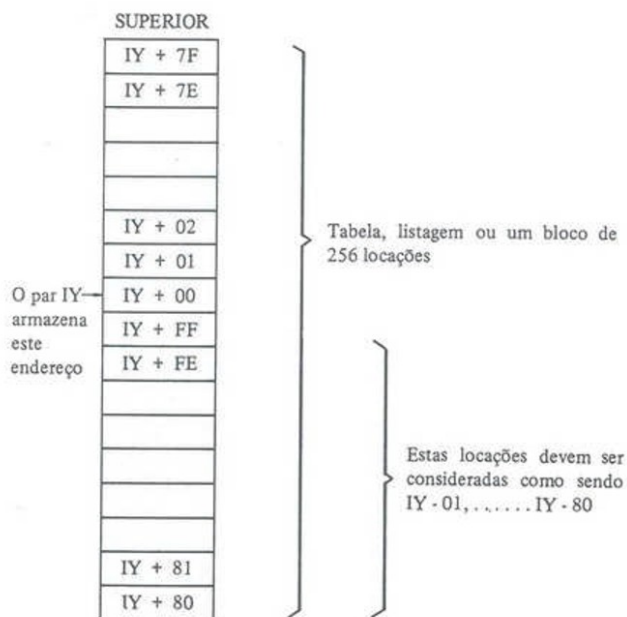
- 1- LD A, (BC) seguida de LD D, A, que alterará o conteúdo do registro A, ou:
- 2- LD H, B, seguida de LD L, C e finalmente LD D, (HL), que alterará o conteúdo de HL.

SUBGRUPO 3 – INSTRUÇÕES USANDO ENDEREÇAMENTO INDEXADO

As instruções deste subgrupo permitem ao programador carregar simples bytes de dados, em registros simples, armazenados sob a forma de listas, tabelas, ou apenas blocos de dados. O endereço base é armazenado no par de registros de indexação apropriado.

NOTA: Você já deve ter notado que as instruções envolvem os pares de registros IX e IY diferem apenas no código inicial, ou seja, para o par IX utiliza-se o código DD, e para o par IY utiliza-se o código FD. Portanto, no grupo de instruções anterior, para as instruções IY, substitua os códigos DD por FD.

O diagrama abaixo ilustra esse efeito:



Mnemônica	Código hexa
LD A, (IX + d)	DD73d
LD H, (IX + d)	DD66d
LD L, (IX + d)	DD6Ed
LD B, (IX + d)	DD46d
LD C, (IX + d)	DD4Ed
LD D, (IX + d)	DD56d
LD E, (IX + d)	DD5Ed

É interessante notar o tempo que o microprocessador Z80 leva para executar essas instruções. As instruções mais rápidas são as que compõem o subgrupo 2, que requerem do Z80 a busca de um simples byte de código e, então, o byte seguinte de dado.

As instruções do subgrupo 1 são mais complicadas e conseqüentemente levam mais tempo para ser executadas. Em termos técnicos, elas necessitam de 16 a 20 ciclos de tempo, e, finalmente, as instruções do subgrupo 3 são as que levam mais tempo ainda, apesar da sua grande praticidade. Elas necessitam de 19 ciclos de tempo para ser executadas.

Nenhuma das instruções deste grupo afeta as bandeiras indicadoras.

GRUPO 5 — INSTRUÇÕES PARA ARMAZENAR DADOS COPIADOS DE REGISTROS, OU VALORES NUMÉRICOS, EM ENDEREÇOS OU LOCAÇÕES DA MEMÓRIA

De uma maneira geral, as instruções deste grupo efetuam operações opostas às do grupo anterior.

Estas instruções permitem que conteúdos de registros especificados pelo programador sejam copiados em endereços da memória específicos, ou que valores numéricos sejam armazenados nesses endereços.

Outra vez, estas instruções são melhor analisadas, se divididas em três subgrupos:

SUBGRUPO 1 — INSTRUÇÕES UTILIZANDO ENDEREÇAMENTO ABSOLUTO

Mnemônica	Código hexa
LD (end), A	32 end
LD (end), HL	22 end ou ED63 end (forma não usual)
LD (end), BC	ED43 end
LD (end), DE	ED53 end
LD (end), IX	DD22 end
LD (end), IY	FD22 end
LD (end), SP	ED73 end

As instruções acima são as únicas a utilizar endereçamento absoluto, e é importante notar que não existe uma instrução para carregar um endereço específico com um valor numérico. Se o programador necessita efetuar essa operação, deve fazê-lo, carregando primeiramente o registro A, com o valor especificado, para então armazenar o valor desejado no endereço especificado.

Novamente uma instrução do tipo "LD (end), HL" é na realidade uma instrução dupla, pois requer LD (end), L e então LD (end + 1), H.

As instruções deste subgrupo são muito utilizadas para armazenar endereços e números em locações da memória, quando estes valores são considerados variáveis.

Por exemplo, é muito comum programadores em linguagem de máquina escreverem:

LD (RAMTOP), HL

SUBGRUPO 2 — INSTRUÇÕES QUE USAM ENDEREÇAMENTO INDIRETO

As instruções deste subgrupo permitem ao programador a cópia de conteúdos de registros simples em endereços da memória, que estejam armazenados nos pares de registros HL, BC, DE. Também existe uma instrução para carregar um byte em uma locação endereçada pelo par de registros HL.

Mnemônica	Código hexa
LD (HL), A	77
LD (BC), A	02
LD (DE), A	12
LD (HL), H	74
LD (HL), L	75
LD (HL), B	70
LD (HL), C	71
LD (HL), D	72
LD (HL), E	73
LD (HL), dd	36dd

SUBGRUPO 3 — INSTRUÇÕES UTILIZANDO ENDEREÇAMENTO INDEXADO

Mnemônica	Código hexa
LD (IX + d), A	DD77d
LD (IX + d), H	DD74d
LD (IX + d), L	DD75d
LD (IX + d), B	DD70d
LD (IX + d), C	DD71d
LD (IX + d), D	DD72d
LD (IX + d), E	DD73d
LD (IX + d), dd	DD36d dd

Novamente, para instruções que envolvem o par de registros IY, mude DD para FD e IX para IY.

GRUPO 6 – INSTRUÇÕES DE ADIÇÃO

Este grupo é o primeiro dos quatro grupos do conjunto de instruções do microprocessador Z80, que envolve operações aritméticas ou lógicas.

As instruções de adição permitem ao programador adicionar, em aritmética binária absoluta, um número especificado ao conteúdo de um registro simples, ou ao conteúdo de um par de registros ou, ainda, a um endereço indexado da memória.

As instruções deste grupo podem ser subdivididas em três subgrupos, de acordo com as suas mnemônicas.

Estes três subgrupos são:

- 1- Instruções de adição (ADD)
- 2- Instruções de incremento (INC), ou seja, um caso especial de adição, quando apenas uma unidade (1) é adicionada a um número existente.
- 3- Instruções de adição, porém considerando-se o estado da bandeira indicadora de transporte (carry flag) (ADC). Conforme já vimos, esta bandeira indicadora de transporte é um bit do registro F, utilizado para nos avisar se houve aquele nosso conhecido “vai um” nas operações aritméticas de soma.

As instruções de adição, bem como as instruções de adição com transporte, afetam a bandeira indicadora de transporte, mas as instruções de incremento não, fato esse que em algumas situações oferece vantagens.

SUBGRUPO 1 – INSTRUÇÕES DE ADIÇÃO – ADD

<i>Mnemônica</i>	<i>Código hexa</i>
ADD A,dd	C6dd
ADD A,A	87
ADD A,H	84
ADD A,L	85
ADD A,B	80
ADD A,C	81
ADD A,D	82
ADD A,E	83
ADD A,(HL)	86
ADD A,(IX + d)	DD86d
ADD A,(IY + d)	FD86d
ADD HL,SP	39

ADD IX,IX	DD29
ADD IX,BC	DD09
ADD IX,DE	DD19
ADD IX,SP	DD39

Para instruções que envolvem o par de registros IY, substitua IX por IY e os códigos DD por FD.

Veja os exemplos:

- 1- Registro A armazenando 60h
Registro B armazenando 90h
ADD A,B resultará:
Registro A armazenando F0h
Registro B armazenando 90h (conteúdo inalterado)
Bandeira indicadora de transporte resetada
- 2- Registro A armazenando A8h
Registro B armazenando 7Eh
ADD A,B resultará:
Registro A armazenando 26h
Registro B armazenando 7Eh (inalterado)
Bandeira indicadora de transporte setada (houve transporte).

Confira com lápis e papel.

SUBGRUPO 2 – INSTRUÇÕES DE INCREMENTO

As instruções deste grupo permitem que o conteúdo de um simples registro, ou de um par de registros, ou mesmo de uma locação da memória seja incrementado em uma unidade. Em todos os casos, a bandeira indicadora de transporte é ignorada, ou seja, inalterada.

<i>Mnemônica</i>	<i>Código hexa</i>
INC A	3C
INC H	24
INC L	2C
INC B	04
INC C	0C
INC D	14
INC E	1C
INC (HL)	34
INC (IX + d)	DD34d
INC (IY + d)	FD34d

INC HL	23
INC BC	03
INC DE	13
INC SP	33
INC IX	DD23
INC IY	FD23

SUBGRUPO 3 – INSTRUÇÕES DE ADIÇÃO COM TRANSPORTE (ADC)

<i>Mnemônica</i>	<i>Código hexa</i>
ADC A,dd	CEdd
ADC A,A	8F
ADC A,H	8C
ADC A,L	8D
ADC A,B	88
ADC A,C	89
ADC A,D	8A
ADC A,E	8B
ADC A,(HL)	8E
ADC A,(IX + d)	DD8Ed
ADC A,(IY + d)	FD8Ed
ADC HL,HL	ED6A
ADC HL,BC	ED4A
ADC HL,DE	ED5A
ADC HL,SP	ED7A

As instruções deste subgrupo permitem ao programador adicionar dois números, juntamente com o estado da bandeira indicadora de transporte, pois todas as instruções deste grupo, conforme as próprias mnemônicas dizem, afetam essa bandeira. Esta será resetada (0) se a instrução ADC em execução não der excesso, e será setada (1), se houver excesso na operação corrente.

Por exemplo:

- 1- Registro A armazenando 60h
Registro B armazenando 90h
Bandeira indicadora de transporte setada
A operação ADC A,B resultará:
Registro A armazenando F1h
Registro B armazenando 90
Bandeira indicadora de transporte resetada

- 2- Registro A armazenando A8h
Registro B armazenando 7Eh
Bandeira indicadora de transporte setada
A operação ADC A,B resultará:
Registro A armazenando 27h
Registro B armazenando 7Eh
E a bandeira de transporte setada

Agora, pegue um lápis e um papel, pegue esses valores hexadecimais acima, converta-os para o sistema decimal e confira o resultado, podendo considerar a bandeira indicadora de transporte como sendo um “nono bit” do acumulador (registro A).

GRUPO 7 – INSTRUÇÕES DE SUBTRAÇÃO

As instruções de subtração permitem que o programador subtraia, em aritmética binária absoluta, um número especificado do conteúdo de um registro simples, de um par de registros ou de uma locação endereçada da memória.

Novamente este grupo pode ser subdividido em três subgrupos, conforme suas mnemônicas.

Esses três subgrupos são:

- 1- Instruções SUB – de subtração simples
- 2- Instruções DEC – casos especiais de subtração, onde um número específico é decrementado de uma unidade.
- 3- Instruções SBC – o valor da bandeira indicadora de transporte também é subtraído do resultado.

Deste grupo, apenas as instruções DEC não afetam a bandeira indicadora de transporte.

SUBGRUPO 1 – INSTRUÇÕES SUB

<i>Mnemônica</i>	<i>Código hexa</i>
SUB dd	D6dd
SUB A	97
SUB H	94
SUB L	95
SUB B	90

SUB C	91
SUB D	92
SUB E	93
SUB (HL)	96
SUB (IX + d)	DD96d
SUB (IY + d)	FD96d

Nota: As mnemônicas para as instruções de subtração SUB são normalmente escritas na forma acima, ou seja, "SUB L" é o mesmo que "SUB A,L" etc, pois todas as instruções de subtração deste subgrupo envolvem o acumulador, ou o registro A.

No Z80, as instruções de subtração fornecem uma subtração binária absoluta "verdadeira", conforme exemplo a seguir. A bandeira indicadora de transporte é resetada se o valor original do registro A é "maior que" ou "igual" ao subtraendo (o segundo número na subtração), e é setada se o valor de registro A é "menor que".

Exemplos:

- 1- Com o registro A armazenando DCh e o registro B armazenando AAh

A instrução SUB B resultará em A armazenando 32h, e a bandeira indicadora de transporte estará resetada, porque não houve excesso.

- 2- Com o registro A armazenando AAh e o registro B armazenando DCh

A instrução SUB B resultará em A armazenando CEh, o registro B armazenando DCh e a bandeira indicadora de transporte estará setada, pois houve excesso.

Confira.

SUBGRUPO 2 – INSTRUÇÕES DEC (DE DECREMENTO)

As instruções deste subgrupo permitem que seja subtraída uma unidade do conteúdo de um registro simples de 8 bits, ou de um par de registros, ou de um endereço da memória. Em qualquer caso, a bandeira indicadora de transporte não é afetada.

Mnemônica	Código hexa
DEC A	3D
DEC H	25
DEC L	2D
DEC B	05
DEC C	0d
DEC D	15
DEC E	LD
DEC (HL)	35
DEC (IX + d)	DD35d
DEC (IY + d)	FD35d
DEC HL	2B
DEC BC	0B
DEC DE	1B
DEC SP	3B
DEC IX	DD2B
DEC IY	FD2B

SUBGRUPO 3 – INSTRUÇÕES DE SUBTRAÇÃO LEVANDO-SE EM CONSIDERAÇÃO A BANDEIRA INDICADORA DE TRANSPORTE

Mnemônica	Código hexa
SBC A,dd	DEdd
SBC A,A	9F
SBC A,H	9C
SBC A,L	9D
SBC A,B	98
SBC A,C	99
SBC A,D	9A
SBC A,E	9B
SBC A,(HL)	9E
SBC A,(IX + d)	DD9Ed
SBC A,(IY + d)	FD9Ed
SBC HL,HL	ED62
SBC HL,BC	ED42
SBC HL,DE	ED52
SBC HL,SP	ED72

Uma operação SBC efetuará uma subtração verdadeira se a bandeira indicadora de transporte estiver resetada, mas executará uma subtração, considerando o excesso, se a bandeira de transporte estiver setada.

GRUPO 8 – INSTRUÇÕES DE COMPARAÇÃO

As instruções deste grupo são usadas muito frequentemente em qualquer rotina de código de máquina. Elas permitem que o programador compare o valor armazenado no registro A, com uma constante, um valor armazenado em um registro qualquer ou um valor armazenado em um endereço da memória.

Uma instrução de comparação efetua uma operação de subtração, sem transporte, mas descarta a resposta após usá-la para setar as devidas bandeiras indicadoras do registro F. O valor original do registro A permanece inalterado.

A bandeira indicadora de transporte é afetada da mesma maneira que numa operação de subtração. Uma comparação que seja "maior que" ou "igual a" *reseta* a bandeira de transporte, e uma comparação que seja "menor que", *seta* a bandeira de transporte.

As instruções deste grupo são instruções de comparação simples, e as instruções de comparações de blocos serão consideradas mais tarde.

Mnemônica	Código hexa
CP dd	FEdd
CP A	BF
CP H	BC
CP L	BD
CP B	B8
CP C	B9
CP D	BA
CP E	BB
CP (HL)	BE
CP (IX + d)	DDBEd
CP (IY + d)	FDBEd

Essas instruções desenvolvem processamento análogo às instruções em Basic, de operações e decisões lógicas, tipo IF...THEN...

GRUPO 9 – INSTRUÇÕES LÓGICAS

No conjunto de instruções do Z80 existem instruções para processarem operações AND, OR e XOR que comparam o conteúdo do acumulador com o conteúdo de outra locação especificada. A operação é desenvolvida bit a bit, e o resultado de 8 bits é armazenado no acumulador.

Dividimos este grupo em três subgrupos, de acordo com suas mnemônicas.

SUBGRUPO 1 – INSTRUÇÕES AND

A operação lógica AND efetuada entre dois bits dará como resultado um bit com valor 1, apenas se os dois bits envolvidos na operação valerem 1 também. Em outro caso, o resultado será 0.

O exemplo a seguir mostra como uma instrução AND desenvolve oito operações separadas bit a bit.

10101010	hexa AA
AND	AND
11000000	CO
=10000000	= 80

Outra vez, pegue um lápis e papel, e confira.

Mnemônica	Código hexa
AND dd	E6dd
AND A	A7
AND H	A4
AND L	A5
AND B	A0
AND C	A1
AND D	A2
AND E	A3
AND (HL)	A6
AND (IX + d)	DDA6d
AND (IY + d)	FDA6d

Ao usar uma instrução AND, todos os bits do acumulador serão resetados. Esse processo permite ao programador controlar certos bits de um byte de dados.

SUBGRUPO 2 – INSTRUÇÕES OR

A operação lógica OR, executada entre dois bits, dará como resultado um terceiro bit, valendo 1, se apenas um ou até os dois bits envolvidos valerem 1.

O exemplo a seguir ilustra esse processo.

10101010	AA
OR	OR
11000000	CO
=11101010	=EA
<i>Mnemônica</i>	<i>Código hexa</i>
OR dd	F6dd
OR A	B7
OR H	b4
OR L	B5
OR B	B0
OR C	B1
OR D	B2
OR E	B3
OR (HL)	B6
OR (IX + d)	DDB6d
OR (IY + d)	FDB6d

SUBGRUPO 3 – INSTRUÇÕES XOR

O operação lógica XOR, desenvolvida entre dois bits, dará como resultado um terceiro bit, que valerá 1 se apenas um bit dos dois envolvidos também valer 1, mas, veja bem, somente se um dos bits envolvidos valer 1, e nunca os dois!

O exemplo a seguir demonstra essa operação.

10101010	AA
XOR	XOR
11000000	CO
=01101010	=6A
<i>Mnemônica</i>	<i>Código hexa</i>
XOR dd	EEd
XOR A	AF
XOR H	AC
XOR L	AD
XOR B	A8
XOR C	A9
XOR D	AA
XOR E	AB
XOR (IX + d)	DDAEd
XOR (IY + d)	FDAEd

Ao se usar a instrução XOR, os bits do registro A serão alterados se necessário for.

O uso das instruções XOR em rotinas em código de máquina são um pouco complicadas, mas a instrução XOR A é freqüentemente usada como uma alternativa a LD A,0. Ambas as instruções limpam o acumulador, mas a primeira utiliza apenas um endereço, enquanto a segunda consome dois endereços.

GRUPO 10 – INSTRUÇÕES DE SALTO (JUMP)

No conjunto de instruções do Z80 existem 17 instruções de salto, que permitem que o programador efetue saltos dentro de um programa. Um salto em código de máquina pode ser comparado à instrução BASIC "GOTO".

As instruções deste grupo são melhor analisadas, se divididas em oito subgrupos. Quatro destes subgrupos contêm instruções condicionais que dependem do estado das bandeiras indicadoras do registro F.

SUBGRUPO 1 – INSTRUÇÃO DE SALTO ABSOLUTO (JP de jump)

Esta é a instrução clássica de salto. Quando a instrução JP end é executada, permite que o endereço especificado seja armazenado no PC, ou contador de programas, fazendo com que a execução do programa continue a partir daquele endereço.

<i>Mnemônica</i>	<i>Código hexa</i>
JP end	C3 end

SUBGRUPO 2 – INSTRUÇÕES DE SALTO QUE UTILIZAM ENDEREÇAMENTO INDIRETO

<i>Mnemônica</i>	<i>Código hexa</i>
JP (HL)	E9
JP (IX)	DDE9
JP (IY)	FDE9

Estas três instruções permitem que um byte devidamente armazenado no par de registros especificado seja carregado no contador de programas. As instruções deste subgrupo são freqüentemente utilizadas quando deve ser feito um salto para uma locação especificada em uma tabela de endereços.

SUBGRUPO 3 – INSTRUÇÕES DE SALTO RELATIVO (JR de jump relative)

Mnemônica	Código hexa
JR e	18 e

Esta instrução permite que o programador salte 127 endereços para a frente (positivos) e 128 endereços para trás (negativos), a partir do endereço corrente. Note que o endereço corrente é de fato o seguinte ao deslocamento e, especificado na instrução.

O deslocamento é sempre considerado, em aritmética, complemento de dois, e um deslocamento positivo dá o número de locações que devem ser saltados acima, enquanto um deslocamento negativo mostra em quanto o contador de programas deve ser reduzido.

SUBGRUPO 4 – INSTRUÇÕES DE SALTO CONDICIONAIS RELATIVAS AO ESTADO DA BANDEIRA INDICADORA DE TRANSPORTE

São quatro instruções neste subgrupo, que permitem que seja feito um salto somente se a bandeira indicadora de transporte estiver no estado especificado pela instrução.

Agora vamos ver realmente o que é esta bandeira indicadora de transporte.

Esta bandeira é o bit 0 do registro F e é essencialmente um indicador que mostra se houve ou não um excesso numa operação binária, ou seja, se ela está setada em certas situações e resetada em outras ocasiões. Em muitas situações, a bandeira indicadora de transporte não é afetada pela execução de algumas instruções.

Em resumo:

1. Todas as instruções ADD e ADC afetam esta bandeira. Se não houver excesso a bandeira será resetada, mas se houver excesso, a bandeira será setada.
2. Todas as instruções SUB, SBC e CP afetam também esta bandeira.
3. Instruções como AND, OR e XOR resetam a bandeira indicadora de transporte.
4. As instruções de rotação que veremos adiante também afetam esta bandeira.

As instruções deste subgrupo são:

Mnemônica	Código hexa
JP NC, end	D2 end
JR NC, e	30 e
JP C, end	DA end
JR C, e	38 e

Nas duas primeiras instruções, o salto somente será executado se a bandeira indicadora de transporte estiver resetada.

Nas duas últimas, o salto será executado se a bandeira de transporte estiver setada (valer 1).

SUBGRUPO 5 – INSTRUÇÕES DE SALTO CONDICIONAIS RELATIVAS AO ESTADO DA BANDEIRA ZERO

Esta bandeira zero é o bit 6 do registro F e em muitas vezes indica se o resultado de uma determinada operação foi zero (quando ela estará setada, ou seja, valerá 1) ou se o resultado de uma operação foi diferente de zero (quando então ela estará resetada, quer dizer, valerá 0).

Por exemplo:

6C ADD 5A resultará C6 e a bandeira zero resultada, mas
6C ADD 94 resultará 00 e a bandeira zero setada.

Pegue o lápis e papel confira estes resultados, pois na minha opinião esta é a melhor maneira de você fixar estes conceitos fundamentais.

Resumindo:

1. Instruções ADD, INC, ADC, SUB, DEC, SBC, CP, AND, OR e XOR usando registros simples de um byte, e as instruções ADC e SBC usando pares de registros irão setar a bandeira zero, se o resultado da operação em questão for zero.
2. Instruções de rotação, ou instruções de testar determinados bits (que também veremos mais à frente) ou instruções de procura de blocos afetam a bandeira zero.
3. Instruções LD, com exceção de LD A, I e LD A, R, não afetam esta bandeira.

Neste subgrupo são quatro instruções, que permitem que seja efetuado um salto apenas se o estado da bandeira zero coincidir com o especificado pela instrução.

<i>Mnemônica</i>	<i>Código hexa</i>
JP NZ, end	C2 end
JR NZ, e	20 e
JP Z, end	CA end
JR Z, e	28 e

Como o caso anterior, as duas primeiras instruções permitem que seja considerado o salto apenas se a bandeira zero estiver resetada, enquanto nas duas últimas, o salto somente será executado se a bandeira zero estiver setada.

SUBGRUPO 6 – INSTRUÇÕES DE SALTO CONDICIONAIS RELATIVAS AO ESTADO DA BANDEIRA DE SINAL

Esta bandeira de sinal é o bit 7 do registro F, e em muitos casos é uma cópia do bit mais à esquerda do resultado.

Sempre que um número de 8 bits, ou um número de 16 bits é considerado na sua forma de complemento de 2 aritmético, então o bit mais à esquerda, ou seja, o bit 7 ou o bit 15 é considerado como bit de sinal. *Atenção – este bit de sinal será resetado para números positivos e setado para números negativos.*

Resumindo:

1. Instruções ADD, INC, ADC, SUB, DEC, SBC, CP, AND, OR e XOR usando registros simples de 8 bits, e as instruções ADC e SBC usando pares de registros afetam a bandeira de sinal.

2. Instruções de busca de blocos e muitas instruções de rotação também afetam esta bandeira.

3. Instruções LD, com exceção de LD A,I e LD A,R, não afetam a bandeira de sinal.

Neste subgrupo, são duas instruções que permitem que seja realizado um salto somente se o estado da bandeira de sinal coincidir com a especificação da instrução.

<i>Mnemônica</i>	<i>Código hexa</i>
JP P, end	F2 end
JP M, end	FA end

Na primeira instrução o salto somente será executado se o resultado for positivo, e na segunda, se for negativo.

As instruções deste subgrupo não são comumente usadas, porque requerem endereçamento absoluto e também porque um bit de sinal pode ser lido de diversas maneiras.

SUBGRUPO 7 – INSTRUÇÕES DE SALTO CONDICIONAIS RELATIVAS AO ESTADO DA BANDEIRA DE PARIDADE/EXCESSO

São duas instruções que permitem que seja efetuado um salto somente se a bandeira de paridade/excesso estiver nas condições especificadas pela instrução.

Esta bandeira é o bit 2 do registro F e é uma bandeira de propósito duplo. Certas instruções usam-na para indicar “excesso”, enquanto outras instruções utilizam-na para armazenar o resultado de um teste de paridade.

O conceito de excesso, aqui, não se aplica ao excesso binário, mas ao excesso de um complemento de dois aritmético ilustrado a seguir:

Consideremos:

Em hexa 0A ADD 5C = 66

Em decimal 10 ADD 92 = 102

Correto – não houve excesso

Em hexa 6A ADD 32 = 9C

Em decimal 106 ADD 50 = -100 (lembre-se do bit de sinal)

Errado – houve excesso

Excessos também acontecem com subtração. Veja:

Em hexa 83 SUB 14 = 6F

Em decimal -125 SUB 20 = 111

Errado – houve excesso

A bandeira de excesso/paridade é setada quando ocorrem excessos.

Lápis e papel na mão, para descobrir as operações exemplificadas acima.

O conceito de paridade refere-se ao número de bits setados em um determinado byte. A paridade existirá quando o número de bits setados for par.

Por exemplo:

- o byte 01010101 tem paridade par e a bandeira setada;
- o byte 00000001 tem paridade ímpar e a bandeira resetada.

Resumindo

1. Instruções ADD, ADC, SBC, CP usando registros simples e instruções ADC e SBC usando pares de registros têm seu resultado testado em função do excesso.

2. Instruções AND, OR e XOR e rotações têm seu resultado testado em função da paridade.

3. Uma instrução INC vai setar esta bandeira se o resultado for 80h, e uma instrução DEC vai setar a bandeira se o resultado for 7Fh.

4. Várias outras instruções também afetam a bandeira de excesso/paridade.

As instruções deste subgrupo são:

<i>Mnemônica</i>	<i>Código hexa</i>
JP PO, end	E2 end
JP PE, end	EA end

Na primeira instrução, o salto será executado se a paridade for ímpar ou não houver excesso.

Na segunda, o salto será executado se a paridade for par ou houver excesso.

As instruções deste subgrupo não são muito utilizadas, devido à confusão que podem causar, e porque também podem ser substituídas por outras instruções.

GRUPO 11 — INSTRUÇÃO DJNZ, e

Esta única instrução deste grupo é uma das mais úteis e das mais usadas de todo o conjunto de instruções do microprocessador Z80.

<i>Mnemônica</i>	<i>Código hexa</i>
DJNZ, e	10 e

A mnemônica significa “decremente o registro B e efetue um salto relativo se a bandeira zero estiver resetada”.

Esta instrução pode ser comparada a um loop Basic, com passo negativo, do tipo:

```
FOR f = 10 to 1 STEP -1 : ... :NEXT f
```

Nesse loop, a variável de controle f é inicializada em 10, e a cada passagem pela variável, ela é decrementada em uma unidade, até atingir o valor limite.

A instrução “DJNZ, e” é usada de uma maneira similar. Primeiramente o programador deve especificar o tamanho da variável do loop e armazená-la no registro B; a seguir, as instruções que serão repetidas, e, finalmente, muito cuidado deve ser tomado, a fim de que o valor “e” seja apropriado.

O exemplo abaixo mostra como se imprime o alfabeto na tela, com o emprego da instrução “DJNZ, e”.

LD B, 1A	São 26 letras no alfabeto
loop LD A, 5B	A letra A é a primeira
SUB B	Em hexa 5B - 1A = 41
RST 0010	Instrução para imprimir mensagens na tela (veremos adiante)
DJNZ, loop	Pula para B, C etc., da mesma maneira que NEXT f

Os códigos hexa para o exemplo acima são:

06, 1A, 3E, 5B, 90, D7, 10, FA

Lápis e papel, através do Anexo A deste livro, converta esses valores para decimal e tente fazer um programa que execute essa rotina.

GRUPO 12 — INSTRUÇÕES DA PILHA (STACK)

Em muitos programas em código de máquina, o uso extensivo da pilha da máquina é feito pelo programador como um lugar para guardar dados, e pelo microprocessador, para guardar endereços que serão utilizados posteriormente. As instruções que formam este grupo podem ser subdivididas em dois grupos do usuário e três subgrupos do microprocessador.

SUBGRUPO 1 – INSTRUÇÕES PUSH E POP

Estas instruções permitem que o programador "PUSH", isto é, guarde, ou salve, dois bytes de dados na pilha da máquina, para mais tarde "POP", ou seja, copiar da pilha aqueles dois bytes.

Este par de bytes pode ser copiado de e para um par de registros especificado.

<i>Mnemônica</i>	<i>Código hexa</i>
PUSH AF	F5
PUSH HL	E5
PUSH BC	C5
PUSH DE	D5
PUSH IX	DDE5
PUSH IY	FDE5
POP AF	F1
POP HL	E1
POP BC	C1
POP DE	D1
POP IX	DDE1
POP IY	FDE1

Quando uma instrução PUSH é executada, o ponteiro da pilha (*stack pointer*) é primeiramente decrementado para apontar para uma locação livre. Uma cópia do conteúdo do registro mais significativo do par de registros envolvido é então feita nesta locação. Então o ponteiro da pilha é decrementado novamente para armazenar na nova locação o valor contido no registro menos significativo do par envolvido.

As ações inversas e essas descritas são executadas no caso de instruções POP. Note que sempre que uma dessas instruções é executada, ao final delas, o ponteiro da pilha vai apontar para aquela locação especificada pelo programador, ou aquela onde estava o processamento normal do programa.

Nota importante — estas instruções são sempre usadas aos pares, ou seja, para cada instrução PUSH utilizada deve obrigatoriamente haver uma instrução POP equivalente.

SUBGRUPO 2 – INSTRUÇÕES DE MUDANÇA DO VALOR DA PILHA

As instruções deste subgrupo não são muito usadas, mas em algumas ocasiões são muito úteis.

<i>Mnemônicas</i>	<i>Código hexa</i>
EX (SP), HL	E3
EX (SP), IX	DDE3
EX (SP), IY	FDE3

Estas instruções permitem ao programador mudar, ou trocar, o valor corrente armazenado num par de registros especificado pela última entrada na pilha da máquina. E o ponteiro da pilha, qual seja, o par de registros SP não é alterado.

O uso destas instruções é um pouco confuso, principalmente em se tratando de rotinas em código de máquina mais extensas, e elas são melhores consideradas como alternativas a PUSH e POP em casos especiais.

Considere, por exemplo, a situação a seguir:

O valor AA está na pilha da máquina, e o valor BB está armazenado no par de registros HL.

É desejo do programador trocar estes valores por outros.

Existem duas maneiras de fazê-lo:

1. Usando a instrução EX (SP), HL;
2. Usando outro par de registros para armazenamento temporário para AA.

POP BC	Salva AA em BC
PUSH HL	B é colocado na pilha
PUSH BC	Move AA para HL
POP HL	de um modo ou de outro

As instruções deste subgrupo também podem ser utilizadas para manipular endereços de retorno.

SUBGRUPO 3 – INSTRUÇÕES CALL

As instruções em código de máquina CALL são diretamente equivalentes ao comando Basic GOSUB. As instruções estão incluídas neste grupo porque o microprocessador usa a pilha da máquina como uma área onde os endereços de retorno são armazenados.

Existem nove instruções neste subgrupo que permitem que uma sub-rotina seja chamada condicional ou incondicionalmente em relação ao estado das principais bandeiras indicadoras de estado.

<i>Mnemônica</i>	<i>Código hexa</i>	<i>Comentários</i>
CALL end	CD end	Incondicional
CALL C, end	DC end	Bandeira C setada
CALL NC, end	D4 end	Bandeira C resetada
CALL Z, end	CC end	Bandeira zero setada
CALL NZ, end	C4 end	Bandeira zero resetada
CALL M, end	FC end	Bandeira sinal setada
CALL P, end	F4 end	Bandeira sinal resetada
CALL PE, end	EC end	Bandeira paridade/excesso setada
CALL PO, end	E4 end	Bandeira paridade/excesso resetada

As ações de uma instrução CALL são:

1. O valor corrente do PC, ou contador de programas, isto é, o endereço da primeira locação após "end" da instrução CALL, é salvo na pilha da máquina. O ponteiro da pilha é manipulado como numa instrução PUSH. O byte mais significativo do contador de programas vai para a locação seguinte à do byte menos significativo.

2. O endereço é então copiado no contador de programas e a execução do programa propriamente dito continua.

SUBGRUPO 4 – INSTRUÇÕES RET

As instruções em código de máquina RET são diretamente equivalentes ao comando Basic RETURN. Também são nove instruções neste subgrupo, que permitem o retorno condicional ou incondicional, dependendo do estado das bandeiras principais.

<i>Mnemônica</i>	<i>Código hexa</i>	<i>Comentários</i>
RET	C9	Incondicional
RET C	D8	Bandeira C setada
RET NC	D0	Bandeira C resetada
RET Z	C8	Bandeira zero setada
RET NZ	C0	Bandeira zero resetada
RET M	F8	Bandeira sinal setada
RET P	F0	Bandeira sinal resetada
RET PE	E8	Bandeira paridade/excesso setada
RET PO	E0	Bandeira paridade/excesso resetada

A ação de uma instrução RET é a de copiar a última entrada na pilha da máquina para o contador de programas. No entanto, o ponteiro da pilha é incrementado duas vezes.

Não é nada comum, em Basic, manipular a pilha de GOSUB, mas em linguagem de máquina freqüentemente somos obrigados a fazê-lo, o que requer muitos cuidados com os valores a serem processados, bem como com os endereços a serem manipulados.

SUBGRUPO 5 – INSTRUÇÕES RST (RESTART)

O último subgrupo de instruções deste grupo contém as instruções especiais RST, ou RESTART (reinício). Estas instruções, muito úteis por sinal, são na verdade instruções CALL, que diferem das CALL verdadeiras apenas por não requererem a especificação de endereço.

<i>Mnemônica</i>	<i>Código hexa</i>	<i>Comentário</i>
RST 0000	C7	CALL 0000
RST 0008	CF	CALL 0008
RST 0010	D7	CALL 0010
RST 0018	DF	CALL 0018
RST 0020	E7	CALL 0020
RST 0028	EF	CALL 0028
RST 0030	F7	CALL 0030
RST 0038	FF	CALL 0038

No TK 90X, as oito instruções RST referem-se a endereços de entrada em sub-rotinas em linguagem de máquina armazenados na memória ROM do micro, e serão discutidas mais tarde.

GRUPO 13 – INSTRUÇÕES DE ROTAÇÃO

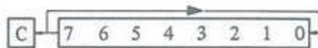
No conjunto de instruções do Z80, existe um grande número de instruções para rotação de bits de determinados bytes. São instruções quase sempre muito úteis.

Rodar um byte para a esquerda tem o efeito de duplicar o seu valor, sem perder o conteúdo do bit mais significativo, enquanto, rodar o byte à direita significa reduzir à metade o seu valor.

O diagrama a seguir mostra a variedade de rotações que são possíveis.

RLC & RCLA

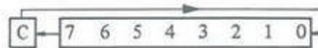
rotate left with carry
rotação à esquerda com transporte



Bit 7
vai para .
Bandeira C
e para Bit 0.

RL & RLA

rotate left
rotação à esquerda



Bit 7
vai para
Bandeira C,
que vai para
Bit 0.

SLA

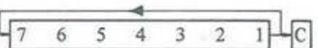
Shift left
Deslocamento à esquerda



Bit 0 é
resetado.
Bit 7
vai para
Bandeira C.

RRC & RRCA

rotate right with carry
rotação à direita com transporte



Bit 0
vai para
Bandeira C
e para Bit 7.

RR & RRA

rotate right
rotação à direita



Bit 0
vai para
Bandeira C.
Bandeira C
vai para
Bit 7.

SRA

Shift right
Deslocamento à direita



Bit 0
vai para
Bandeira C.

SRL

Logical shift right
Deslocamento lógico à direita

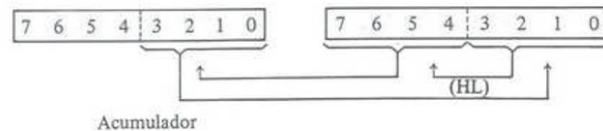


Bit 0
vai para
Bandeira C.
Bit 7
é resetado.

RLD

rotate digit left and right between

Accumulator and location (hl)

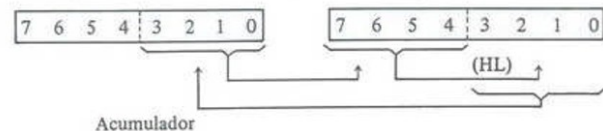


Acumulador

RRD

rotate digit right and left between

Accumulator and location (HL)



Acumulador

A tabela abaixo mostra as instruções deste grupo:

	RLC	RL	SLA	RRC	RR	SRA	SRL
A	CB07	CB17	CB27	CB0F	CB1F	CB2F	CB3F
H	CB04	CB14	CB24	CB0C	CB1C	CB2C	CB3C
L	CB05	CB15	CB25	CB0D	CB1D	CB2D	CB3D
B	CB00	CB10	CB20	CB08	CB18	CB28	CB38
C	CB01	CB11	CB21	CB09	CB19	CB29	CB39
D	CB02	CB12	CB	CB	CB1A	CB2A	CB3A
E	CB03	CB13	CB23	CB0B	CB1B	CB2B	CB3B
(HL)	CB06	CB16	CB26	CB0E	CB1E	CB2E	CB3E
(IX + d)	DDCB	DDCB	DDCB	DDCB	DDCB	DDCB	DDCB
	d06	d16	d26	d0E	d1E	d2E	d3E
(IY + d)	FDCB	FDCB	FDCB	FDCB	FDCB	FDCB	FDCB
	d06	d16	d26	d0E	d1E	d2E	d3E

Existem ainda quatro instruções de um byte, para rodar o acumulador, e duas instruções de manipulação de "nibbles", ou seja, quatro bits de um byte dividido ao meio, do bit 0 ao bit 3 e do bit 4 ao bit 7, respectivamente, nibble menos significativo e nibble mais significativo.

Mnemônica	Código hexa
RLCA	07
RLA	17
RRCA	0F
RRA	1F
RRD	ED67
RLD	ED6F

Quanto às bandeiras indicadoras:

1. Todas as instruções, exceto RLD e RRD afetam a bandeira indicadora de transporte.
2. Todas as instruções, exceto as quatro instruções de bytes simples, afetam respectivamente as bandeiras zero, de sinal e de paridade/excesso.

GRUPO 14 – INSTRUÇÕES DE MANIPULAÇÃO DE BITS

Existem, no conjunto das instruções em código de máquina do microprocessador Z80, instruções que permitem ao programador testar, setar ou resetar um bit especificado de um byte qualquer armazenado em um registro ou em um endereço da memória. Estes três tipos de instrução serão vistos a seguir.

SUBGRUPO 1 – INSTRUÇÕES BIT

Estas instruções permitem que o programador determine o estado de um bit específico.

Uma instrução BIT seta a bandeira zero, se o bit testado está resetado (vale 0), e vice-versa.

SUBGRUPO 2 – INSTRUÇÕES SET

Estas instruções permitem ao programador setar determinado bit de um byte. Nenhuma bandeira é afetada.

SUBGRUPO 3 – INSTRUÇÕES RES (RESET)

Inversamente ao grupo anterior, estas instruções permitem que o programador resete um determinado bit de um byte. Também não afetam nenhuma das bandeiras indicadoras de estado.

As instruções destes três subgrupos estão na tabela a seguir.

		BIT 0	BIT 1	BIT 2	BIT 3	BIT 4	BIT 5	BIT 6	BIT 7
Reg.	BIT	47	4F	57	5F	67	6F	77	7F
A	RES	87	8F	97	9F	A7	AF	B7	BF
CB ...	SET	C7	CF	D7	DF	E7	EF	F7	FF
Reg.	BIT	44	4C	54	5C	64	6C	74	7C
H	RES	84	8C	94	9C	A4	AC	B4	BC
CB ...	SET	C4	CC	D4	DC	E4	EC	F4	FC
Reg.	BIT	45	4D	55	5D	65	6D	75	7D
L	RES	85	8D	95	9D	A5	AD	B5	BD
CB ...	SET	C5	CD	D5	DD	E5	ED	F5	FD
Reg.	BIT	40	48	50	58	60	68	70	78
B	RES	80	88	90	98	A0	A8	B0	B8
CB ...	SET	C0	C8	D0	D8	E0	E8	F0	F8
Reg.	BIT	41	49	51	59	61	69	71	79
C	RES	81	89	91	99	A1	A9	B1	B9
CB ...	SET	C1	C9	D1	D9	E1	E9	F1	F9
Reg.	BIT	42	4A	52	5A	62	6A	72	7A
D	RES	82	8A	92	9A	A2	AA	B2	BA
CB ...	SET	C2	CA	D2	DA	E2	EA	F2	FA
Reg.	BIT	43	4B	53	5B	63	6B	73	7B
E	RES	83	8B	93	9B	A3	AB	B3	BB
CB ...	SET	C3	CB	D3	DB	E3	EB	F3	FB
(HL)	BIT	46	4E	56	5E	66	6E	76	7E
CB ...	RES	86	8E	96	9E	A6	AE	B6	BE
	SET	C6	CE	D6	DE	E6	EE	F6	FE

GRUPO 15 – INSTRUÇÕES DE MANIPULAÇÃO DE BLOCOS

São, no total, oito instruções de manipulação de blocos, do conjunto de instruções do microprocessador Z80. Estas instruções são muito úteis e muito interessantes, pois permitem ao programador mover um bloco de dados de uma área da memória para outra, ou procurar uma área da memória.

Para mover um bloco de dados, o endereço base deve estar armazenado no par de registros HL, o endereço de destino deve estar armazenado no par de registros DE, e o número de bytes do bloco, ou seja, o seu comprimento deve estar armazenado no par de registros BC.

Para procurar na memória a primeira ocorrência de um determinado valor, o endereço base também deve estar armazenado no par de registros HL, o número de bytes da área de pesquisa deve estar no par de registros BC, e o registro A deve armazenar uma cópia do valor a ser encontrado.

As instruções deste grupo são:

	Mnemônica	Código hexa	Comentários
AUTO-MÁTICAS	LDIR	EDB0	Mover bloco – incrementa
	LDDR	EDB8	Mover bloco – decrementa
	CPIR	EDB1	Procura bloco – incrementa
	CPDR	EDB9	Procura bloco – decrementa
NÃO AUTO-MÁTICAS	LDI	EDA0	Mover byte – incrementa
	LDD	EDA8	Mover byte – decrementa
	CPI	EDA1	Compara byte – incrementa
	CPD	EDA9	Compara byte – decrementa

Como você deve ter notado, existem instruções automáticas e instruções não-automáticas. As automáticas são geralmente mais úteis, mais rápidas e, portanto, mais utilizadas.

Vamos analisar cada instrução detalhadamente.

INSTRUÇÕES AUTOMÁTICAS

1. **LDIR**: load location (DE) with location (HL), increment DE, HL, decrement BC and repeat until BC = 0. Esse é o nome em inglês. Traduzindo, essa instrução, LDIR, vai mover um bloco de dados cujo endereço fonte está armazenado no par de registros HL, para uma área da memória, cujo endereço

inicial está armazenado em DE (DEstination = destino...), e o comprimento desse bloco está armazenado no par de registros BC.

Quando em operação, um simples byte é transferido de HL, ou seja, para onde HL está apontando, para DE, o destino. O valor armazenado no par BC é então decrementado, e os valores armazenados nos pares HL e DE são incrementados. Enquanto o contador de bytes BC não chegar a zero, o processo será repetido.

Esta instrução reseta a bandeira de paridade/excesso.

2. **LDDR**: load location (DE) with location (HL), decrement DE, HL and BC and repeat until BC = 0.

Repare nos títulos desta e da instrução anterior e note qual a única diferença entre elas. Enquanto a primeira incrementa DE e HL, esta decrementa estes dois pares de registros, juntamente com o contador de bytes, BC.

Portanto, esta instrução requer como endereço base do bloco a sua última locação.

CPIR: Compare location (HL) and Accumulator, increment HL, decrement BC and repeat until BC = 0.

Esta instrução procura, em uma área específica da memória, pela ocorrência de um determinado valor, pela primeira vez. O par de registros HL deve armazenar o endereço base; o par BC deve armazenar o número de bytes a serem pesquisados; o acumulador (registro A) deve armazenar o valor, em particular, a ser pesquisado.

Em operação, o byte armazenado no par HL é comparado com aquele armazenado no registro A. Se a comparação não for verdadeira, então a instrução decrementa o contador de bytes e incrementa o endereço base mantido no par HL para proceder à próxima comparação.

A operação continua até uma delas ter resultado verdadeiro, ou seja, o conteúdo do endereço apontado por HL ser igual ao conteúdo do acumulador, ou então, se não acontecer a verdade, a operação termina quando BC atingir o valor 0. Nesse caso, as bandeiras zero e de paridade/excesso serão resetadas.

CPDR: Compare location (HL) and accumulator, decrement HL and BC repeat until BC = 0.

A operação desenvolvida por esta instrução é similar à anterior, com a única diferença que o bloco de dados é pesquisado a partir do seu último endereço.

Instruções não-automáticas

1. **LDI**: Load location (DE) with location (HL), increment DE, HL, decrement BC.

A execução desta instrução faz com que um byte simples armazenado no endereço apontado por (HL) seja movido, ou transferido para o endereço apontado por (DE). O valor armazenado no par BC é decrementado. A bandeira de paridade/excesso será setada, a menos que o par BC assuma o valor 0. Os valores nos pares HL e DE são incrementados.

2. **LDD**: Load location (DE) with location (HL), decrement DE, HL and BC.

A única diferença desta instrução para a anterior é que esta decrementa os pares de registros DE e HL, em vez de incrementá-los.

3. **CPI**: Compare location (HL) and accumulator, increment HL and decrement BC.

A execução desta instrução vai fazer com que o byte endereçado pelo par HL seja copiado no microprocessador e armazenado, enquanto o valor em HL é incrementado e o valor em BC é decrementado. O valor armazenado no microprocessador é então comparado com o valor do acumulador. Se o resultado da comparação for verdadeiro, então a bandeira zero é setada, e, de outra forma, ela é resetada. A bandeira de sinal é resetada e a bandeira de paridade/excesso também é resetada, até que o valor em BC atinja zero, quando ela passa a ser setada.

4. **CPD**: Compare location (HL) and accumulator, decrement HL and BC.

Esta instrução é similar à instrução CPI, exceto que o valor armazenado no par de registros HL é decrementado.

GRUPO 16 — INSTRUÇÕES DE ENTRADA E SAÍDA (INPUT/OUTPUT)

Estas instruções de entrada ou de saída de dados (bytes) permitem que o programador receba dados de uma fonte externa (IN) ou envie dados para um dispositivo externo (OUT).

São instruções simples, não-automáticas e automáticas.

Em todos os casos de instruções de entrada/saída, os dados manipulados são bytes de 8 bits enviados em paralelo.

Quando o microprocessador está executando uma instrução IN, ele pega o byte determinado nas vias de dados e copia-o em um determinado registro. A via de controle $\overline{IO}/\overline{QR}$ é ativada, bem como a linha \overline{RD} durante a sua execução.

Quando está executando uma instrução OUT, o microprocessador coloca uma cópia do valor armazenado em um registro especificado nas vias de dados, de onde ele será coletado por um dispositivo externo. As vias \overline{IORQ} e \overline{WR} se tornarão ativas durante sua execução.

Em adição ao estado das vias \overline{RD} , \overline{WR} e \overline{IORQ} , um dispositivo externo também será ativado pelo uso de um endereço apropriado colocado nas vias de endereçamento, durante a execução de instruções tipo IN ou OUT. Este endereço é denominado "porta de endereço" que no caso do Z80 é um endereço de 16 bits.

As instruções deste grupo são:

Mnemônica	Código hexa	I/O Reg.	Alto E.P.	Baixo E.P.
IN A, (dd)	DBdd	A	A	dd
IN A, (C)	ED78	A	B	C
IN H, (C)	ED60	H	B	C
IN L, (C)	ED68	L	B	C
IN B, (C)	ED40	B	B	C
IN C, (C)	ED48	C	B	C
IN D, (C)	ED50	D	B	C
IN E, (C)	ED58	E	B	C
OUT (dd), A	D3dd	A	A	dd
OUT (C), A	ED79	A	B	C
OUT (C), H	ED61	H	B	C
OUT (C), L	ED69	L	B	C
OUT (C), B	ED41	B	B	C
OUT (C), C	ED49	C	B	C
OUT (C), D	ED51	D	B	C
OUT (C), E	ED59	E	B	C

Nessa listagem, Alto E.P. quer dizer byte mais significativo do endereço da porta, e Baixo E.P. quer dizer byte menos significativo do endereço da porta.

As instruções automáticas e não-automáticas são:

Mnemônica	Código hexa	Comentários
INI	EDA2	Não-automática – Incrementa
INIR	EDB2	Automática – Incrementa
IND	EDAA	Não-automática – Decrementa
INDR	EDBA	Automática – Decrementa
OUTI	EDA3	Não-automática – Incrementa
OUTIR	EDB3	Automática – Incrementa
OUTD	EDAB	Não-automática – Decrementa
OUTDR	EDBB	Automática – Decrementa

As mnemônicas querem dizer:

INI carrega locação apontada pelo par HL, com entrada da porta (C), incrementa HL e decrementa B.

INIR carrega locação apontada pelo par HL, com entrada da porta (C), incrementa HL e decrementa B, repete até que B = 0.

IND carrega locação apontada pelo par HL, com entrada da porta (C), decrementa HL e B.

INDR carrega locação apontada pelo par HL, com entrada da porta (C), decrementa HL e decrementa B, até que B = 0.

OUTI carrega porta de saída (C) com locação apontada pelo par (HL), incrementa HL e decrementa B.

OUTIR carrega porta de saída (C) com locação apontada pelo par (HL), incrementa HL, decrementa B e repete até que B = 0.

OUTD carrega porta de saída (C) com locação apontada pelo par (HL), decrementa HL e B.

OUTDR carrega porta de saída (C) com locação apontada pelo par (HL), decrementa HL e B, e repete até que B = 0.

GRUPO 17 – INSTRUÇÕES DE INTERRUPTÃO

Existem, ao todo, sete instruções que permitem que o programador manipule o sistema de interrupções do microprocessador Z80. São elas:

Mnemônica	Código hexa
EI	FB
DI	F3
IM 0	ED46
IM 1	ED56
IM 2	ED5E
RETI	ED4D
RETN	ED45

Vamos analisar cada uma destas instruções.

EI (enable interrupt): Quando ligamos o microcomputador, um sistema de interrupção “mascarada” é habilitado para interromper o funcionamento do microprocessador Z80. Em outros termos, quando ligamos o microcomputador, o seu microprocessador imediatamente começa a trabalhar, executando as rotinas do seu sistema monitor. Nestas rotinas, necessariamente deve haver um sistema de interrupção do microprocessador, para que ele possa reconhecer alguma tecla que pressionemos, ou para que possamos dar-lhe algum comando em sua linguagem residente, qual seja, a Basic.

No TK 90X, o sistema de interrupção mascarada é usado para um relógio de tempo real, e para a rotina de reconhecimento de teclado, para saber se e qual tecla foi pressionada, e a interrupção é gerada a cada 1/60 segundos.

DI (disable interrupt): Em qualquer ponto de qualquer rotina em linguagem de máquina, o programador pode decidir “desligar” o sistema de interrupção mascarada, através desta instrução DI, o que torna o microprocessador insensível a qualquer sinal da linha INT. Através da utilização desta instrução, em alguns casos, chegamos a ganhar mais de 50% de tempo de processamento, já que não existe mais o reconhecimento do teclado.

IM0: São três modos de interrupção. Este modo é selecionado automaticamente pelo microprocessador quando ligamos o microcomputador pela primeira vez, ou também pela execução desta própria instrução. Mas, este modo de interrupção não é utilizado pelo TK 90X.

IMI: Este modo de interrupção é selecionado somente pela execução desta instrução, e é o modo utilizado pelo TK 90X. O programa monitor contido nos 16K da ROM do microcomputador possui esta instrução como parte da rotina de inicialização.

Neste modo, a instrução **RESTART 0038h** será sempre selecionada após receber um sinal da linha INT, o que significa que o sistema de interrupção mascarado foi habilitado. No TK 90X, a rotina em código de máquina, com início em 0038, atualiza o relógio de tempo real e faz o esquadramento do teclado (reconhecimento).

IM2: Este modo não é utilizado pelo TK 90X, mas, dos três modos de interrupção possíveis é o mais poderoso. Neste modo, um dispositivo periférico pode indicar ao microprocessador qual das 128 diferentes sub-rotinas deve ser executada após receber a interrupção mascarada. O conteúdo do registro I e o byte fornecido pelo dispositivo periférico são usados juntos para formar um endereço de 16 bits, que é utilizado para endereçar uma tabela de vetores, previamente preparada na memória.

RETI: Esta instrução é um "retorno" especial, para ser empregado em rotinas de interrupção mascaradas. O efeito desta instrução é o de retornar com a mesma interrupção mascarada depois de ter sido interrompido o processamento normal.

RETN: Esta instrução é similar à anterior, mas é aplicada no fim de uma rotina de interrupção não mascarada.

GRUPO 18 — INSTRUÇÕES DIVERSAS

Existem ainda seis instruções que não foram mencionadas. São elas:

Mnemônica	Código hexa
CPL	2F
NEG	ED44
SCF	37
CCF	3F
HALT	76
DAA	27

As mnemônicas significam:

CPL	—	Complementar acumulador
NEG	—	Negativo do acumulador (complemento de 2)
SCF	—	Seta bandeira de transporte
CCF	—	Complementa bandeira de transporte
HALT	—	Aguarde por uma interrupção ou por um resetar
DAA	—	Ajuste decimal do acumulador

Vamos analisá-las:

CPL: Esta é uma instrução simples que complementa o acumulador, ou seja, seta o bit que está resetado, e vice-versa. Esta operação é chamada complemento de 1. Não afeta bandeiras.

NEG: Esta instrução calcula o complemento de 2 do acumulador. As bandeiras de sinal e a zero dependem do resultado para serem alteradas. A bandeira de transporte será resetada se o valor original for zero, de outra forma será setada, e a bandeira de paridade/excesso será setada se o valor original for 80h.

SCF: Seta a bandeira indicadora de transporte.

CCF: Complementa a bandeira indicadora de transporte.

HALT: Esta é uma instrução especial que faz com que o microprocessador pare o seu trabalho até que ocorra uma interrupção. No TK 90X, as únicas interrupções que podem ocorrer são as mascaradas. O comando **PAUSE** usa deste artifício para contar 1/60 segundo.

DAA: Esta é a instrução que faz o ajuste decimal do acumulador. Em aritmética binária BCD (*binary coded arithmetic*), os algarismos de 0 a 9 são representados pelos "nibbles binários" 0000-1001, e os nibbles 1010-1111 não são utilizados. Portanto:

- o byte 0000 0000 representa o número 0;
- o byte 0011 1001 representa o número 39.

$$= 3 = 9$$

Esta instrução converte, portanto, bytes em sua forma binária absoluta para a forma BCD.

A bandeira indicadora de sinal e a bandeira zero são afetadas pelo resultado, e a bandeira de paridade/excesso será setada se houver paridade par. O

efeito na bandeira indicadora de transporte vai depender se houve excesso nas adições ou subtrações na forma BCD.

E assim nós encerramos este capítulo chato que trata de todas as instruções do microprocessador Z80. Não é à toa que ele é considerado o mais complicado dos microprocessadores de 8 bits.

As rotinas da ROM de 16K

O TK 90X tem uma ROM de 16K que está dividida em:

1. O sistema operacional
2. O interpretador Basic
3. O conjunto de 96 caracteres

Esta ROM ocupa os endereços de 0 a 16383, ou 0000 a 3FFFh, e não pode ser movida desta área, na versão padrão da máquina. A instrução em código de máquina no endereço 0 é a primeira a ser executada quando ligamos o micro pela primeira vez.

As sub-rotinas do programa monitor podem ser utilizadas pelo programador em seus próprios programas em linguagem de máquina, fazendo com que estes sejam relativamente mais curtos, por não repetirem certos processamentos, e conseqüentemente mais rápidos.

É muito instrutivo para os aficionados estudarem o programa monitor, para perceberem como um programa tão grande pode ser estruturado. Isso você faz através de um programa ferramenta denominado "Monitor & Disassembler", que mostra as mnemônicas dos endereços e seus respectivos códigos.

Vamos agora ver alguns detalhes da ROM.

SISTEMA OPERACIONAL — ROTINAS

1. Rotina de inicialização É a primeira rotina a ser executada, quando ligamos o micro. Ela ocupa os endereços 0000 a 0007h, e 11CBh a 12A1h.

Um dos propósitos principais desta rotina é o de checar o quanto de memória RAM está disponível para o usuário e acertar os devidos valores das variáveis de sistema (valores padrão).

2. Rotina de execução principal Esta rotina ocupa os endereços de 12A2h a 15AEh e é a dominante de todo o programa monitor, pois é ela que chama, se necessário, a rotina do comando LIST, a rotina do Editor, a rotina de checar sintaxe de linhas em Basic. No caso de se usar o micro em modo imediato, é chamada então a rotina de rodar linha (LINE RUN) para interpretá-la e executá-la. Se houver erro, então é feita uma referência a uma tabela de mensagens de erros, que ocupa os endereços 1391h a 1536h.

3. Rotina do Editor Ocupa os endereços 0F2Ch a 10A7h.

Esta rotina permite que o usuário construa uma linha em Basic, aparentemente na parte inferior da tela. Digo aparentemente porque na realidade a linha é formada na área de edição e então copiada, com símbolos expandidos, para a área da tela.

Uma única entrada para esta rotina é feita pela chamada da rotina Entrada do Teclado (KEYBOARD INPUT).

4. Entrada do teclado (KEYBOARD INPUT) Esta rotina ocupa os endereços 10A8h a 111Ch.

Ela pega o código da última tecla pressionada pela leitura da variável de sistema KEYPRS, de endereço 23560d.

5. Teclado O esquadrinhamento do teclado é uma interrupção que ocorre a cada 1/60 segundos. São cinco sub-rotinas separadas envolvidas pela execução da rotina principal de teclado, que ocupa os endereços 02BFh a 030Fh.

O esquadrinhamento do teclado é feito pela sub-rotina KEY SCAN, nos endereços 028Eh a 02BEh. Esta sub-rotina retorna o valor do código da tecla no par de registros DE, para que outras rotinas dele se utilizem.

6. Saída de dados (PRINT OUTPUT) A rotina de impressão de saída de dados na tela que ocupa os endereços 09F4h a 0D4Ch é uma das mais importantes do programa monitor.

Esta rotina é em efeito a rotina chamada pela instrução em código de máquina RST 0010h. O endereço 09F4h é obtido através da área de informação de canais.

A instrução RST 0010h, ou a rotina PRINT OUTPUT vai permitir que o caractere, cujo código está armazenado no registro A, seja impresso ou na tela da TV, ou na impressora standard. A rotina testa uma determinada bandeira indicadora de estado de uma variável de sistema, para saber qual caminho de saída deve ser adotado. Esta é uma rotina muito poderosa, visto que manipula tanto códigos de caracteres, quanto códigos de controle.

No caso da saída de dados ser pela TV, a posição de impressão é coletada nas variáveis de sistema apropriadas, utilizadas e armazenadas novamente no lugar de onde foram retiradas. A posição de impressão indica o número da linha e o número da coluna da área a ser utilizada, bem como o endereço correspondente no arquivo de imagem.

Esta rotina possui sub-rotinas que são chamadas, se necessário, para transferir os 64 bits de um caractere da área de conjunto de caracteres para a locação requisitada no arquivo de imagens.

Outra sub-rotina altera o byte dos atributos daquela área de caractere, de acordo com o valor armazenado nas variáveis de sistema apropriadas.

INTERPRETADOR BASIC

A rotina de interpretação de comandos do programa monitor é chamada tanto para checar sintaxe quanto para execução de linhas.

As diferentes partes do interpretador Basic são:

1. Tabela de comandos Esta tabela é encontrada nos endereços 1A48h a 1B16h.

No TK 90X são 51 comandos Basic e esta tabela contém detalhes de cada comando, os caracteres separadores e os endereços das rotinas de execução desses comandos.

2. Rotina controladora Contida nos endereços 1B17h a 1C00h, esta rotina contém as instruções de controle que permitem que o interpretador Basic passe de uma declaração Basic para outra, conforme requerido pelo programa.

3. **Rotinas das classes de comandos** Estas rotinas, nos endereços de 1C01h a 1CDDh são principalmente concebidas para a análise dos parâmetros que seguem os comandos em Basic.

4. **Rotinas dos comandos** Muitas das rotinas de execução dos comandos em Basic estão contidas nos endereços 1CDEh a 24FAh. Existe uma rotina de execução para cada um dos comandos, e a sua execução é a essência da interpretação Basic.

5. **Avaliador de expressões** Esta rotina ocupa os endereços 24FBh a 28B1h.

O resultado obtido pela avaliação de uma expressão qualquer tanto pode ser numérico, quanto alfanumérico. Um resultado numérico será retornado pelo avaliador de expressões como um número de 5 bytes em representação de ponto flutuante, colocado no topo da pilha do calculador. No caso de um resultado alfanumérico os 5 bytes vão representar um conjunto de parâmetros que descrevem a cadeia de caracteres.

6. **Rotinas de manipulação de variáveis** Este conjunto de rotinas ocupa a área da ROM desde o endereço 2813h até o endereço 2ACBh.

Estas rotinas retornam o valor corrente dos parâmetros de uma dada variável armazenada na área de variáveis da RAM.

A CALCULADORA

Esta grande e variada, bem como, complicada rotina ocupa os endereços 2F9Bh a 386Dh. É normalmente chamada pela instrução em linguagem de máquina RST 0028, que atua como um salto indireto para o endereço 335Bh. No total são 66 sub-rotinas, onde cada uma desempenha a sua função. A chamada destas sub-rotinas normalmente não se faz através da instrução em linguagem de máquina CALL, mas sim pelo uso de valores hexadecimais literais entre 00h e 41h.

Por exemplo, o valor literal 04 equivale à multiplicação; o valor literal 17 equivale à concatenação de cadeias de caracteres.

Estes valores literais são incluídos em uma rotina em linguagem de máquina, como bytes de dados (DEFB's = defined bytes é como você vai encontrar em rotinas de linguagem de máquina) que seguem a instrução RST 0028. O byte final é sempre 38, que desenvolve uma operação de fim de cálculo, e atua como uma saída, um retorno da calculadora.

AS DIFERENTES PARTES DO PROGRAMA MONITOR

A seguir, mostrarei alguns endereços e áreas das principais rotinas do programa monitor, na medida em que eles ocorrem na ROM.

0000-0007 — RST 0000h — início — equivale a RAND USR 0, que desabilita interrupções mascaradas, limpa o registro A, carrega o par de registros DE com FFFFh e salta para o endereço 11CBh.

0008-000F — RST 0008h — rotina de erros, que traz a mensagem apropriada quando ocorreu um erro.

0010-0012 — RST 0010h — rotina de saída de dados.

0018-0024 — RST 0018h e RST 0020 — procura pelo caractere em uso.

0028-0029 — RST 0028h — chamada calculadora.

0030-0037 — RST 0030 — rotina para liberar área de trabalho da memória.

0038-0052 — rotina de interrupção mascarada.

0066-0072 — rotina de interrupção não mascarada.

0095-0204 — tabela de símbolos.

0205-028D — tabela de teclas; no total são 6 tabelas, uma para cada modo permitido.

028E-02BE — sub-rotina de esquadrinhamento do teclado.

02BF-03B4 — sub-rotinas do teclado.

03B5-03F7 — sub-rotina de som.

03F8-046D — rotina do comando SOUND.

04C2-09F3 — rotinas dos comandos SAVE, LOAD, VERIFY e MERGE.

09F4-0D4C — rotina de saída de dados.

0D4D-0D6A — sub-rotina de conjuntos de cores temporárias.

06DB-0EAB — rotina do comando CLS.

0EAC-0F2B — rotinas da impressora (comandos COPY, LLIST e LPRINT).

0F2C-10A7 — editor.

10A8-111C — sub-rotina de entrada do teclado.

11B7-11CA — rotina do comando NEW.

11CB-12A1 — rotina de inicialização.

12A2-15AE — rotina de execução principal.

15AF-15C5 — tabela de dados de canais.
 15C6-15D2 — tabela de dados de fluxo.
 15D4-1651 — rotina de acesso aos canais.
 1652-16E4 — sub-rotinas diversas.
 16DB-16E5 — sub-rotinas de indexação de tabelas.
 16E6-1792 — rotinas dos comandos OPEN e CLOSE.
 1795-1A47 — rotinas de listagem.
 1A48-1B16 — duas tabelas de comandos. A primeira é uma tabela de comandos que indexa, a segunda, uma tabela de parâmetros.
 1B17-1C00 — rotinas de controle do interpretador Basic.
 1C01-1C0C — tabela de classes de comandos.
 1C0D-1CDD — rotinas de classes de comandos.
 1CDE-24FA — rotinas dos comandos.
 24FB-28B1 — avaliador de expressões.
 2AFF-2BF0 — rotina do comando LET.
 2C02-2C87 — rotina do comando DIM.
 2C88-2F9A — diversas rotinas aritméticas.
 2F9B-386D — sub-rotina calculadora.
 386E-3CFF — conjunto de caracteres.

Como utilizar rotinas do programa monitor

O objetivo deste capítulo é o de mostrar como se pode escrever rotinas em linguagem de máquina de uma maneira relativamente fácil, através da utilização das diversas sub-rotinas que estão sempre disponíveis para o usuário, no programa monitor.

Você pode fazer uso de um programa Basic carregador de códigos de máquina, que lhe permite digitar os códigos hexa da sua listagem mnemônicas, para que ele automaticamente converta esses valores em números decimais (bytes) e armazene-os nos endereços que você determinar.

Eis a listagem de um carregador hexa muito simples, mas que nem por isso deixa de ser útil.

```

1 REM Carregador de codigos h
2
3
4
5
6
7
8
9
10 POKE 23558,5
11 INPUT "Endereco inicial " ; e
12 LET inicio=end
13 INPUT "Entre com o valor em
14 hexa " ; h$
15 IF h$(1)="#" THEN LET end=e
16 GO TO 30
17 IF h$(1)="s" THEN STOP
18 IF h$(1)="l" THEN INPUT "No
19 me do arquivo " ; y$ : SAVE y$CODE
20 inicio,(end-inicio)
21 PRINT end;TAB 10;h$;TAB 20;
22 LET byte=16*(CODE h$(1)-48-

```



```

17 AND h$(1) > "9") + (CODE h$(2) - 4
8 - (17 AND h$(2) > "9"))
60 PRINT butt
70 POKE end, butt
80 LET end = end + 1
90 GO TO 30

```

Vamos ver então algumas rotinas de execução de comandos:

1. **Rotina do comando SOUND** Em Basic, o comando SOUND tem o formato:

SOUND duração, tom

Onde a duração deve ser um número positivo menor que 10 e o tom deve ser um número positivo ou negativo, porém dentro da faixa de abrangência do comando, em função do DO central.

Em linguagem de máquina existem 2 maneiras de se produzir um SOUND. A primeira é chamando a sub-rotina SOUNDER, com os valores apropriados armazenados nos pares de registros DE e HL; a segunda é chamando a rotina de comando SOUND com os valores para duração e tom armazenados na pilha do calculador.

Qualquer rotina que você elabore em linguagem de máquina pode ser armazenada, a princípio, em qualquer lugar da memória.

A princípio porque um programa Basic que execute aquela rotina ocupa uma certa área, e seus códigos de máquina devem começar após o término do programa Basic.

Através das variáveis de sistema apropriadas, você pode determinar onde começa a área de memória desocupada.

A fim de padronizarmos nossas rotinas deste livro, eu sugiro como endereço inicial da memória, para armazenarmos as rotinas em linguagem de máquina, a locação decimal 65000, que vamos converter para hexadecimal:

```

65000 | 256
-512 253
1380
-1280
1000
- 768
232

```

Portanto, $65000 = 253 \times 256 + 232$.

Consultando o Apêndice A deste livro, temos para

253 = FD e
232 = E8

Portanto, 65000d = FDE8h

Conferindo, através do processo inverso:

F	D	E	8	
↓	↓	↓	↓	
				$8 \times 16^0 = 8 \times 1 = 8$
				$14 \times 16^1 = 14 \times 16 = 224$
				$13 \times 16^2 = 13 \times 256 = 3328$
				$15 \times 16^3 = 15 \times 4096 = 61440$
				<u>65000 e confere!</u>

Podemos converter de outra forma:

```

65000 | 256
-512 253
1380
-1280
1000
- 768
232

232 | 16
-16 14
72
-64
8

```

Portanto 1ª casa = 8d = 8h
2ª casa = 14d = Eh

```

253 | 16
-16
93
-80
13

```

3ª casa = 13d = Dh
4ª casa = 15d = Fh

Ou seja, FDE8h!

Guarde muito bem esses métodos de conversão, pois às vezes você está no meio da digitação de uma rotina qualquer e esqueceu de converter um só valor, e não tem condições de carregar um programa que faça a conversão para você.

Vamos, então, chamar este endereço FDE8 de END, e voltemos ao SOUND.

19 método:

Endereço	Código de máquina	Mnemônica	Comentário
END	110501	LD DE, 0105	Duração 1 segundo
END + 3	216606	LD HL, 0666	Tom DO central
END + 6	CDB503	CALL SOUNDER	Aciona SOUND
END + 9	C9	RET	Retorna à Basic

Os valores para DE e HL são assim calculados:

— considere uma nota de frequência f ; por exemplo, DO central tem a frequência de 261.63 Hz;

— a duração requerida para um período p é calculada por " $f \times p$ ". Este valor é colocado no par DE;

— o valor a ser armazenado no par HL é dado por:

$$437.500/f - 30125.$$

Note que com este método não existe a limitação de 10 segundos de duração.

29 método:

Endereço	Código de máquina	Mnemônica	Comentário
END	3E01	LD A, 01	Duração de 1 segundo.
END + 2	CD282D	CALL STACK A	Coloca na pilha da calculadora.
END + 5	3E00	LD A, 00	Tom DO central.
END + 7	CD282D	CALL STACK A	Coloca na pilha da calculadora.
END + 10	CDF803	CALL SOUND	Aciona SOUND.
END + 11	C9	RET	Retorna à Basic.

2. Rotinas de SAVE e LOAD As sub-rotinas SAVE bytes e LOAD bytes podem ser chamadas em linguagem de máquina de uma maneira muito particular.

Em ambos os casos, o par de registros IX deve armazenar o endereço destino, e o par de registros DE, o contador de bytes.

O registro A deve armazenar FF para mostrar que um bloco de dados está sendo movimentado.

Finalmente, para LOAD, a bandeira indicadora de transporte deve ser setada.

A rotina a seguir executa o comando SAVE bytes.

Endereço	Código de máquina	Mnemônica	Comentário
END	3EFF	LD A, FF	Significa bloco de dados.
END + 2	DD21dd	LD IX, dd (início)	Carrega par IX c/endereço destino.
END + 6	11xx	LD DE, xx (comprim)	Carrega par DE c/quantidade de bytes.
END + 9	CDC204	CALL SAVE byte	Executa SAVE.
END + 12	C9	RET	Retorna à Basic.

Note que os bytes serão gravados sem o cabeçalho (*header*), que dá as informações sobre o programa, e conseqüentemente este bloco só será carregado por LOAD se o contador de bytes for conhecido.

A rotina para LOAD:

Endereço	Código de máquina	Mnemônica	Comentário
END	37	SCF	Seta bandeira de transporte (será resetado para VERIFY).
END + 1	3EFF	LD A, FF	Significa bloco de dados.
END + 3	DD21dd	LD IX, dd	Carrega par IX.
END + 7	11dd	LD DE, dd	Carrega par DE.
END + 10	CD5605	CALL LOAD bytes	Executa LOAD.
END + 13	C9	RET	Retorna à Basic.

Uma pequena nota sobre o cabeçalho (header)

Ele tem 17 bytes de comprimento, que respectivamente informam:

- Tipo - 1 byte	Contém o tipo de arquivo: 0 = programa Basic 1 = matriz numérica 2 = matriz alfanumérica 3 = códigos de máquina
- Nome - 10 bytes	Contém o nome do arquivo.
- Comprimento - 2 bytes	Contém o comprimento do arquivo.
- Endereço	
Inicial - 2 bytes	Contém o número da linha para AUTO. RUN ou o endereço inicial dos códigos.
- ? - 2 bytes	?

Aqui já dá para você mesmo escrever um programa copiador ...

3. Rotinas dos comandos de cores - no TK 90X todos os bytes de atributos têm o seguinte formato:

BIT 7 - FLASH
BIT 6 - BRIGHT
BIT 5 a 3 - PAPER
BIT 2 a 0 - INK

O comando BORDER e os seis comandos de cores serão discutidos agora.

3.1. BORDER - a qualquer hora, a cor da borda pode ser alterada pelo uso da instrução

OUT (FE), A

A rotina mais simples para alterar BORDER:

Endereço	Código de máquina	Mnemônica	Comentário
END	3E01	LD A, "blue"	Cor azul = 1
END + 2	CD9B22	CALL BORD-1	Altera borda (cor)
END + 5	C9	RET	Retorna à Basic

Note que com esta rotina, a cor da tinta passa a ser contrastante em relação à cor da borda.

3.2. PAPER - a cor permanente do papel é dada pelos bits 5 a 3 da variável de sistema ATCLR P. A rotina abaixo mostra a alteração de PAPER 0 - PAPER 7 apenas uma vez na qual os bits podem ser alterados sem se alterar os outros valores de atributos.

Endereço	Código de máquina	Mnemônica	Comentário
END	3A8D5C	LD A, (5C8D)	Carrega acumulador com valor da variável de sistema.
END + 3	OF	RRCA	Move os bits 5 a 3 para bits 2 a 0.
END + 4	OF	RRCA	
END + 5	OF	RRCA	
END + 6	E6F8	AND F8	Descarta cor antiga.
END + 8	C604	ADD A, "green"	Carrega cor verde.
END + 10	07	RLCA	
END + 11	07	RLCA	Roda o byte três vezes para a esquerda.
END + 12	07	RLCA	
END + 13	328D5C	LD (5C8D), A	Restaura variável de sistema.
END + 16	C9	RET	Retorna à Basic.

3.3. INK - a cor da tinta permanente é dada pelos bits 2 a 0 da variável de sistema armazenada no endereço 5C8D. A rotina a seguir mostra como esses bits podem ser alterados para INK 0 a INK 7.

Endereço	Código de máquina	Mnemônica	Comentário
END	3EF8	LD A, F8	Prepara máscara.
END + 2	FDA653	AND (53A6)	Busca bits 7 a 3 da variável de sistema em 53A6.
END + 5	C602	ADD A, red	Tinta vermelha.
END + 7	328D5C	LD (5C8D), A	Restaura variável de sistema.
END + 10	C9	RET	Retorna à Basic.

3.4. FLASH - os três estados de FLASH podem ser assim programados:

FLASH 0 - reseta bit 7 de ATCLR P "RES 7, (ATCLR P)"
FLASH 1 - seta bit 7 de ATCLR P "SET 7, (ATCLR P)"
FLASH 8 - seta bit 7 de MASKCLRP "SET 7, (MASKCLRP)"

Similarmente, os três estados de BRIGHT:

BRIGHT 0 — reseta bit 6 de ATCLR P "RES 6, (ATCLR P)"
 BRIGHT 1 — seta bit 6 de ATCLR P "SET 6, (ATCLR P)"
 BRIGHT 8 — seta bit 6 de MASKCLRP "SET 6, (MASKCLRP)"

Os dois estados de OVER e INVERSE:

OVER 1 — seta bit 1 de SFLAG4 "SET 1, (SFLAG4)"
 OVER 0 — reseta bit 1 de SFLAG4 "RES 1, (SFLAG4)"

INVERSE 1 — seta bit 3 de SFLAG4 "SET 3, (SFLAG4)"
 INVERSE 0 — reseta bit 3 de SFLAG4 "RES 3, (SFLAG4)"

4. Rotinas dos comandos CLS e SCROLL — uma das grandes vantagens na utilização da linguagem de máquina ao invés de Basic é que o usuário não está limitado à sintaxe dos comandos Basic apenas.

No programa monitor do TK 90X existem sub-rotinas para limpar apenas parte da tela, bem como para fazer rolar apenas algumas linhas dela (scroll). Mas, note que estas sub-rotinas somente podem ser utilizadas por programas em linguagem de máquina.

4.1. CLS — a operação completa de limpar a tela e atribuir a todos os bytes de atributos os valores permanentes é obtida por:

CALL CLS ou mnemônica CALL 0D6B, mas é importante destacar que o canal "S" (de screen) deve ser aberto antes da chamada dessa sub-rotina. Veja como:

Endereço	Código de máquina	Mnemônica	Comentário
END	3E02	LD A, 02	Abre canal "S".
END + 2	CD0116	CALL CHAN OPEN	
END + 5	CD6B0D	CALL CLS	Limpa a tela.
END + 8	C9	RET	Retorna à Basic.

Outra forma, onde se pode limpar apenas parte da tela é utilizando-se a sub-rotina CL LINE.

Antes de se chamar esta sub-rotina, o registro B deve armazenar um valor na faixa 01 a 18h, onde 18h significa limpar a tela toda (24 linhas), assim como o canal S também deve ser aberto.

A rotina:

Endereço	Código de máquina	Mnemônica	Comentário
END	0617	LD B, 17	17h = 23d limpa 23 linhas, deixando apenas a superior.
END + 2	CD440E	CALL CL LINE	Limpa a tela.
END + 5	C9	RET	Retorna à Basic.

4.2. SCROLL — esta é uma sub-rotina muito interessante que permite ao usuário "rolar" somente algumas linhas ou a tela toda.

O registro B é novamente utilizado para armazenar o número de linhas a ser rolado, mas aqui, é o número atual de linhas menos uma, isto é, na faixa de 01 a 17h (um mínimo de duas linhas é requerido por esta sub-rotina).

Endereço	Código de máquina	Mnemônica	Comentário
END	0616	LD B, 16	Deixa de rolar somente a primeira linha.
END + 2	CD000E	CALL CL SCROLL	
END + 5	C9	RET	Retorna à Basic.

5. Sub-rotinas de saída de dados — para se utilizar qualquer sub-rotina de impressão de caracteres ou mensagens na tela, é preciso, antes de mais nada, abrir-se o canal "S" (de screen). Como vimos anteriormente, para isso basta:

LD A, 02
 CALL CHAN OPEN

5.1. Rotina RST 0010 — no TK 90X toda e qualquer impressão de caracteres, ou mensagens na tela é feita usando-se esta instrução. Com o canal S aberto, ela tem o efeito de usar a rotina PRINT OUTPUT, que começa no endereço 09F4, como rotina de saída.

A instrução RST 0010 é muito poderosa e pode ser utilizada na impressão de qualquer caractere, para a alteração da posição corrente de impressão pelo uso de AT e TAB, para impressão de palavras expandidas, bem como nos itens de cores provisórias.

A rotina abaixo mostra esses usos:

Endereço	Código de máquina	Mnemônica	Comentários
END	3E02	LD A, 02	
END + 2	CD0116	CALL CHAN OPEN	Abre canal "S".
END + 5	0618	LD B, 18	
END + 7	CD440E	CALL CL LINE	Limpa toda a tela.
END + 10	3E16	LD A, "AT"	
END + 12	D7	RST 0010	
END + 13	3E05	LD A, 05	
END + 15	D7	RST 0010	Similar a PRINT AT 5, 0
END + 16	3E00	LD A, 00	
END + 18	D7	RST 0010	
END + 19	3E41	LD A, "A"	PRINT "A"
END + 21	D7	RST 0010	
END + 22	3E0D	LD A, "ENTER"	PRINT
END + 24	D7	RST 0010	
END + 25	3EF9	LD A, CHR\$ 249	PRINT CHR\$ 249
END + 27	D7	RST 0010	
END + 28	3E0D	LD A, "ENTER"	PRINT
END + 30	D7	RST 0010	
END + 31	3E11	LD A, "PAPER"	
END + 33	D7	RST 0010	
END + 34	3E02	LDA, "RED"	Similar a PAPER 2
END + 36	D7	RST 0010	
END + 37	3E06	LD A, ","	PRINT,
END + 39	D7	RST 0010	
END + 40	3E42	LD A, "B"	PRINT "B"
END + 42	D7	RST 0010	
END + 43	C9	RET	Retorna à Basic

Note nessa sub-rotina, que é feita uma chamada para a sub-rotina CL LINE, que limpa a tela, mas que isto não leva a posição de impressão para a posição padrão, ou seja, linha 0 e coluna 0 (mas CALL CLS leva).

Quando usando a instrução RST 0010 para imprimir um item de cor, duas chamadas separadas são necessárias, e quando usando "AT" ou "TAB" como delimitadores, três chamadas são necessárias.

Um método alternativo para alterar a posição de impressão é o seguinte:

– carregar o par de registros BC com os devidos valores da nova posição de impressão;

– CALL CL SET & CALL ODD9, que coloca os valores de posicionamento nas variáveis de sistema HV POS e DFPSRT.

Os valores para o par de registros BC para uma determinada posição de impressão "AT x, y" são, respectivamente, para o registro B, $(18 - x)h$ e para o registro C, $(21 - y)h$.

Por exemplo:

PRINT AT 5,0 requer:

LD BC, 1321

CALL ODD9

5.2. Impressão de série de caracteres – uma cadeia de caracteres, para ser impressa, deve ter sempre no par de registros DE o endereço do primeiro caractere dela, e o par de registros BC deve armazenar um valor igual ao seu número total de caracteres.

Na rotina do comando PRINT, a sub-rotina PR STRING é utilizada para imprimir qualquer série de caracteres.

Os detalhes da sub-rotina são:

PR STRING equ. 203C

NOTA: Esta primeira linha é aquela notação citada anteriormente, que associa o nome determinado ao início de uma sub-rotina específica em linguagem de máquina.

PR STRING	LD A, B OR C	Dependendo do comprimento da série de caracteres em questão, o seu valor está em B ou em C.
	DEC	Decrementa o contador.
	RET Z	Se o contador for zero, então fim.
	LD A, (DE)	Carrega acumulador com caractere.
	INC DE	Passa para o próximo caractere.
	RST 0010	Imprime o caractere.
	JR PR STRING	Retorna ao início para outra impressão.

Portanto, qualquer série de caracteres pode ser impressa desta outra maneira:

- carregar seu endereço inicial no par de registros DE;
- carregar seu comprimento no par de registros BC;
- CALL PR STRING – CALL 203C.

Como exemplo, digite a rotina abaixo:

Endereço	Código de máquina	Mnemônica	Comentários
END	11 01 10 09 16 0A 05 41 52 51 55 49 54 52 4F 4E 20 49 4E 46 4F 52 4D 41 54 49 43 41 20 4C 54 44 41 20 20 20 20 20 20 41 56 20 42 52 49 47 20 46 41 52 49 41 20 4C 49 4D 41 2C 20 31 35 38 30 20 43 4A 20 32 32		Código da série de caracteres.
END + 62	3E02	LD A, 02	
END + 64	CD0116	CALL CHAN OPEN	Abre canal "S".
END + 67	0618	LD B, 18	
END + 69	CD440E	CALL CL LINE	Limpa a tela.
END + 72	11 END	LD DE, END	Endereço inicial da série de caracteres.
END + 75	014500	LD BC, 0045	Comprimento da série.
END + 78	CD3C20	CALL PR STRING	Imprime a mensagem.
END + 81	C9	RET	Retorna à Basic.

5.3. Imprimindo números – a sub-rotina PRINT FP (FP de floating point – número em ponto flutuante) é usada para imprimir em formato decimal qualquer número de formato em ponto flutuante de 5 bytes. Esta sub-rotina considera como operando a última entrada na pilha da calculadora. Após isso, o valor é removido da pilha e abandonado.

A rotina a seguir usa esta sub-rotina para imprimir PI/2 a partir da tabela de constantes da ROM.

Endereço	Código de máquina	Mnemônica	Comentários
END	3E02	LD A, 02	
END + 2	CD0116	CALL CHAN OPEN	Abre canal S.
END + 5	CD6B0D	CALL CLS	Limpa a tela.
END + 8	3E02	LD A, 02	Reabre canal S.
END + 10	CD0116	CALL CHAN OPEN	
END + 13	EF	RST 0028	Chama a calculadora.
END + 14	A3	DEFB A3	Pega a quarta constante.
END + 15	38	DEFB 38	Final de cálculo.
END + 16	CDE32D	CALL PRINT FP	Imprime o resultado.
END + 19	C9	RET	Retorna à Basic.

O uso da linha, em modo direto, RAND USR END tem o mesmo efeito que PRINT PI/2.

Existe outra sub-rotina para imprimir números, só que na faixa decimal, de 0 a 9999, pois esta sub-rotina é empregada para produzir números de linhas de programas em listagens, ou em mensagens da parte inferior da tela. Quando em uso, esta sub-rotina requer que o número em questão esteja armazenado no par de registros BC, na ordem normal, ou seja, "byte alto/byte baixo" (mais significativo/menos significativo).

Por exemplo:

Endereço	Código de máquina	Mnemônica	Comentários
END	3E02	LD A, 02	
END + 2	CD0116	CALL CHAN OPEN	Abre canal S.
END + 5	CD6B0D	CALL CLS	Limpa toda a tela.
END + 8	3E02	LD A, 02	
END + 10	CD0116	CALL CHAN OPEN	Reabre canal S.
END + 13	010F27	LD BC, 270F	Carrega BC com o valor 9999d.
END + 16	CD1B1A	CALL OUT NUM	Imprime o número.
END + 19	C9	RET	Retorna à Basic.

6. **PLOT, DRAW e CIRCLE** — estes três comandos tratam com as pixéis na tela. Para as 22 linhas superiores, são no total 256 x 176 pontos.

As variáveis de sistema LSTPLOT e COORDS armazenam as coordenadas do último ponto plotado.

6.1. **PLOT** — destes três comandos, este é o mais fácil de ser manipulado, pois envolve somente a identificação de um simples bit no arquivo de imagens, e seu subsequente desenho ou não.

Existem três pontos de entrada na rotina do comando PLOT:

1. **CALL PLOT-CALL 22DC** — esta entrada requer os valores de x e y no topo da pilha da calculadora, sendo o valor de x abaixo do valor de y.

2. **CALL PLOT 1-CALL 22DF** — aqui, o registro B deve armazenar o valor y, e o registro C deve armazenar o valor de x.

3. **CALL PLOT BC-CALL 22E5** — novamente o registro B deve conter o valor de y, e o registro C deve conter o valor de x. É esta sub-rotina que realmente desenha o ponto na tela.

Por exemplo:

Endereço	Código de máquina	Mnemônica	Comentário
END	3E02	LD A, 02	
END + 2	CD0116	CALL CHAN OPEN	Abre canal S.
END + 5	CD6B0D	CALL CLS	Limpa toda a tela.
END + 8	3E02	LD A, 02	
END + 10	CD0116	CALL CHAN OPEN	Reabre canal S.
END + 13	016432	LD BC, 3264	
END + 16	CDE522	CALL PLOT BC	PLOT 100, 50.
END + 19	C9	RET	Retorna à Basic.

6.2. **DRAW** — para este comando, existem dois pontos de entrada:

1. **DRAW 1-CALL 2477** — requer os valores das coordenadas x e y no topo da pilha da calculadora.

2. **DRAW 2-CALL 24BA** — requer para o registro B, o valor absoluto de y, e, para o registro C, o valor absoluto de x. O registro D deve conter o sinal de x e o registro E deve conter o sinal de y.

A rotina a seguir ilustra esse processo. Note que o par de registros H'L' tem seu valor guardado (salvo), para ser recuperado mais tarde, caso necessário.

Endereço	Código de máquina	Mnemônica	Comentários
END	3E02	LD A, 02	
END + 2	CD0116	CALL CHAN OPEN	
END + 5	CD6B0D	CALL CLS	
END + 8	3E02	LD A, 02	
END + 10	CD0116	CALL CHAN OPEN	
END + 13	D9	EXX	Troca HL por H'L' e guarda na pilha da máquina.
END + 14	E5	PUSH HL	
END + 15	D9	EXX	
END + 16	FD364364	LD (LSTPLOT), 64	Últimos valores de coordenadas.
END + 20	FD364464	LD (COORDS), 64	B = 50 e C = 50.
END + 24	013232	LD BC, 3232	Sinais positivos.
END + 27	110101	LD DE, 0101	DRAW 50, 50.
END + 30	CDBA24	CALL DRAW 2	Recupera valor de HL.
END + 33	D9	EXX	
END + 34	E1	POP HL	
END + 35	D9	EXX	
END + 36	C9	RET	Retorna à Basic.

Essa rotina coloca, como últimas coordenadas, os valores 100 e 100 e então traça a linha 50, 50.

Uma variante do comando DRAW, é quando se quer desenhar um arco de circunferência. Para tanto, os três valores x, y e raio devem ser colocados na pilha da calculadora, para que então se faça a chamada para a sub-rotina DRAW ARC-CALL 2394.

A rotina seguinte mostra a execução de um comando DRAW 50, 50, 1 a partir de um ponto de coordenadas 100, 100.

Endereço	Código de máquina	Mnemônica	Comentários
----------	-------------------	-----------	-------------

END

:

:

IDÊNTICO À ANTERIOR

END + 20

END + 24	34	DEFB 34	Dado da pilha.
END + 25	40B00032	DEFBs	= 50 decimal.
END + 29	31	DEFB 31	"Duplica".
END + 30	A1	DEFB A1	Coloca na pilha.
END + 31	38	DEFB 38	Final de cálculo.
END + 32	CD9423	CALL DRAW ARC	Traça o arco.
END + 35	D9	EXX	
END + 36	E1	POP HL	
END + 37	D9	EXX	
END + 38	C9	RET	

6.3. Circle — mais uma vez, os três operandos do comando CIRCLE devem ser colocados na pilha da calculadora, antes de ser feita a chamada para CIRCLE 1 - CALL 232D.

Repare como o comando CIRCLE é lento.

Endereço	Código de máquina	Mnemônica	Comentários
----------	-------------------	-----------	-------------

END

:

:

IDÊNTICO AO DRAW

END + 15

END + 16	EF	RST 0028	Usa calculadora.
END + 17	34	DEFB 34	Dado na pilha.
END + 18	40B00064	DEFBs	= 100 decimal.
END + 22	31	DEFB 31	Duplica.
END + 23	34	DEFB 34	Dado na pilha.
END + 24	40B00030	DEFBs	= 48 decimal.
END + 28	38	DEFB 38	Final de cálculo.
END + 29	CD2D23	CALL CIRCLE 1	Traça círculo.
END + 32	D9		
END + 33	E1		
END + 34	D9		
END + 35	C9		

Esta rotina traça o círculo 100, 100, 48.

NOTA: Os nomes das rotinas aqui utilizadas não querem dizer absolutamente nada. Somente foram baseadas na ROM do Sindair 2X Spectrum, como referência.

Programas e rotinas

NOTAS INICIAIS

1. O byte mais significativo e o byte menos significativo.

Tomemos como exemplo o nosso endereço padrão, 65000.

Sabemos que

$$(253 \times 256) + 232 = 65000$$

Onde,

253 equivale ao byte mais significativo e
232 equivale ao byte menos significativo.

Esse processo é análogo ao sistema decimal, onde trabalhamos com, por exemplo, dezenas e unidades. Por exemplo:

$$27 = (2 \times 10) + 7$$

Onde,

2 é o valor mais significativo e
7 é o valor menos significativo.

Para fazer POKE de um valor x, em uma variável de sistema que consuma dois endereços, você deve se utilizar da fórmula:

```
POKE (variável sistema), x - 256 x INT (x/256)
POKE (variável sistema + 1), INT (x/256)
```

E, para saber o valor contido naquela variável, utilize:

```
PRINT PEEK (variável sistema) - 256 x PEEK (variável sistema + 1).
```

2. Você deve ter observado, no conjunto de instruções do microprocessador Z80, a diferença entre instruções que empregam parênteses em suas mnemônicas e instruções que não possuem parênteses.

Nas primeiras, ou seja, nas instruções que usam parênteses, o uso destes faz referência ao conteúdo daquele endereço apontado pelo registro ou par de registros envolvido na instrução.

Por exemplo:

```
LD A, B      - carrega acumulador com conteúdo do registro B.
LD A, (HL)   - carrega acumulador com conteúdo da locação
               apontada pelo par de registros HL.
```

Passemos então às rotinas e aos programas.

JOGO DE DAMAS

Este programa é uma combinação de Basic e linguagem de máquina, cujo objetivo é jogar gostosas partidas de dama com você.

O micro mostra o tabuleiro, com as suas peças (vermelhas) e as dele, pretas, em cima. As posições são identificadas por letras e números, respectivamente de A até H e de 1 a 8.

O computador checka a validade da sua jogada, e não permite saltos múltiplos.

O programa trabalha da seguinte maneira:

A parte em Basic serve para mostrar o tabuleiro, definir caracteres gráficos, aceitar ou não as jogadas do adversário e executá-las.

A parte em linguagem de máquina é para as jogadas do computador, a fim de que você não fique esperando muito tempo para ele jogar.

Os códigos decimais da linguagem de máquina estão nas declarações DATA, das linhas 9000 até 9450. Procure calmamente traduzir estes códigos para as respectivas mnemônicas, utilizando para tanto os Apêndices deste livro, juntamente com os Apêndices do manual do micro. É um grande exercício para quem está disposto a aprender realmente linguagem de máquina. Ou, então, use um programa Disassembler.

Como parte da linguagem de máquina, o computador tem uma representação do tabuleiro na memória, como sendo uma série de bytes, motivo dos diversos POKE e PEEK do programa, e, porque também são uma opção melhor que uma matriz, para efeito de jogadas.

Note na linha 3000, a chamada para a execução da rotina em linguagem de máquina.

A seguir, a listagem do programa.

```

10 REM JOGO DE DAMAS
20 REM HABILIDADE DO JOGADOR
30 REM OS JOGADORES SAO: 0=COMPUTADOR, 1=USUARIO
40 REM O JOGADOR 0 JOGA EM VERMELHO, O JOGADOR 1 EM PRETO
50 REM O JOGADOR 0 JOGA EM VERMELHO, O JOGADOR 1 EM PRETO
60 REM O JOGADOR 0 JOGA EM VERMELHO, O JOGADOR 1 EM PRETO
70 REM O JOGADOR 0 JOGA EM VERMELHO, O JOGADOR 1 EM PRETO
80 REM O JOGADOR 0 JOGA EM VERMELHO, O JOGADOR 1 EM PRETO
90 REM O JOGADOR 0 JOGA EM VERMELHO, O JOGADOR 1 EM PRETO
100 REM O JOGADOR 0 JOGA EM VERMELHO, O JOGADOR 1 EM PRETO
110 REM O JOGADOR 0 JOGA EM VERMELHO, O JOGADOR 1 EM PRETO
120 REM O JOGADOR 0 JOGA EM VERMELHO, O JOGADOR 1 EM PRETO
130 REM O JOGADOR 0 JOGA EM VERMELHO, O JOGADOR 1 EM PRETO
140 REM O JOGADOR 0 JOGA EM VERMELHO, O JOGADOR 1 EM PRETO
150 REM O JOGADOR 0 JOGA EM VERMELHO, O JOGADOR 1 EM PRETO
160 REM O JOGADOR 0 JOGA EM VERMELHO, O JOGADOR 1 EM PRETO
170 REM O JOGADOR 0 JOGA EM VERMELHO, O JOGADOR 1 EM PRETO
180 REM O JOGADOR 0 JOGA EM VERMELHO, O JOGADOR 1 EM PRETO
190 REM O JOGADOR 0 JOGA EM VERMELHO, O JOGADOR 1 EM PRETO
200 REM O JOGADOR 0 JOGA EM VERMELHO, O JOGADOR 1 EM PRETO
210 REM O JOGADOR 0 JOGA EM VERMELHO, O JOGADOR 1 EM PRETO
220 REM O JOGADOR 0 JOGA EM VERMELHO, O JOGADOR 1 EM PRETO
230 REM O JOGADOR 0 JOGA EM VERMELHO, O JOGADOR 1 EM PRETO
240 REM O JOGADOR 0 JOGA EM VERMELHO, O JOGADOR 1 EM PRETO
250 REM O JOGADOR 0 JOGA EM VERMELHO, O JOGADOR 1 EM PRETO
260 REM O JOGADOR 0 JOGA EM VERMELHO, O JOGADOR 1 EM PRETO
270 REM O JOGADOR 0 JOGA EM VERMELHO, O JOGADOR 1 EM PRETO
280 REM O JOGADOR 0 JOGA EM VERMELHO, O JOGADOR 1 EM PRETO
290 REM O JOGADOR 0 JOGA EM VERMELHO, O JOGADOR 1 EM PRETO
300 GOTO 3000
310 REM O JOGADOR 0 JOGA EM VERMELHO, O JOGADOR 1 EM PRETO
320 REM O JOGADOR 0 JOGA EM VERMELHO, O JOGADOR 1 EM PRETO
330 REM O JOGADOR 0 JOGA EM VERMELHO, O JOGADOR 1 EM PRETO
340 REM O JOGADOR 0 JOGA EM VERMELHO, O JOGADOR 1 EM PRETO
350 REM O JOGADOR 0 JOGA EM VERMELHO, O JOGADOR 1 EM PRETO
360 REM O JOGADOR 0 JOGA EM VERMELHO, O JOGADOR 1 EM PRETO
370 REM O JOGADOR 0 JOGA EM VERMELHO, O JOGADOR 1 EM PRETO
380 REM O JOGADOR 0 JOGA EM VERMELHO, O JOGADOR 1 EM PRETO
390 REM O JOGADOR 0 JOGA EM VERMELHO, O JOGADOR 1 EM PRETO
400 REM O JOGADOR 0 JOGA EM VERMELHO, O JOGADOR 1 EM PRETO
410 REM O JOGADOR 0 JOGA EM VERMELHO, O JOGADOR 1 EM PRETO
420 REM O JOGADOR 0 JOGA EM VERMELHO, O JOGADOR 1 EM PRETO
430 REM O JOGADOR 0 JOGA EM VERMELHO, O JOGADOR 1 EM PRETO
440 REM O JOGADOR 0 JOGA EM VERMELHO, O JOGADOR 1 EM PRETO
450 REM O JOGADOR 0 JOGA EM VERMELHO, O JOGADOR 1 EM PRETO
460 REM O JOGADOR 0 JOGA EM VERMELHO, O JOGADOR 1 EM PRETO
470 REM O JOGADOR 0 JOGA EM VERMELHO, O JOGADOR 1 EM PRETO
480 REM O JOGADOR 0 JOGA EM VERMELHO, O JOGADOR 1 EM PRETO
490 REM O JOGADOR 0 JOGA EM VERMELHO, O JOGADOR 1 EM PRETO
500 REM O JOGADOR 0 JOGA EM VERMELHO, O JOGADOR 1 EM PRETO
510 REM O JOGADOR 0 JOGA EM VERMELHO, O JOGADOR 1 EM PRETO
520 REM O JOGADOR 0 JOGA EM VERMELHO, O JOGADOR 1 EM PRETO
530 REM O JOGADOR 0 JOGA EM VERMELHO, O JOGADOR 1 EM PRETO
540 REM O JOGADOR 0 JOGA EM VERMELHO, O JOGADOR 1 EM PRETO
550 REM O JOGADOR 0 JOGA EM VERMELHO, O JOGADOR 1 EM PRETO
560 REM O JOGADOR 0 JOGA EM VERMELHO, O JOGADOR 1 EM PRETO
570 REM O JOGADOR 0 JOGA EM VERMELHO, O JOGADOR 1 EM PRETO
580 REM O JOGADOR 0 JOGA EM VERMELHO, O JOGADOR 1 EM PRETO
590 REM O JOGADOR 0 JOGA EM VERMELHO, O JOGADOR 1 EM PRETO
600 REM O JOGADOR 0 JOGA EM VERMELHO, O JOGADOR 1 EM PRETO
610 REM O JOGADOR 0 JOGA EM VERMELHO, O JOGADOR 1 EM PRETO
620 REM O JOGADOR 0 JOGA EM VERMELHO, O JOGADOR 1 EM PRETO
630 REM O JOGADOR 0 JOGA EM VERMELHO, O JOGADOR 1 EM PRETO
640 REM O JOGADOR 0 JOGA EM VERMELHO, O JOGADOR 1 EM PRETO
650 REM O JOGADOR 0 JOGA EM VERMELHO, O JOGADOR 1 EM PRETO
660 REM O JOGADOR 0 JOGA EM VERMELHO, O JOGADOR 1 EM PRETO
670 REM O JOGADOR 0 JOGA EM VERMELHO, O JOGADOR 1 EM PRETO
680 REM O JOGADOR 0 JOGA EM VERMELHO, O JOGADOR 1 EM PRETO
690 REM O JOGADOR 0 JOGA EM VERMELHO, O JOGADOR 1 EM PRETO
700 REM O JOGADOR 0 JOGA EM VERMELHO, O JOGADOR 1 EM PRETO
710 REM O JOGADOR 0 JOGA EM VERMELHO, O JOGADOR 1 EM PRETO
720 REM O JOGADOR 0 JOGA EM VERMELHO, O JOGADOR 1 EM PRETO
730 REM O JOGADOR 0 JOGA EM VERMELHO, O JOGADOR 1 EM PRETO
740 REM O JOGADOR 0 JOGA EM VERMELHO, O JOGADOR 1 EM PRETO
750 REM O JOGADOR 0 JOGA EM VERMELHO, O JOGADOR 1 EM PRETO
760 REM O JOGADOR 0 JOGA EM VERMELHO, O JOGADOR 1 EM PRETO
770 REM O JOGADOR 0 JOGA EM VERMELHO, O JOGADOR 1 EM PRETO
780 REM O JOGADOR 0 JOGA EM VERMELHO, O JOGADOR 1 EM PRETO
790 REM O JOGADOR 0 JOGA EM VERMELHO, O JOGADOR 1 EM PRETO
800 REM O JOGADOR 0 JOGA EM VERMELHO, O JOGADOR 1 EM PRETO
810 REM O JOGADOR 0 JOGA EM VERMELHO, O JOGADOR 1 EM PRETO
820 REM O JOGADOR 0 JOGA EM VERMELHO, O JOGADOR 1 EM PRETO
830 REM O JOGADOR 0 JOGA EM VERMELHO, O JOGADOR 1 EM PRETO
840 REM O JOGADOR 0 JOGA EM VERMELHO, O JOGADOR 1 EM PRETO
850 REM O JOGADOR 0 JOGA EM VERMELHO, O JOGADOR 1 EM PRETO
860 REM O JOGADOR 0 JOGA EM VERMELHO, O JOGADOR 1 EM PRETO
870 REM O JOGADOR 0 JOGA EM VERMELHO, O JOGADOR 1 EM PRETO
880 REM O JOGADOR 0 JOGA EM VERMELHO, O JOGADOR 1 EM PRETO
890 REM O JOGADOR 0 JOGA EM VERMELHO, O JOGADOR 1 EM PRETO
900 DATA 0,224,248,252,252,254,254,254
910 DATA 127,127,127,63,63,31,7
920 DATA 254,254,254,252,252,24
930 DATA 254,254,254,252,252,24

```



```

00000 DATA 0,7,01,60,66,116,66,67
00005 DATA 0,204,248,60,208,48,70,
00010 DATA 97,98,118,66,60,31,7,0
00015 DATA 134,70,48,66,60,248,20
00020 PRINT " A B C D E F G H":
00025 PRINT
00030 LET d=3: LET l=5
00035 FOR y=1 TO 8: PRINT AT y+y,
00040 "FOR z=1 TO 2: FOR x=1 TO 4:
00045 PRINT PAPER d: " "; PAPER l
00050 " "; NEXT x: PRINT : PRINT "
00055 " "; NEXT z
00060 NEXT y
00065 LET p=1
00070 FOR x=2 TO 8 STEP 2
00075 LET c=0: LET y=1: GO SUB 10
00080 LET y=3: GO SUB 1000
00085 LET c=2: LET y=7: GO SUB 10
00090 NEXT x
00095 FOR x=1 TO 7 STEP 2
00100 LET c=0: LET y=5: GO SUB 10
00105 LET c=2: LET y=6: GO SUB 10
00110 LET y=8: GO SUB 1000: NEXT x
00115 GO SUB 2000
00120 GO SUB 3000
00125 GO TO 300
00130 PAPER l: INK c: PRINT AT y+
00135 y,x+x:
00140 IF p=0 THEN PRINT " ";
00145 IF p=1 THEN PRINT "AB";
00150 IF p=2 THEN PRINT "EF";
00155 PRINT AT y+y+1,x+x:
00160 IF p=0 THEN PRINT " ";
00165 IF p=1 THEN PRINT "CD";
00170 IF p=2 THEN PRINT "GH";
00175 RETURN
00180 PAPER 7: INK 0: PRINT AT 6,
00185 "Sus vez";
00190 BEEP .5,10: PRINT AT 7,22;"
00195 De:
00200 PRINT AT 8,22;"Para: ";
00205 LET l=7: GO SUB 4000: LET f
00210 x=x: LET fy=y
00215 LET fp=32419+9*fy+fx: LET f
00220 =PEEK fp
00225 IF f<>2 AND f<>130 THEN GO
00230 TO 2010
00235 LET l=8: GO SUB 4000: LET t
00240 x=x: LET ty=y
00245 LET tp=32419+9*ty+tx: LET t
00250 =PEEK tp

```

```

20050 LET dx=tx-fx: LET dy=ty-fy
20055 IF (ABS dx<>1 AND ABS dx<>2
20060 ) OR ABS dx<>ABS dy THEN GO TO 2
20065 IF (dy=1 OR dy=2) AND f=2 T
20070 HEN GO TO 2010
20075 IF ABS dx=1 THEN GO TO 2500
20080 IF t<>0 THEN GO TO 2010
20085 LET jx=fx+dx/2: LET jy=fy+d
20090 c/d
20095 LET jp=32419+9*jy+jx: LET j
20100 =PEEK jp
20105 IF j<>1 AND j<>129 THEN GO
20110 TO 2010
20115 LET c=2: LET p=0: LET x=fx:
20120 LET y=fy: GO SUB 1000: POKE fp,
20125 0
20130 LET p=1: IF f=130 OR ty=1 T
20135 HEN LET p=2
20140 LET x=tx: LET y=ty: GO SUB
20145 1000: POKE tp,f: IF ty=1 THEN PO
20150 KE tp,130
20155 LET p=0: LET x=jx: LET y=jy
20160 GO SUB 1000: POKE jp,0
20165 RETURN
20170 IF t<>0 THEN GO TO 2010
20175 LET p=0: LET c=2: LET x=fx:
20180 LET y=fy: GO SUB 1000: POKE fp,
20185 0
20190 LET p=1: IF f=130 OR ty=1 T
20195 HEN LET p=2
20200 LET x=tx: LET y=ty: GO SUB
20205 1000: POKE tp,f: IF y=1 THEN POK
20210 E tp,130
20215 RETURN
20220 LET mp=USR 32020
20225 PAPER 7: INK 0: PRINT AT 10
20230 "Minha vez"
20235 PRINT AT 11,22;"De: ";
20240 PRINT AT 12,22;"Para: ";
20245 IF mp=0 THEN GO TO 3900
20250 LET fy=INT (mp/256): LET ty
20255 =mp-256*fy
20260 LET fx=INT (fy/16): LET fy=
20265 fy-16*fx
20270 LET tx=INT (ty/16): LET ty=
20275 ty-16*tx
20280 LET x=fx: LET y=fy: LET z=1
20285 GO SUB 3500: LET x=tx: LET y=
20290 ty: LET z=12: GO SUB 3500
20295 LET p=0: LET x=fx: LET y=fy
20300 GO SUB 1000
20305 LET tp=32419+9*ty+tx: LET t
20310 =PEEK tp
20315 LET p=1: IF t=129 THEN LET
20320 p=2
20325 LET c=0: LET x=tx: LET y=ty

```

```

: GO SUB 1000
3090 IF ABS (tx-fx)<>2 THEN RETU
RN
3100 LET x=fx+(tx-fx)/2: LET y=f
x+(ty-fy)/2
3110 LET p=0: GO SUB 1000: RETUR
N
3200 PRINT AT 2,28;CHR$ (x+64);C
HR$ (y+48): RETURN
3300 PRINT INK 0;AT 20,0;"Voce v
enceu - Parabens"
3400 PRINT AT 21,0;"Outra partid
a"
3500 IF INKEY$="" THEN GO TO 390
0
3600 IF INKEY$="s" THEN GO TO 10
00
3700 STOP
3800 PRINT AT 1,28; FLASH 1;" ";
CHR$ 8;
3900 LET ss=INKEY$
4000 IF ss="0" THEN GO TO 5000
4010 IF ss<"a" OR ss>"h" THEN GO
TO 4010
4030 PRINT ss: LET x=CODE ss-96
4040 PRINT AT 1,29; FLASH 1;" ";
CHR$ 8;
4050 LET ss=INKEY$
4060 IF ss<"1" OR ss>"8" THEN GO
TO 4050
4070 PRINT ss: LET y=CODE ss-48
4080 RETURN
4100 PRINT INK 0;AT 20,0;"Azar s
eu venci"
4200 GO TO 3901
4300 RESTORE 9100: FOR x=32020 T
O 32301
4400 READ a: POKE x,a: NEXT x
4500 RETURN
4600 DATA 175,33,177,125,6,10,11
4700 DATA 16,252,6,35,33,174,126
4800 DATA 254,1,40,16,254,129,32
4900 DATA 17,246,255,205,187,125
5000 DATA 255,205,187,125,17,8,0
5100 DATA 187,125,17,10,0,205,18
5200 DATA 35,35,16,219,58,182,12
5300 DATA 40,18,42,183,125,237,9
5400 DATA 125,229,126,54,0,25,54
5500 DATA 25,119,24,19,58,177,12

```

```

5600 DATA 40,29,42,178,125,237,9
5700 DATA 125,126,54,0,229,25,11
5800 DATA 225,205,135,125,121,23
5900 DATA 8,192,26,246,128,18,20
6000 DATA 0,0,201,235,205,146,12
6100 DATA 235,205,146,125,71,201
6200 DATA 197,17,164,126,175,237
6300 DATA 17,9,0,60,237,82,48,25
6400 DATA 71,25,125,60,7,7,7,7
6500 DATA 128,193,209,225,201,0,
6600 DATA 0,0,0,0,0,0,0,205
6700 DATA 19,126,254,255,200,230
6800 DATA 1,200,254,2,40,78,229,
6900 DATA 213,17,246,255,205,13,
7000 DATA 39,17,248,255,205,13,1
7100 DATA 31,17,8,0,205,5,125,40
7200 DATA 23,17,10,0,205,5,126,4
7300 DATA 15,209,225,34,178,125,
7400 DATA 180,125,62,1,50,177,12
7500 DATA 58,177,125,167,40,235,
7600 DATA 201,205,19,126,230,127
7700 DATA 201,205,19,126,254,130
7800 DATA 25,125,225,201,229,25,
7900 DATA 126,225,254,0,192,62,1
8000 DATA 182,125,34,183,125,237
8100 DATA 125,201

```

PROGRAMA REPRODUTOR DA FALA

Este programa, muito interessante por sinal, permite que o computador TK 90X armazene e reproduza posteriormente a sua fala, transformando-a numa sucessão de bits armazenados em sua memória, agrupados por bytes. Para reproduzir esses "bytes", ele envia os bits para a saída de som.

Para manter o som que ele recebeu o mais semelhante possível com a voz, muita memória é consumida, na razão de 1 a 2K de RAM por segundo de fala.

Digite o programa cuidadosamente, e grave-o.

A seguir, digite a rotina "ouvir", a partir de um endereço determinado, digamos, 65000. Grave-a também, logo a seguir, do programa Basic. Finalmente, digite a rotina "falar", num endereço posterior aos utilizados pela rotina "ouvir", para não haver sobreposição. Grave-a igualmente, imediatamente após a última.

Siga estas instruções para utilizar o programa:

1. Coloque uma fita cassete virgem no seu gravador, ou simplesmente acione-o em modo de gravação, anulando a trava antigração, para que você possa utilizar o seu microfone (supondo-se que ele tenha um μ em caso negativo, providencie a ligação de um qualquer no gravador);

2. Desconecte o cabo de microfone entre o gravador e o computador, e ligue o cabo "ear" para "ear" (fone de ouvido-monitor de som);

3. Carregue o programa no computador. Se você gravou-o sem a opção de "AUTO RUN", então dê RUN G0. Para gravar com essa opção, digite "SAVE falador" LINE 610;

4. Responda as perguntas do computador, relativas ao número de palavras e seu respectivo comprimento. A linha 150 permite que o gravador simplesmente armazene aquela palavra, não tendo nada a ver com a reprodução do som. Na linha 420, o tempo de atraso faz referência ao tom da sua voz. Valores baixos, tipo 10, são melhores para a posterior reprodução. Quando aparecer a mensagem da linha 450, e você pressionar ENTER, o computador estará pronto para receber a sua palavra, para armazenamento. Aca-bando o tempo, automaticamente ele passa para a sua reprodução. Pressione SPACE para interromper o processo;

5. Não se esqueça de aumentar o volume da televisão.

Estude as rotinas em linguagem de máquina, e veja como o computador utiliza as portas de entrada/saída para receber e enviar sinais.

A seguir as listagens do programa.

```

0>REM ARQUITRON INFORMATICA
LTD A
10 BORDER 0: PAPER 0: INK 6: B
RIGHT 1: CLS
15 GO TO 1000
30 CLEAR 32767
50 LET lis=65000: LET sp=65100
70 LET ze=SGN PI: LET on=SGN P
I: LET tw=on+on: LET th=INT PI:
LET po=255
80 LET st=64999: LET lim=32768
110 INPUT "Numero maximo de pal
avras ";m: IF m<on THEN GO TO 11
0
120 INPUT "Comprimento maximo d
e palavras ";n: IF n<ze THEN GO
TO 120
130 DIM a(m): DIM n$(m,n)
140 FOR c=on TO m
150 INPUT "Digite a palavra ";(
c): "a$
160 LET n$(c)=a$(TO (LEN a$ AN
D LEN a$<=n)+(n AND LEN a$>n))
170 GO SUB 370
180 IF a(c,th)=lim THEN PRINT "
Nao cabe na memoria ": GO TO 100
0
190 NEXT c
200 PRINT "Vocabulario completo
": GO TO 1000
220 CLS: PRINT "Numero da pala
vra ";TAB 15;"Atraso";TAB 21;"In
icio";TAB 28;"Fim"
230 FOR c=on TO m
240 PRINT c;TAB th;n$(c);TAB 18
;a(c,on);TAB 21;a(c,tw);TAB 27;a
(c,th)
250 GO SUB 520
260 NEXT c
270 GO TO 1000
290 INPUT "Qual das ";(m);" pal
avras voce quer ouvir ";c
300 IF c>m OR c<on THEN GO TO 2
90
310 GO SUB 520: GO TO 1000
330 INPUT "Qual das ";(m);" pal
avras voce quer mudar ";c
340 IF c>m OR c<on THEN GO TO 3
30
350 LET st=a(c,tw): LET lim=a(c
,th)
360 GO SUB 370: GO TO 1000
380 POKE lis+on,st-po*INT (st/p
o)
390 POKE lis+tw,INT (st/po)
400 POKE lis+4,lim-po*INT (lim/
po)

```



```

410 POKE lis+5,INT ((lim/po)
420 INPUT "Tempo de atraso (1-2
55) ";d
430 POKE lis+40,d
440 LET a(c,on)=d: LET a(c,tw)=
st
450 INPUT "Pressione ENTER para
falar e em seguida ouvir"; LIN
E a$
460 CLS: PRINT "Pressione BREA
K ou aguarde o final"
470 LET a(c,th)=(USR(lis)+on
480 IF a(c,th)>st THEN PRINT "N
ao ha som": LET c=c-on: RETURN
490 GO SUB 520
500 LET st=a(c,th)-on
510 RETURN
520 POKE sp+on,a(c,tw)-po*INT (
a(c,tw)/po)
540 POKE sp+tw,INT (a(c,tw)/po)
550 POKE sp+4,a(c,th)-po*INT (a
(c,th)/po)
560 POKE sp+5,INT (a(c,th)/po)
570 POKE sp+50,a(c,on)
580 LET a=USR sp
590 RETURN
600 CLEAR 64999: LOAD "ouvir"CO
DE: LOAD "falar"CODE: RUN
610 INPUT "Gravar da palavra ";
c1:" ate palavra ";c2
640 IF c1>m OR c2>m OR c1<ze OR
c2<ze OR c1>c2 THEN GO TO 630
650 LET stt=a(c2,th): LET le=a(
c1,tw)-stt+on
660 IF le<ze THEN PRINT "Compr
imento negativo": GO TO 630
670 CLS: PRINT "Escrever abai
xo ""Endereco inicial = ";stt
680 INPUT "Nome do arquivo ";a$
690 IF a$="" OR LEN a$>10 THEN
GO TO 680
700 SAVE a$CODE stt,le
710 GO TO 1000
1000 CLS: PRINT "*****
COMANDOS"
1010 PRINT " 1 - CONSTRUIR VOCAB
ULARIO"
1020 PRINT " 2 - VER VOCABULARIO
"
1030 PRINT " 3 - OUVIR ALGUMA PA
LAVRA "" 4 - MUDAR ALGUMA PALAV
RA"
1040 PRINT " 5 - GRAVAR PALAVRAS
"
1050 PAUSE 0: LET e$=INKEY$
1070 IF e$="1" THEN GO TO 30
1080 IF e$="2" THEN GO TO 210

```

```

1090 IF e$="3" THEN GO TO 280
1100 IF e$="4" THEN GO TO 320
1110 IF e$="5" THEN GO TO 620
1120 GO TO 1000

```

ROTINA OUVIR

```

255000
255001 21FF5A ld hl,5AFF
255002 110040 ld de,4000
255003 05 push hl
255004 A7 and a
255005 ED52 sbc hl,de
255006 23 inc hl
255007 4C ld c,h
255008 45 ld b,l
255009 31 pop hl
25500A 383A jr c,FE32
25500B 0C inc c
25500C F3 di
25500D 167F ld d,7F
25500E D86F exx
25500F D860 ld d,60
255010 D87F exx
255011 0B7F ld a,7F
255012 DBFE in a,(FE)
255013 1F rra
255014 382B jr nc,FE32
255015 CB6F bit 5,a
255016 20F5 jr nz,FE00
255017 D9 exx
255018 08A ld e,d
255019 DBFE in a,(FE)
255020 CB17 rl a
255021 CB17 rl a
255022 CB1B rr a
255023 3806 jr c,FE1D
255024 060C ld b,0C
255025 10FE djnz FE19
255026 10F0 jr FE0D
255027 7B ld a,e
255028 03 exx
255029 77 ld (hl),a
255030 2B dec hl
255031 7A ld a,d
255032 DBFE in a,(FE)
255033 1F rra
255034 3805 jr nc,FE2C
255035 10E2 djnz FE0B
255036 0D dec c
255037 20DF jr nz,FE0B
255038 23 inc hl
255039 7E ld a,(hl)
255040 0C inc a
255041 20FB jr z,FE2C
255042 2B dec hl
255043 41 ld b,h
255044 4D ld c,l

```



```
FE34 FB      ei
FE35 C9      ret
```

ROTINA FALAR

```
>Disassemble f24c
FE36 117F50A ld hl,5AFF
FE37 110040 ld de,4000
FE38 09 push hl
FE39 07 and a
FE3A 05 sbc hl,de
FE3B 03 inc hl
FE3C 04 ld c,h
FE3D 06 ld b,l
FE3E 01 pop hl
FE3F 00 inc c
FE40 4850 ld a,(5C48)
FE41 00 rrc a
FE42 00 rrc a
FE43 00 rrc a
FE44 07 and 07
FE45 00 exx
FE46 04 ld c,a
FE47 00 push hl
FE48 00 ld b,10
FE49 10 exx
FE4A 00 di
FE4B 07 ld a,(hl)
FE4C 00 rrc a
FE4D 00 rrc a
FE4E 00 rrc a
FE4F 00 rrc a
FE50 00 exx
FE51 00 ld e,a
FE52 00 ld d,08
FE53 00 ld a,e
FE54 04 and h
FE55 01 or c
FE56 03 out (FE),a
FE57 00 rrc e
FE58 00 dec d
FE59 00 jr nz,FE53
FE5A 00 ld b,00
FE5B 00 djnz,FE7F
FE5C 00 jr FE73
FE5D 00 exx
FE5E 00 dec hl
FE5F 00 ld a,00
FE60 00 djnz,FE5A
FE61 00 dec c
FE62 00 jr nz,FE5A
FE63 00 exx
FE64 00 pop hl
FE65 00 exx
FE66 00 ei
FE67 00 ret
```

ROTINAS DE SONS DIVERSOS

A única maneira de se extrair sons do seu micro, na linguagem Basic, é através do comando SOUND, o que, convenhamos, é um tanto limitado, ainda mais considerando-se a sua capacidade, que vai muito além desse SOUND.

A seguir são listadas três rotinas que produzem sons diversos e jamais conseguidos através da linguagem Basic.

Repare com muita atenção nas duas primeiras listagens, e note a pequena diferença entre elas. Enquanto a primeira utiliza a instrução INC E, a segunda usa a instrução DEC E, para justamente alterar a nota musical. Você também pode alterar os tons, através do comando POKE END + 7, para carregar o registro E com outros valores.

A terceira rotina produz outro tipo de som, que, dependendo de um determinado valor, pode parecer o ronco de um motor de um carro, o barulho de um avião, ou até um tiro de revólver.

Após digitar esta última, execute-a através de RAND USR END, para ver seu efeito.

Agora, digite o pequeno programa a seguir:

```
10 INPUT "Duração ";d
20 POKE END + 11, d - 256 x INT (d/256)
30 POKE END + 12, INT (d/256)
40 PRINT d
50 IF INKEY$ <> " " THEN RAND USR END
60 GOTO 50
```

1ª ROTINA

```
>Disassemble f24c
FDE8 3A4850 ld a,(5C48)
FDE9 0F rrc a
FDEA 0F rrc a
FDEB 0F rrc a
FDEC 01 ld e,01
FDED 03 di
FDEE 03 out (FE),a
FDEF 00 xor 10
FDF0 43 ld b,e
FDF1 10FE djnz FDF5
FDF2 10FE inc e
FDF3 00FE jr nz,FDF1
FDF4 00 ei
FDF5 00 ret
```

2ª ROTINA

```

>Dis a s s e m b l e f d e s
F0D00000 ld hl,4050
F0D00001 ld hl, (SC48)
F0D00002 rrc a
F0D00003 rrc a
F0D00004 rrc a
F0D00005 rrc a
F0D00006 ld hl,00
F0D00007 di
F0D00008 out (FE),a
F0D00009 xor 10
F0D0000A ld hl,00FE
F0D0000B djnz FDF6
F0D0000C djnz FDF6
F0D0000D djnz FDF6
F0D0000E djnz FDF6
F0D0000F ret

```

3ª ROTINA

```

>Dis a s s e m b l e f d e s
F0D00000 ld hl,4050
F0D00001 ld hl, (SC48)
F0D00002 rrc a
F0D00003 rrc a
F0D00004 rrc a
F0D00005 rrc a
F0D00006 ex af,af'
F0D00007 ld hl,00
F0D00008 ld hl,0050
F0D00009 ex af,af'
F0D0000A out (FE),a
F0D0000B xor 10
F0D0000C ex af,af'
F0D0000D ld hl,00
F0D0000E ld hl,00FE
F0D0000F end a
F0D00010 sbc hl,de
F0D00011 ld hl,de
F0D00012 add hl,de
F0D00013 add hl,de
F0D00014 ld hl,de
F0D00015 sub hl,de
F0D00016 jr c,FE0D
F0D00017 dec hl
F0D00018 ld hl,a
F0D00019 djnz FE0E
F0D0001A djnz FE0E
F0D0001B djnz FE0E
F0D0001C djnz FE0E
F0D0001D djnz FE0E
F0D0001E djnz FE0E
F0D0001F ret

```

ROTINA PARA ROLAR A TELA (SCROLL) LENTAMENTE

Esta rotina é mais conhecida como "scroll lento", e serve como apresentação de textos de muitos jogos ingleses. Você deve conhecê-la. Digite-a, preencha a sua tela com palavras, ou qualquer coisa, e dê, por exemplo, os comandos em modo direto:

FOR f=1 TO 176:RAND USR 65000:NEXT f

```

>Dis a s s e m b l e f d e s
F0D00000 ld hl,4000
F0D00001 push hl
F0D00002 ld de,5B00
F0D00003 ld bc,0020
F0D00004 ldir
F0D00005 pop hl
F0D00006 ld hl,00
F0D00007 push bc
F0D00008 push hl
F0D00009 ld hl,00
F0D0000A push bc
F0D0000B push hl
F0D0000C ld hl,00
F0D0000D push bc
F0D0000E push hl
F0D0000F pop de
F0D00010 inc hl
F0D00011 ld bc,0020
F0D00012 ldir
F0D00013 pop hl
F0D00014 inc hl
F0D00015 pop bc
F0D00016 djnz FDF1
F0D00017 push hl
F0D00018 ld de,0050
F0D00019 sbc hl,de
F0D0001A pop de
F0D0001B ld bc,0020
F0D0001C ldir
F0D0001D pop hl
F0D0001E ld de,0020
F0D0001F add hl,de
F0D00020 pop bc
F0D00021 djnz FDF6
F0D00022 ld de,0050
F0D00023 add hl,de
F0D00024 push hl
F0D00025 ld de,0020
F0D00026 add hl,de
F0D00027 pop de
F0D00028 ld bc,0020
F0D00029 ldir
F0D0002A pop hl

```

```

00000000 ld de, 0000
00000001 add hl, de
00000002 pop bc
00000003 djnz FDF7
00000004 ld de, 0020
00000005 sbc hl, de
00000006 ex de, hl
00000007 ld hl, 0000
00000008 ld b, 20
00000009 ld a, (hl)
0000000A nop
0000000B ld (de), a
0000000C inc de
0000000D inc hl
0000000E djnz FE44
0000000F ret

```

ROTINA PARA REPRODUÇÃO DE FITAS CASSETES

Esta rotina trabalha de um modo similar ao programa falador. Ela lê a porta de entrada EAR aproximadamente a cada 17 microssegundos (um microssegundo é um milionésimo de segundo), o que produz uma faixa de leitura de 57000 vezes por segundo (57 khz).

Para usar esta rotina, faça o seguinte:

1. desconecte o cabo MIC para MIC entre o gravador e o micro;
2. conecte o cabo EAR para EAR entre o gravador e o micro;
3. coloque uma fita com músicas gravadas e, com o volume no máximo, coloque o gravador em modo PLAY, após acionar esta rotina através de LET x=USR END. Para interromper a rotina, utilize a tecla BREAK.

```

>Disassemble fdeb
FDEB 01000000 di
FDEB 01000001 ld bc, 0000
FDEB 0BFE in a, (FE)
FDEB 0F rra
FDEB 0F rra
FDEB 03FE out (FE), a
FDEB 0F xor a
FDEB 03FE out (FE), a
FDEB 10FB djnz FDEC
FDEB 0D dec c
FDEB 0E0FA jr nz, FDEC
FDEB 0E7F ld a, 7F
FDEB 0BFE in a, (FE)
FDEB 1F rra
FDEB 0E0B jr c, FDEC
FDEB 0E01 ei
FDEB 0E02 ret

```

ROTINA PARA LER TECLADO (RECONHECER CÓDIGOS DE TECLAS PRESSIONADAS)

Infelizmente, no Manual do TK 90X, o capítulo sobre as instruções Basic, IN e OUT, saiu fraco. Mas, se você leu o meu livro de Basic, sabe que é possível ler o teclado com a instrução IN, cuja grande vantagem sobre INKEYS ou qualquer outro método em Basic é o de poder reconhecer diversas teclas ao mesmo tempo, fazendo com que a combinação de mais de uma tecla pressionada forneça uma outra ordem a ser executada.

A rotina abaixo faz exatamente isso, da seguinte maneira:

Tecla Q – movimento para cima.
Tecla A – movimento para baixo.
Tecla O – movimento para a esquerda.
Tecla P – movimento para a direita.
Teclas de CAPS SHIFT a SPACE – atirar.

Você pode introduzir esta rotina em seus jogos ou programas, fazendo com que novas direções sejam aceitas, como as quatro diagonais, através das combinações das teclas padrão da rotina.

```

>Disassemble fdeb
FDEB 01000000 ld bc, 0000
FDEB 3EDF ld a, DF
FDEB 0BFE in a, (FE)
FDEB 1F rra
FDEB 0303 jr c, FDF5
FDEB 0C inc c
FDEB 1805 jr FDF4
FDEB 1F rra
FDEB 0302 jr c, FDF4
FDEB 0B09 set 1, c
FDEB 3EFD ld a, FD
FDEB 0BFE in a, (FE)
FDEB 1F rra
FDEB 0304 jr c, FE05
FDEB 0E01 set 0, c
FDEB 1809 jr FE0E
FDEB 03FE ld a, FB
FDEB 0BFE in a, (FE)
FDEB 1F rra
FDEB 0302 jr c, FE0E
FDEB 0B09 set 0, c
FDEB 0E7F ld a, 7F
FDEB 0BFE in a, (FE)
FDEB 1F cpl
FDEB 0E01 and 1F
FDEB 0E02 set 4, c
FDEB 0E03 ret

```

ROTINA PARA ATRIBUTOS NA TELA

Como você sabe, pode escolher uma das oito cores do TK 90X tanto para INK quanto para PAPER. Você também pode especificar onde quer BRIGHT ou FLASH 1 ou 0. O único problema entretanto é que em Basic você não pode mudar qualquer desses atributos facilmente, em relação a qualquer caractere na tela, sem especificá-lo com os novos atributos.

A rotina a seguir permite que você faça essas alterações instantaneamente, operando num retângulo especificado na tela, sem alterar o que já está impresso ali.

A rotina opera num retângulo especificado da tela, usando as coordenadas do comando PRINT AT.

Consideremos, então, (X1, Y1) as coordenadas do canto superior esquerdo, e as coordenadas (X2, Y2) do canto inferior direito.

Estas coordenadas devem estar armazenadas nos endereços 23332, 23333, 23334 e 23335, através dos comandos:

```
POKE 23332, X1
POKE 23333, Y1
POKE 23334, X2
POKE 23335, Y2
```

Todas as cores do TK 90X derivam das três cores primárias: verde, vermelho e azul.

As operações possíveis com esta rotina são:

```
Pôr atributos.
Tirar atributos.
Mudar atributos.
Deixar atributos como estão.
```

Através das combinações das três cores primárias, surgem as outras cores do micro.

Para os atributos, devemos associar certos valores:

Tinta azul	1
Tinta vermelha	2
Tinta verde	4
Papel azul	8

Papel vermelho	16
Papel verde	32
BRIGHT	64
FLASH	128

Para as operações funcionarem com os nossos atributos, precisamos:

POKE 23328, soma dos valores dos atributos sem alterá-los.

POKE 23329, soma dos valores dos atributos a serem impressos.

POKE 23331, soma dos valores dos atributos a serem alterados.

Para executar a rotina, você deve dar o comando RAND USR END ou LET t = USR END, após, evidentemente ter dado os devidos POKE, para as suas modificações das cores.

```
Disassemble fdes
FE00 001205B ld hl, 5B20
FE01 001205B ld a, (hl)
FE02 001205B cpl
FE03 001205B inc hl
FE04 001205B and (hl)
FE05 001205B inc hl
FE06 001205B ld (hl), a
FE07 001205B inc hl
FE08 001205B inc hl
FE09 001205B ld b, (hl)
FE0A 001205B inc hl
FE0B 001205B inc hl
FE0C 001205B ld a, (hl)
FE0D 001205B cp 10
FE0E 001205B ret nc
FE0F 001205B sub b
FE10 001205B ret c
FE11 001205B inc a
FE12 001205B push af
FE13 001205B inc b
FE14 001205B dec b
FE15 001205B push hl
FE16 001205B ld hl, 5B00
FE17 001205B jr z, FE0C
FE18 001205B ld de, 0020
FE19 001205B add hl, de
FE1A 001205B djnz FE06
FE1B 001205B ex de, hl
FE1C 001205B pop hl
FE1D 001205B push de
FE1E 001205B inc hl
FE1F 001205B ld a, (hl)
FE20 001205B cp 20
FE21 001205B jr c, FE18
FE22 001205B pop bc
FE23 001205B pop bc
FE24 001205B ret
```



```

dec hl
dec hl
ld c, (hl)
pop hl
add hl, bc
sub c
jr c, FE18
inc a
ld b, a
ld a, b
sub b
ld a, a
push bc
push de
ld a, (0023)
ld d, a
ld e, (0020)
ld a, (hl)
xor d
and a
ld e, a
ld a, (0020)
add a, e
ld (hl), a
inc hl
djnz FE2E
pop de
add hl, de
pop bc
pop af
dec a
ret z
push af
jr FE28

```

ROTINA DE INVERSÃO DOS ATRIBUTOS

Esta rotina pega todas as "TINTAS" e "PAPÉIS" de cada caractere dentro de um retângulo especificado da tela, e os inverte.

Este retângulo é especificado exatamente da mesma maneira que a rotina anterior. Portanto, você deve, antes de mais nada, dar aqueles quatro POKEs para criar o retângulo. A seguir, para executar a rotina, digite LET t = USR END.

```

)Disassemble fda8
F0F080 ED4B045B ld bc, (0024)
F0F081 ED6B045B ld hl, (0025)
F0F082 7D ld a, l
F0F083 FE18 cp 18
F0F084 D0 ret nc
F0F085 91 sub c

```

```

ret c
ld e, a
ld a, h
cp 00h
ret nc
sub b
ret c
ld d, a
inc d
inc e
push de
ld hl, 0050
xor a
cp c
ld de, 0000
jr z, FE00h
add hl, de
dec c
jr nz, FE0A
cp b
inc hl
jr z, FE14
djnz FE11
pop bc
push de
push hl
push bc
ld a, 07
and (hl)
rrca
ld c, a
ld a, 08
and (hl)
add a, c
rrca
ld c, a
ld a, C0
and (hl)
add a, d
ld (hl), a
inc hl
djnz FE18
pop bc
pop hl
pop de
dec c
add hl, de
jr nz, FE15
ret

```

Aqui está uma versão desta rotina de inversão de atributos, mas, só para a tela inteira. Não é necessário dar-se nenhum POKE, e esta ocupa somente 29 bytes.

```

:Dis a s s e m b l e f d e s
FDE000 00000000 ld hl, 0000
FDE001 00000007 ld a, 07
FDE002 00000000 and (hl)
FDE003 00000000 rrca
FDE004 00000000 rrca
FDE005 00000007 ld d, a
FDE006 00000008 ld a, 08
FDE007 00000000 and (hl)
FDE008 00000004 add a, d
FDE009 00000000 rrca
FDE010 00000000 rrca
FDE011 00000007 ld d, a
FDE012 00000008 ld a, 08
FDE013 00000000 and (hl)
FDE014 00000004 add a, d
FDE015 00000000 ld (hl), a
FDE016 00000000 inc hl
FDE017 00000007 ld a, h
FDE018 00000000 cp 08
FDE019 00000000 jr c, FDEB
FDE020 00000000 ret

```

ROTINAS PARA "SCROLL" TEXTOS E GRÁFICOS

Aqui está um conjunto de rotinas para que você faça diversos SCROLL na tela. Através do estudo destas, você pode criar o movimento que quiser, bem como na área que quiser.

Da mesma forma que a rotina anterior, aqui, para cada movimento existem duas opções:

- a primeira é para rolar um determinado retângulo da tela; e
- a segunda, para rolar a tela inteira.

No caso de se utilizar a primeira opção, primeiramente você deve então criar o seu retângulo, através daqueles mesmos POKEs anteriores. Você tem três opções ao utilizar estas rotinas:

1. manter a linha ou coluna que não foi movimentada exatamente como estava antes;
2. rolar a linha ou coluna que seria deletada pelo movimento escolhido; e
3. preencher a linha ou coluna que não foi movimentada com um dos 256 padrões definidos por um POKE 23347, um número entre 0 e 255.

A rotina a seguir faz o movimento de SCROLL para a direita, em um retângulo especificado por você. Para fazê-la trabalhar em sentido contrário, você deve fazer estas três pequenas alterações:

- | | | |
|----|--------|------|
| a. | NOP | 00 |
| b. | INC HL | 23 |
| c. | LDIR | EDB0 |

Se você não quiser armazenar as duas rotinas separadamente, poderá converter de uma para outra, durante um programa Basic, ou através de comandos diretos:

```

LET T = END (endereço inicial)
ESQUERDA DIREITA
POKE T + 23, 0 POKE T + 23, 68
POKE T + 44, 35 POKE T + 44, 43
POKE T + 52, 176 POKE T + 52, 184

```

Pois então, não perca mais tempo e digite a rotina.

```

:Dis a s s e m b l e f d e s
FDE000 ED40000000 ld bc, (0004)
FDE001 00000000 ld hl, (0000)
FDE002 7D000000 ld a, l
FDE003 FE180000 cp 18
FDE004 D0000000 ret nc
FDE005 91000000 sub c
FDE006 D0000000 ret c
FDE007 57000000 ld d, a
FDE008 7C000000 ld a, h
FDE009 FE200000 cp 20
FDE010 D0000000 ret nc
FDE011 00000000 sub b
FDE012 D0000000 ret c
FDE013 57000000 ld e, a
FDE014 14000000 inc d
FDE015 D0000000 push de
FDE016 44000000 ld b, h
FDE017 79000000 ld a, c
FDE018 FE180000 and 18
FDE019 00000000 add a, 40
FDE020 57000000 ld h, a
FDE021 7C000000 ld a, c
FDE022 07000000 add a, a
FDE023 07000000 add a, a
FDE024 07000000 add a, a
FDE025 07000000 add a, a
FDE026 07000000 add a, a
FDE027 07000000 add a, b
FDE028 00000000

```

```

FDF0 00 ld l,a
FDF1 00 pop bc
FDF2 00 push bc
FDF3 00 push hl
FDF4 00 ld d,h
FDF5 00 ld e,l
FDF6 00 ld a,(hl)
FDF7 00 dec hl
FDF8 00 ex af,af'
FDF9 00 xor a,af'
FDD0 00 ld b,a
FDD1 00 cp c,FE1D
FDD2 00 jr nz,FE1D
FDD3 00 ld dr,hl,(5B32)
FDD4 00 ld a,l
FDD5 00 cp 01
FDD6 00 jr c,FE2A
FDD7 00 ld a,h
FDD8 00 jr nz,FE2B
FDD9 00 ex af,af'
FDE0 00 ld (de),a
FDE1 00 pop hl
FDE2 00 inc h
FDE3 00 ld a,h
FDE4 00 and 07
FDE5 00 jr nz,FE0E
FDE6 00 ld de,FE20
FDE7 00 add hl,de
FDE8 00 ld a,l
FDE9 00 cp 20
FDEA 00 jr nc,FE3D
FDEB 00 ld b,07
FDEC 00 add hl,bc
FDED 00 pop bc
FDEE 00 djnz FE0F
FDEF 00 ret

```

E aqui estão as versões simplificadas da rotina anterior, trabalhando apenas com a tela toda.

Elas requerem, para funcionar, apenas dois POKEs nos endereços 23346 e 23347.

Esquerda

```

>Disassemble fdeb
FDEB 210040 ld hl,4000
FDEB 011F00 ld bc,001F
FDEB 7E ld a,(hl)
FDEB 54 ld d,h
FDEB 5D ld e,l
FDF1 23 inc hl

```

```

FDF0 00 ld dr
FDF1 00 ex af,af'
FDF2 00 ld bc,(5B32)
FDF3 00 ld a,c
FDF4 00 cp 01
FDF5 00 jr c,FE03
FDF6 00 ld a,b
FDF7 00 jr nz,FE02
FDF8 00 ex af,af'
FDF9 00 ld (de),a
FDD0 00 ld a,h
FDD1 00 cp 58
FDD2 00 jr c,FDEB
FDD3 00 ret

```

Direita

```

FDF0 00 ld dr
FDF1 00 ex af,af'
FDF2 00 ld bc,(5B32)
FDF3 00 ld a,c
FDF4 00 cp 01
FDF5 00 jr c,FE03
FDF6 00 ld a,b
FDF7 00 jr nz,FE02
FDF8 00 ex af,af'
FDF9 00 ld (de),a
FDD0 00 ld a,h
FDD1 00 cp 58
FDD2 00 jr c,FDEB
FDD3 00 ret

```

A seguir, as versões, para a tela toda, de scroll para cima e para baixo.

São rotinas simples, que ocupam respectivamente 67 e 69 bytes.

As rotinas de scroll para cima e para baixo em áreas de retângulos definidos na tela são um pouco mais complicadas que as rotinas de movimento lateral, já que ocupam respectivamente 109 bytes cada uma. Só que infelizmente não as incluí no livro, para que fiquem como exercício.

A seguir, estão listadas as rotinas que movimentam a tela inteira, respectivamente, para cima e para baixo, com 67 e 69 bytes de comprimento cada. Compare esta rotina de scroll para cima com aquela anterior de scroll para cima lentamente, e veja a diferença entre as duas.

```

>Dis assemble f de8
FDF000 21100040 ld hl,4000
FDF001 1110005B ld de,5BFF
FDF002 0050 push de
FDF003 0050 push hl
FDF004 03E03 ld a,03
FDF005 012000 ld bc,0020
FDF006 0050 push bc
FDF007 0050 push hl
FDF008 00B0 ldir
FDF009 01 pop de
FDF00A 00E0 ld c,E0
FDF00B 00B0 ldir
FDF00C 00007 ld b,07
FDF00D 0000 add hl,bc
FDF00E 0001 dec a
FDF00F 0001 pop bc
FDF010 000F0 jr nz,FDF5
FDF011 000335B ld a,(5B32)
FDF012 0001 cp 01
FDF013 0011 jr c,FE1D
FDF014 0007 jr nz,FE15
FDF015 0005B ld hl,5BFF
FDF016 00B0 ldir
FDF017 0010 jr FE1D
FDF018 0041 ld b,c
FDF019 00335B ld a,(5B33)
FDF020 0010 ld (de),a
FDF021 0010 inc de
FDF022 00FC djnz FE19
FDF023 0001 pop hl
FDF024 0001 pop de
FDF025 0004 inc h
FDF026 007C ld a,h
FDF027 0040 cp 40
FDF028 0009 jr c,FDEE
FDF029 0000 ex de,hl
FDF030 0073 ld (hl),e
FDF031 0000 inc l
FDF032 00FC jr nz,FE26
FDF033 0000 ret

```

Rotina para scroll para cima na tela inteira.

E aqui, a rotina de scroll para a tela inteira para baixo.

```

>Dis assemble f de8
FDF000 211FF07 ld hl,57FF
FDF001 111FF5B ld de,5BFF
FDF002 0050 push de
FDF003 0050 push hl
FDF004 03E03 ld a,03
FDF005 012000 ld bc,0020
FDF006 0050 push bc
FDF007 0050 push hl

```

```

FDF07 EDB8 lddr
FDF09 01 pop de
FDF0A 00E0 ld c,E0
FDF0B EDB8 lddr
FDF0C 05F9 ld b,F9
FDF0D 009 add hl,bc
FDF0E 001 pop bc
FDF0F 000 dec a
FDF10 000F0 jr nz,FDF5
FDF11 000335B ld a,(5B32)
FDF12 0001 cp 01
FDF13 0011 jr c,FE1D
FDF14 0007 jr nz,FE15
FDF15 0005B ld hl,5BFF
FDF16 EDB8 lddr
FDF17 000B jr FE20
FDF18 0041 ld b,c
FDF19 00335B ld a,(5B33)
FDF20 0010 ld (de),a
FDF21 0010 dec de
FDF22 00FC djnz FE19
FDF23 0001 pop hl
FDF24 0001 pop de
FDF25 0004 dec h
FDF26 007C ld a,h
FDF27 0040 cp 50
FDF28 0009 jr nc,FDEE
FDF29 0000 xor a
FDF30 0020 ld b,20
FDF31 0010 ld (de),a
FDF32 0010 dec de
FDF33 00FC djnz FE28
FDF34 0000 ret

```

ROTINA DE LIMPAR A TELA (CLS) DE UMA MANEIRA DIFERENTE

Aqui estão duas rotinas para limpar a tela, só que de uma maneira diferente, e muito mais bonita, além de interessante.

Imagine que a sua tela, os desenhos, textos, caracteres e gráficos estejam sobre uma toalha que você enrola de um lado ou de outro, para limpar.

Pois é exatamente isso que estas rotinas fazem. O papel dessa toalha imaginária que limpa a tela.

A primeira rotina enrola a toalha para a direita, e a seguinte enrola em sentido inverso.

Você pode variar a velocidade da toalha, apenas com um simples POKE, que controla o comprimento de uma pausa após a limpeza de cada coluna da tela.

Para tanto, digite POKE END + 5, comprimento da pausa. Note que esse valor de comprimento da pausa, se for 0, corresponde a uma PAUSE 256. Para remover essa pausa, digite POKE END + 6, 0. Para voltar ao normal, dê POKE + 6, 118.

Lembre-se que, para utilizar estas rotinas, evidentemente você deve ter alguma coisa impressa na tela, para poder limpar.

Chame-as de maneira usual, através de RAND USR END ou LET B = USR END.

A rotina abaixo faz o movimento da esquerda para a direita, e a seguinte faz o movimento inverso, ou seja, da direita para a esquerda.

```
>Disassemble fdes
F0000 2000 ld l,00
F0001 2040 ld h,40
F0002 0603 ld b,03
F0003 76 halt
F0004 10FD djnz FDEE
F0005 4E ld c,(hl)
F0006 3600 ld (hl),00
F0007 0600 ld b,00
F0008 CB19 rrc
F0009 17 rla
F000A 10FB djnz FDF6
F000B 03 inc hl
F000C 03 or (hl)
F000D 77 ld (hl),a
F000E 111F00 ld de,001F
F000F 10 add hl,de
F0010 7C ld a,h
F0011 0300 cp 00
F0012 0000 jr nz,FDF1
F0013 13 inc de
F0014 0016 ld b,16
F0015 0000 ld a,(SC8D)
F0016 0000 ld (hl),a
F0017 10 add hl,de
F0018 10FC djnz FE0D
F0019 0A845C ld a,(SC84)
F001A 77 ld (hl),a
F001B 10 add hl,de
F001C 77 ld (hl),a
F001D 10 add hl,de
F001E 10 inc l
F001F 7D ld a,l
F0020 FE1F cp 1F
```

```
F0E10 20CC jr nz,FDEA
F0E11 2040 ld h,40
F0E12 06C0 ld b,C0
F0E13 76 ld (hl),d
F0E14 10 add hl,de
F0E15 10FC djnz FE22
F0E16 0A8D5C ld a,(SC8D)
F0E17 0616 ld b,16
F0E18 77 ld (hl),a
F0E19 10 add hl,de
F0E1A 10FC djnz FE2B
F0E1B 0A845C ld a,(SC84)
F0E1C 77 ld (hl),a
F0E1D 10 add hl,de
F0E1E 77 ld (hl),a
F0E1F C9 ret
```

```
>Disassemble fdes
FDE00 201F ld l,1F
FDE01 2040 ld h,40
FDE02 0603 ld b,03
FDE03 76 halt
FDE04 10FD djnz FDEE
FDE05 4E ld c,(hl)
FDE06 3600 ld (hl),00
FDE07 CB19 rrc
FDE08 17 rla
FDE09 10FB djnz FDF6
FDE0A 03 dec hl
FDE0B 03 or (hl)
FDE0C 77 ld (hl),a
FDE0D 112100 ld de,0021
FDE0E 10 add hl,de
FDE0F 7C ld a,h
FDE10 0300 cp 00
FDE11 0000 jr nz,FDF1
FDE12 13 inc de
FDE13 0016 ld b,16
FDE14 0000 ld a,(SC8D)
FDE15 0000 ld (hl),a
FDE16 10 add hl,de
FDE17 10FC djnz FE0D
FDE18 0A845C ld a,(SC84)
FDE19 77 ld (hl),a
FDE1A 10 add hl,de
FDE1B 77 ld (hl),a
FDE1C 10 add hl,de
FDE1D 10 inc l
FDE1E 7D ld a,l
FDE1F C9 ret
```



```

inc hl
ld e, (hl)
inc hl
ld d, (hl)
inc hl
ld a, (hl)
cp nz,a
jr nz,FE2D
ld c,00
ld b,(hl)
cp bz
jr nz,FE06
inc c
ld s,(5B38)
cp (hl)
jr nz,FE1B
ld a,(5B39)
ld e,(hl),s
ld a,(hl)
inc hl
dec de
bit 0,c
jr nz,FE02
cp 0B
jr nz,FE23
ld b,05
inc hl
dec de
ld nz,FE1F
cp 3A
jr z,FE2B
jr 0D
jr nz,FE02
jr FDEB
ld bc,(5B38)
ld a,c
cp (hl)
jr nz,FE35
ld (hl),b
add hl,de
jr FDEB

```

ROTINA PARA DELETAR LINHAS BASIC

Esta é outra rotina utilitária para programadores em linguagem Basic. Ela permite que você delete qualquer parte de um programa, desde a sua primeira linha, até o final, instantaneamente. Obviamente, para utilizá-la, você deve especificar dois valores: os números da primeira e da última linha a serem deletadas.

Vamos chamar a primeira linha a ser deletada de P, e a última de U. Os POKEs necessários são:

POKE 23357, P - 256 x INT (P/256)
POKE 23358, INT (P/256)
POKE 23359, U - 256 x INT (U/256)
POKE 23360, INT (U/256)

Ambos os valores são incluídos no bloco a ser deletado, e o segundo e/ou o quarto POKE pode ser omitido se o número da linha correspondente for menor que 256.

Chame a rotina através de RAND USR END ou LET E = USR END.

Se você quiser deletar a partir de uma determinada linha até o final do programa, então qualquer número de linha maior que o número da última linha de uma listagem (9999) serve como segundo valor. Por exemplo:

POKE 23360, 40

Porque $40 \times 256 = 10240$, seguramente maior que o maior número de linha admissível, que é 9999.

A seguir, a rotina:

```

00000000 Disassemble fde0
00000001 ld hl, (5053)
00000002 ld bc, (503D)
00000003 ex de, hl
00000004 ld hl, (504B)
00000005 and a
00000006 sbc hl, de
00000007 ret z
00000008 ex de, hl
00000009 ld d, (hl)
0000000A inc hl
0000000B ld e, (hl)
0000000C ex de, hl
0000000D sbc hl, bc
0000000E ex de, hl
0000000F jr nc, FE09
00000010 inc hl
00000011 ld e, (hl)
00000012 inc hl
00000013 ld d, (hl)
00000014 inc hl
00000015 add hl, de
00000016 jr FDEF
00000017 push hl
00000018 ld bc, (503F)
00000019 inc hl
0000001A ld e, (hl)
0000001B inc hl
0000001C ld d, (hl)

```



```

dec hl
ld e, (hl)
ld bc, (SB34)
push bc
ld hl, (SC53)
ld a, (hl)
inc a
jr z, FF0D
ld b, (hl)
inc hl
ld c, (hl)
inc hl
ex de, hl
push hl
scf
abc hl, bc
pop hl
jr c, FF0D
ex de, hl
ex (sp), hl
ld bc, (SB36)
add hl, bc
ex (sp), hl
ld c, (hl)
inc hl
ld b, (hl)
inc hl
add hl, bc
jr nc, FF0E
jr ff0E
pop de
pop hl
ld (hl), d
dec hl
ld (hl), e
ld a, 01
ld hl, FF7F
add hl, de
inc nc, FF00
inc a
ld hl, FF9C
add hl, de
inc nc, FF00
inc a
ld hl, FC18
add hl, de
inc nc, FF00
inc a
pop bc
pop hl
push af
push de
ld de, (SCB0)
push de
sub b
jr z, FF51
ld c, a

```

```

push de
jr c, FF42
ld b, 00
add hl, bc
ex (sp), hl
call 1655
jr FF50
cpl
inc a
ld e, a
ld d, 00
ld b, FF
add hl, bc
ex (sp), hl
ex de, hl
add hl, de
call 1655
pop hl
pop bc
pop de
pop af
push hl
ld hl, 0001
dec a
jr z, FF5B
push hl
ld l, 0A
dec a
jr z, FF5B
push hl
ld l, 04
dec a
jr z, FF5B
push hl
ld hl, 03E8
ex de, hl
ld a, 0F
and a
sbc hl, de
inc a
inc nc, FF6F
add hl, de
ld (bc), a
inc bc
dec e
jr nz, FF60
ld h, b
ld l, c
ld c, a
inc e
jr FF0B
pop de
jr FF6C

```


Programa "DESENHISTA"

Eis aqui um programa de desenho muito complexo, com facilidades de utilização muito avançadas, encontradas somente em programas comerciais importados.

Estas facilidades incluem, por exemplo, uma rotina de preenchimento de um polígono qualquer, com uma das 20 texturas pré-selecionadas existentes no programa, desenho de círculos cheios e vazios, linhas inteiras e pontilhadas, velocidade variável do cursor, ampliação do desenho em 4 ou 16 vezes, e armazenamento de duas telas.

Para utilizar o programa, primeiro digite o programa de controle da listagem 1, e grave-o com "auto" run a partir da linha 500. (SAVE "DESENHISTA" LINE 500.)

Não faça alterações nele, exceto no comando LOAD da linha 500, se você deseja armazenar tudo no Microdrive, porque a área de programas Basic está muito reduzida, devido ao endereço inicial da área de códigos de máquina.

A seguir digite o programa da LISTAGEM 2, que é um CARREGADOR DE CÓDIGOS HEXADECIMAIS, com um controle de soma de bytes por linha, facilitando a conferência da digitação, pois ele automaticamente acusa se houve erro em algum byte digitado.

Nesse caso, você deve digitar:

```
POKE 30456, 128
POKE 30457, 96
```

Para em seguida gravar os códigos com

```
SAVE "códigos" CODE 25000, 5512
```

Repare que o endereço inicial dos códigos encontra-se na locação 25000, e por isso a área Basic não pode ser alterada.

Assim que você terminar de digitar os códigos, se tiver digitado o comando CLEAR citado acima, pode digitar o comando NEW para limpar a área Basic, antes de gravar os códigos. Grave-os duas vezes seguidas e não se esqueça do comando VERIFY.

Para testar o programa, carregue-o com LOAD "", que carregará automaticamente os códigos. Antes disso, você deve resetar o sistema, ou des-

ligando o micro e tornando-o a ligá-lo, ou através de PRINT USR 0, que executa a sub-rotina do endereço 0 da ROM: limpa toda a memória.

Se você tiver paciência para digitar todos estes códigos, verá que, ao utilizar o programa, o trabalho realmente compensou, pois ele é incrível.

Se você possuir um programa Monitor Disassembler, estude as rotinas em linguagem de máquina, através das mnemônicas, para saber exatamente como se plota um ponto na tela, variando a sua velocidade, ou como se utilizam os atributos de uma posição. Para não se perder nestes valores malucos, preste atenção nas instruções em linguagem de máquina, do tipo "CP ...", ou "OR ...", ou similares que são as equivalentes ao comando "IF ... THEN ..." da Basic, que executam a função de aguardar a pressão sobre determinadas teclas.

Com o programa rodando, surge um menu de 6 opções, numeradas de 0 a 5. A opção 0 permite que você utilize o programa, no sentido de poder executar seus desenhos; a opção 5 faz um retorno da linguagem de máquina para a Basic, e as outras são opções de armazenamento ou carregamento de códigos, que representam desenhos.

Se você pressionar 0, a tela ficará toda vazia, com a exceção de uma pequena janela de informações no canto superior esquerdo. Assumindo que não existe nada na memória do micro, antes de você carregar o programa, os atributos da tela terão valor 0, e você não poderá utilizar o cursor. Pressione CAPS SHIFT e V e selecione a opção 3 para limpar a tela.

Como usar o programa

Para desenhar, obviamente a primeira coisa que você deve saber é movimentar o cursor. As teclas ao redor da tecla S escarregam-se disso, da seguinte maneira:

```

Q   W   E
  ↖   ↗
A ← S → D
  ↘   ↙
Z   X   C

```

Ou seja, pressionando a tecla A, você leva o cursor para a esquerda, na horizontal.

Estas teclas, quando pressionadas em conjunto com SYMBOL SHIFT, rodam a tela nas respectivas direções, e com CAPS SHIFT, movem a janela de ampliação da tela.

O cursor pode ser utilizado em quatro modos de desenho:

1. SET — desenha o ponto ou o que você quiser;
2. RES — equivale a INVERSE 1, ou seja, se tiver algo desenhado sob o cursor, este o apagará;
3. XOR — inverte pixels; equivale a OVER 1; e
4. SALT — permite que você movimente o cursor sem alterar o que está embaixo — você “salta” por cima.

Estes modos de desenho são obtidos respectivamente com as teclas P, O, I e U.

Para mudar as cores da tela, você deve mudar a partir do cursor, indicado por TELA, na janela de informação, para atributos do cursor — ATRI. A tecla L executa esta alteração. O cursor de atributos comporta-se exatamente como o cursor de pixels, com exceção do seu tamanho. Para retornar, pressione a tecla L novamente.

Em modo SET, você vai desenhar com valores para tinta, papel, brilho e intermitência (flash), indicados na janela de informação, na sua última linha. Estes valores podem ser alterados através das teclas numéricas de 0 a 7 que diretamente pressionadas, alteram a tinta; pressionadas com CAPS SHIFT, alteram a cor do papel; as teclas 8 e 9 sozinhas, ou com CAPS SHIFT, acionam ou desligam o brilho e a intermitência.

Note que quando utilizando o cursor de atributos, pressionando-se SYMBOL SHIFT juntamente com as teclas que movimentam o cursor, os atributos da tela irão rolar. (Scroll)

Finalmente, se você deseja aumentar ou diminuir a velocidade do cursor, pressione SYMBOL SHIFT com o valor da velocidade desejada, começando em 0 (sem repetição) e aumentando até chegar à tecla I.

Para as rotinas de desenho de segmento de retas e circunferências, você deve especificar dois pontos que, no caso do segmento, são seus extremos e, no caso da circunferência, especificam o comprimento do raio, a partir do centro.

Para fixar um desses pontos, pressione CAPS SHIFT junto com ENTER, que define a posição do cursor como último ponto desenhado. Em seguida, mova o cursor para o outro extremo da linha, ou o ponto da circunferência, e repita o procedimento.

Se você pressionar SYMBOL SHIFT e ENTER juntas vai ver o último ponto desenhado piscar, procedimento este muito útil quando se esquece qual foi a última posição plotada.

Para traçar uma linha, evidente que em modo SET, pressione CAPS SHIFT e a tecla J. Para traçar uma circunferência, pressione CAPS SHIFT e H. Se você pressionar SYMBOL SHIFT e H, o círculo será preenchido com a cor da tinta especificada. Se pressionar SYMBOL SHIFT e J, a linha especificada será desenhada em modo pontilhado. Para alterar os valores para este tipo de desenho, inicialmente fixados em 10101010, pressione CAPS SHIFT e K, e digite o novo valor, bit a bit.

Para facilitar a utilização dos atributos no seu desenho, já que é muito mais fácil fazer o desenho em preto e branco, para depois colori-lo, existem duas malhas que se sobrepõem ao desenho, para mostrar-lhe com exatidão o arquivo de atributos. Pressione G para uma malha em preto e branco, ou CAPS SHIFT e G para uma malha com as cores da tela.

Agora vamos ver a rotina de preenchimento de figuras. Mova o cursor (em modo SALT) para dentro da sua figura, e pressione CAPS SHIFT e F, para preenchê-la. Existem 20 texturas predefinidas no programa, que você pode utilizar através de números. Para as dez primeiras, você seleciona diretamente pressionando uma tecla entre 1 e 0; para as outras 10, você seleciona através de CAPS SHIFT e o número da textura desejada. Para utilizá-las no preenchimento de uma figura, você deve pressionar SYMBOL SHIFT junto com a tecla F, seguida do número da textura.

Mas você também pode definir o seu próprio padrão de desenho, que consiste de uma área de tamanho idêntico ao de um atributo definido de uma maneira muito similar a UDG. Para tanto, crie a sua textura, no tamanho apropriado, e com o cursor dentro dessa área, pressione CAPS SHIFT e P. Selecione, então, o padrão como na rotina de preenchimento de figuras, para em seguida pressionar SYMBOL SHIFT com o número da textura a ser substituído, a fim de que o procedimento tenha sucesso.

Similar a esse método de definição de texturas, existe um modo de definição de UDG — para selecioná-lo, pressione CAPS SHIFT e O. Responda à pergunta “UDG?”, pressionando a tecla correspondente ao caractere gráfico que você irá definir na faixa de A até U.

Existem duas áreas de armazenamento de desenhos na memória para que você possa utilizar uma como rascunho, e a outra para o desenho definitivo. Para “gravar” o desenho ou a tela na memória, pressione CAPS SHIFT com a tecla I. Para retornar este desenho, pressione CAPS SHIFT junto com U.

Para escrever na tela, existe o modo texto, obtido através da tecla T. Use as teclas numéricas do cursor, para movimentar o cursor de texto, mas note que a maneira de escrever está invertida: diretamente pressionadas, as teclas resultam em letras maiúsculas, e com CAPS SHIFT resultam em minúsculas. Para sair deste modo, pressione BREAK.

O modo gráfico para UDG, obtém-se pressionando CAPS SHIFT com 9.

A janela de informação

Os dois números que aparecem na sua linha superior, são as coordenadas X e Y do cursor. Estes valores são baseados nos valores normais do comando PLOT x, y da Basic, exceto que nas duas linhas inferiores (22 e 23), as abscissas Y passam a ser negativas.

A segunda linha da janela mostra o modo com o qual você está trabalhando — SET, RES, XOR, SALT ou TEXT e "SE", que significa modo "seguir cursor", em operação, e que será explicado adiante.

A terceira linha da janela mostra o modo do cursor — TELA ou ATRI para cursor normal, GRAF ou nada para cursor de texto, e a ampliação do desenho, cujo padrão é 1. Na última linha desta janela estão as informações sobre tinta, papel, brilho e intermitência — estes últimos, se acionados, aparecem como "B" e "F". Você pode também fazer esta janela sumir, pressionando CAPS SHIFT e N uma vez, e duas vezes para que ela retorne à tela. Normalmente não é preciso fazê-la desaparecer, já que se move automaticamente, procurando não interferir com o seu desenho.

Um dos procedimentos mais úteis do programa é a facilidade de ampliação do seu desenho, em 4 ou 16 vezes.

A área a ser ampliada é chamada janela de ampliação e pode ser obtida pressionando-se CAPS SHIFT e M uma vez, e outra vez para fazê-la sumir. Esta janela pode ser movimentada na tela, através de CAPS SHIFT e as teclas que movem o cursor. SYMBOL SHIFT e M selecionam o tamanho da ampliação, para finalmente através da tecla M sozinha, esta se efetivar.

Se você está desenhando em tamanho ampliado, automaticamente a janela de ampliação segue o cursor, em qualquer direção da tela. Este é o chamado "modo seguir cursor", citado anteriormente. Para desativá-lo, pressione SYMBOL SHIFT junto com a tecla S. Para centralizar sua janela, pressione simplesmente a tecla S, e para colocar o cursor no centro desta, pressione CAPS SHIFT e S.

Para limpar a tela, pressione CAPS SHIFT e V, fazendo surgir o menu de limpeza, e selecione o que limpar.

SYMBOL SHIFT e V selecionam o comando de inversão de atributos, por exemplo, para escrever frases em modo inverso.

Você pode também limpar partes da tela, invertendo-a, contornando-a com uma linha e preenchendo-a ainda em modo inverso, para em seguida retornar ao modo geral.

Para alterar a cor da borda do desenho, pressione B e o número equivalente à cor desejada. A última tecla funcional do programa é R, que traz o menu de armazenamento ou carregamento de desenhos.

RESUMO DAS TECLAS

TECLA	CAPS SHIFT	SYMBOL SHIFT	FUNÇÃO
0 a 5			Menu principal.
Q W E A D Z X C			Movem cursor.
Q a C		x	Rolam a tela nas diversas direções.
Q a C	x		Movem a janela de ampliação.
P			Modo SET.
O			Modo RESET.
I			Modo XOR.
U			Modo SALT.
L			Muda cursor TELA para ATRI e vice-versa.
0 a 7			Alteram cor da tinta.
0 a 7	x		Alteram cor do papel.
8 e 9			Acionam brilho e intermitência.
8 e 9	x		Desligam brilho e intermitência.
0 a 9		x	Alteram velocidade do cursor.
ENTER	x		Fixa extremos da linha e raio e circunferência.
ENTER		x	Mostra piscando o último ponto plotado.
J	x		Traça linha cheia.
J		x	Traça linha pontilhada.
H	x		Traça circunferência vazia.
H		x	Traça circunferência cheia.
K	x		Altera padrão do pontilhado.
G			Insere malha branca e preta.
G	x		Insere malha com as cores da tela.

TECLA	CAPS SHIFT	SYMBOL SHIFT	FUNÇÃO
F	x		Preenche um polígono.
F		x	Permite selecionar textura de preenchimento.
0 a 9			Primeiras dez texturas.
0 a 9	x		Últimas dez texturas.
P	x		Permite criar nova textura.
0 a 9			Substitui textura (entre dez primeiras).
0 a 9	x		Substitui textura (entre dez últimas).
0	x		Permite criar caractere UDG na letra pré-selecionada.
I	x		Grava desenho na memória.
U	x		Traz desenho da memória para a tela.
T			Aciona modo texto.
BREAK			Sai do modo texto.
5 a 8	x		Movem cursor de texto.
9	x		Aciona modo UDG.
N	x		Tira/põe janela de informação.
M	x		Tira/põe janela de ampliação.
M		x	Seleciona tamanho da ampliação.
M			Amplia desenho.
S			Centraliza janela de ampliação.
S		x	Desliga modo de seguir cursor.
S	x		Coloca cursor no centro da janela de ampliação.
V	x		Mostra menu de limpeza.
V		x	Mostra menu de inversão.
R			Mostra menu principal.
B			Altera cor da borda.

Paciência, que o programa é longo, mas compensa.

Muito cuidado, que basta um valor errado e o programa não funcionará: não confunda 8 com B, nem D com O.

De resto, divirta-se!

LISTAGEM 1 — PROGRAMA DE CONTROLE

```
10 LET a=USR VAL "25383"
20 LET o=INT (a/VAL "256"): LE
T b=a-VAL "256"*o
30 IF NOT o THEN CLS : PRINT "
Run para iniciar": STOP
40 INPUT n$: IF LEN n$>VAL "10
" OR NOT LEN n$ THEN GO TO VAL "
30"
50 RESTORE VAL "100"+b: READ c
,l: GO SUB VAL "199"+o
60 RUN
100 DATA USR "a",VAL "158"
101 DATA VAL "30285",VAL "150"
102 DATA VAL "57344",VAL "6912"
200 SAVE n$CODE c,l: VERIFY n$C
ODE: RETURN
201 LOAD n$CODE c,l: RETURN
500 CLEAR 24999: LOAD "c"CODE:
RANDOMIZE USR VAL "25380": RUN
```

LISTAGEM 2 — PROGRAMA CARREGADOR DOS CÓDIGOS HEXADECIMAIS

```
1 DEF FN b(ss)=CODE ss-48-7*(
ss>9)
2 DEF FN h(ss)=FN b(ss(1))+16
+FN b(ss(2))
3 DEF FN bs(a)=CHR$ (a+48+7*(
a>9))
4 DEF FN r#(a)=FN bs(INT (a/1
5))=FN bs(a-10+INT (a/16))
100 POKE 30300,8
1100 POKE 30300,8
1200 PRINT "00000000 TO 30448 STEP 8
"PRINT "00000000": LET c=0: LET
ss="00000000"
130 FOR e=0 TO 7: IF ss(1)=" "
THEN INPUT LINE ss: LET ss=ss+
"00000000"
140 LET h=FN h(ss): LET ss=ss(3
TO 8): POKE a+8+2: PRINT "":FN
h=FN h(ss): LET c=c+2: IF
70 NEXT e: INPUT "some:":k: IF
c<>k THEN PRINT "Erro": GO TO 2
0
150 PRINT: NEXT e: POKE 30455,
1500 POKE 30457,65
1600 SAVE "Desenho"CODE 25000,55
170
```

LISTAGEM DOS CÓDIGOS EM HEXADECIMAL

```
25000 20 20 40 4E 42 0D 4C 4B
25008 4A 48 50 4F 49 50MA 50 4A17
25016 39 38 37 36 31 32 33 34
25024 35 51 57 45 52 54 41 50MA 50 4A17
25032 44 48 47 01 5A 58 43 50MA 50 4A17
25040 FF 00 60 6E 62 6D 6C 60MA 50 4A17
25048 6A 68 70 6F 69 75 70 60MA 50 4A17
25056 09 08 07 06 00 02 00 00MA 50 4A17
25064 05 71 77 65 72 74 61 73MA 50 4A17
25072 64 65 67 00 7A 78 63 70MA 50 4A17
25080 20 00 2E 2C 2A 0D 3D 20MA 50 4A17
25088 2D 5E 22 3B 7F 5D 5B 50MA 50 4A17
25096 29 28 27 26 21 40 23 20MA 50 4A17
25104 25 20 20 20 3C 3E 7E 30MA 50 4A17
25112 5C 75 7D 00 3A 50 3F 50MA 50 4A17
25120 00 00 00 00 0E 05 90 00MA 50 4A17
25128 51 57 68 65 03 06 00 00MA 50 4A17
25136 00 00 00 00 5A 06 0B 00MA 50 4A17
25144 5E 66 06 66 00 00 7E 60MA 50 4A17
25152 7E 66 7E 66 7E 66 7E 60MA 50 4A17
25160 7E 66 7E 66 7E 66 7E 60MA 50 4A17
25168 7E 66 8D 66 62 66 66 60MA 50 4A17
25176 00 00 80 71 9C 66 63 60MA 50 4A17
25184 A1 65 00 00 8C 66 70 60MA 50 4A17
25192 A6 65 A6 65 50 66 66 60MA 50 4A17
25200 00 00 00 00 E2 66 6B 60MA 50 4A17
25208 51 67 66 65 03 66 66 60MA 50 4A17
```

```

25216 8C 73 E8 72 57 70 78 70
      SOMA = 1029
25224 F8 6F FE 6F 00 00 7A 7A
      SOMA = 1048
25232 7A 66 7A 66 7A 66 7A 66
      SOMA = 1006
25240 7A 66 7A 66 7A 66 7A 66
      SOMA = 1006
25248 7A 66 6C 64 61 64 66 66
      SOMA = 1016
25256 00 00 00 00 6B 64 39 6D
      SOMA = 1073
25264 70 64 FE 6E 00 67 00 00
      SOMA = 1071
25272 75 64 7A 64 7F 64 66 6D
      SOMA = 1077
25280 00 00 00 00 E6 66 66 66
      SOMA = 1027
25288 51 67 70 65 D3 66 00 00
      SOMA = 1010
25296 67 73 E4 72 BA 66 02 66
      SOMA = 1176
25304 BE 66 06 66 00 00 AB 66
      SOMA = 1006
25312 AB 66 AB 66 AB 66 AB 66
      SOMA = 1092
25320 AB 66 AB 66 AB 66 AB 66
      SOMA = 1092
25328 AB 66 9D 6D A2 60 A7 6D
      SOMA = 1086
25336 00 00 00 00 AC 60 DA 66
      SOMA = 1001
25344 B1 6D 15 6F 00 67 00 00
      SOMA = 1013
25352 B6 6D 6B 6D C0 6D AB 6A
      SOMA = 1159
25360 11 00 E0 ED 63 2E 77 0D
      SOMA = 1031
25368 0B 70 AF 02 3B 76 32 76
      SOMA = 1021
25376 76 3E 18 02 2D 77 09 09
      SOMA = 1056
25384 73 FE 76 31 00 E0 CD 0A
      SOMA = 1055
25392 63 ED 7B FE 76 C9 3A 77
      SOMA = 1337
25400 76 D3 FE CD 0D 68 CD 0E
      SOMA = 1159
25408 67 CD 3D 67 CD 26 64 CD
      SOMA = 1020
25416 04 74 3A 25 77 B5 08 00
      SOMA = 1004
25424 09 3A 2D 77 A7 CC 2E 07
      SOMA = 751
25432 18 E7 CD AB 67 00 21 BE
      SOMA = 1173
25440 63 CD 56 68 A7 28 CF 7E
      SOMA = 1162

```

```

25448 05 01 FF 00 C8 F5 CD 0D
      SOMA = 9924
25456 6B DD 21 94 63 CD 56 68
      SOMA = 1003
25464 01 4F C5 CD 0D 6B DD 21
      SOMA = 1048
25472 8C 63 CD 66 68 CD 62 68
      SOMA = 1137
25480 01 C9 4F C9 07 03 4E 41
      SOMA = 941
25488 4D 45 3F 27 11 07 63 41
      SOMA = 400
25496 56 45 2F 4C 4F 41 44 27
      SOMA = 920
25504 41 52 45 41 20 55 44 47
      SOMA = 937
25512 27 50 49 4E 54 41 52 50
      SOMA = 903
25520 50 41 44 52 41 4F 27 50
      SOMA = 902
25528 45 4C 41 20 20 27 18 09
      SOMA = 347
25536 4F 50 43 4F 45 63 20 27
      SOMA = 928
25544 45 44 49 54 41 52 20 54
      SOMA = 997
25552 45 4C 41 20 27 53 41 50
      SOMA = 915
25560 45 20 50 2F 20 45 49 54
      SOMA = 487
25568 41 27 4C 4F 41 44 20 20
      SOMA = 498
25576 44 41 20 20 46 49 54 41
      SOMA = 489
25584 27 53 41 55 45 20 50 50
      SOMA = 991
25592 20 4D 49 43 52 4F 44 50
      SOMA = 900
25600 49 56 45 27 4C 4F 41 44
      SOMA = 955
25608 20 20 44 45 20 20 4D 49
      SOMA = 415
25616 43 52 4F 44 52 49 56 45
      SOMA = 686
25624 27 53 41 49 20 44 4F 20
      SOMA = 471
25632 50 52 4F 47 2E 27 21 20
      SOMA = 482
25640 62 3A 28 77 A7 28 09 21
      SOMA = 984
25648 70 52 3D 28 03 21 C0 60
      SOMA = 937
25656 DD 21 20 77 0E 08 DD 7A
      SOMA = 724
25664 00 06 05 1F 36 0A 23 23
      SOMA = 173
25672 10 F9 DD 23 0D 20 EF C9
      SOMA = 1005

```

```

25680 5E 23 56 7A B3 C8 21 2E
      SOMA = 795
25688 67 E5 EB E9 11 FF FF 18
      SOMA = 1351
25696 21 11 00 FF 18 1C 11 01
      SOMA = 375
25704 FF 18 17 11 FF 00 18 12
      SOMA = 816
25712 11 01 00 18 0D 11 FF 01
      SOMA = 328
25720 18 08 11 00 01 18 03 11
      SOMA = 94
25728 01 01 21 94 6B E3 ED 48
      SOMA = 329
25736 2E 77 79 E6 1F 83 FE 00
      SOMA = 54
25744 D0 7B 81 4F 7A A7 28 11
      SOMA = 885
25752 FE 01 28 12 79 D6 20 47
      SOMA = 759
25760 30 07 78 D6 08 47 FE E0
      SOMA = 946
25768 D8 ED 43 2E 77 C9 08 79
      SOMA = 1015
25776 C6 20 4F 30 F4 78 C6 08
      SOMA = 927
25784 47 E6 18 FE 18 20 EA C9
      SOMA = 1070
25792 7B 07 07 07 AA E6 07 AA
      SOMA = 721
25800 07 07 07 AA E6 F8 AA C9
      SOMA = 1040
25808 57 07 07 AB E6 E0 AB 5F
      SOMA = 992
25816 7A 37 1F 37 1F 37 1F AA
      SOMA = 950
25824 E6 F8 AA E6 F8 67 C9 7C
      SOMA = 1538
25832 0F 0F 0F E6 03 C6 58 67
      SOMA = 667
25840 C9 22 2A 77 24 7C E6 07
      SOMA = 793
25848 C0 7D C6 20 6F 38 05 7C
      SOMA = 643
25856 D6 08 67 C9 7C 2F E6 18
      SOMA = 951
25864 C0 2A 2A 77 C9 22 2A 77
      SOMA = 791
25872 25 7C 2F E6 07 C0 7D 06
      SOMA = 976
25880 20 6F 38 E6 7C C6 08 67
      SOMA = 864
25888 C9 7D 07 07 07 AC E6 07
      SOMA = 756
25896 AC 07 07 07 AC E6 F8 AC
      SOMA = 1015
25904 57 C9 0C CB 08 D0 7D 3C
      SOMA = 904

```

```

25912 E6 1F 28 02 2C C9 0D C6
      SOMA = 764
25920 00 37 C9 0D C6 00 D0 7D
      SOMA = 808
25928 3D 2F E6 1F 28 02 2D C9
      SOMA = 657
25936 0C C6 08 37 C9 7D E6 E0
      SOMA = 1058
25944 6F 7D C6 20 6F D0 7C C6
      SOMA = 1107
25952 08 67 2F E6 18 C0 26 40
      SOMA = 706
25960 C9 2A F8 76 22 FA 76 C9
      SOMA = 1212
25968 CD F4 86 ED 4B FA 76 CD
      SOMA = 1436
25976 97 69 CD 93 69 CD 16 67
      SOMA = 1043
25984 20 F1 CD 0A 67 C3 0D 6B
      SOMA = 906
25992 11 00 00 18 26 11 FF FF
      SOMA = 806
26000 18 21 11 00 FF 18 1C 11
      SOMA = 998
26008 01 FF 18 17 11 FF 00 18
      SOMA = 999
26016 12 11 01 00 18 0D 11 FF
      SOMA = 345
26024 01 18 08 11 00 01 18 03
      SOMA = 76
26032 11 01 01 C1 ED 4B F8 76
      SOMA = 890
26040 3A 3B 76 E6 40 28 05 7B
      SOMA = 706
26048 E6 1F 07 07 07 6F 7A E6
      SOMA = 729
26056 1F 07 07 07 57 78 8D FE
      SOMA = 643
26064 C0 D0 47 7B 07 CB 1D 0F
      SOMA = 837
26072 81 C6 12 08 CB 1A 17 AA
      SOMA = 730
26080 E6 01 C0 08 4F 2A F8 76
      SOMA = 918
26088 ED 43 F8 76 4D 44 CD E6
      SOMA = 1250
26096 69 C3 00 69 79 07 07 07
      SOMA = 847
26104 A8 E6 C7 A8 07 07 6F 3A
      SOMA = 948
26112 0B 76 E6 40 20 16 78 A7
      SOMA = 812
26120 1F 37 1F A7 1F A8 E6 F8
      SOMA = 961
26128 A8 67 79 E6 07 3C ED 4B
      SOMA = 1001
26136 3B 76 47 C9 78 07 07 FE
      SOMA = 821

```



```

26144 58 E5 03 EE 58 67 ED 48
          SOMA = 1052
26152 3B 75 05 00 C9 3A 3B 75
          SOMA = 619
26160 F5 E5 BF 32 3B 75 CD 3E
          SOMA = 1160
26168 66 F1 32 3B 75 C9 79 07
          SOMA = 899
26176 07 07 A8 E6 C7 A8 07 07
          SOMA = 793
26184 6F 3A 3B 75 E6 40 20 1B
          SOMA = 699
26192 78 F5 07 0F 0F 0F A8 E6
          SOMA = 816
26200 F8 A8 67 79 E5 07 3C 47
          SOMA = 1008
26208 3E FE 0F 10 FD ED 4B 3B
          SOMA = 971
26216 75 47 C9 78 07 07 EE 78
          SOMA = 1010
26224 E5 03 EE F8 67 ED 4B 3B
          SOMA = 1193
26232 75 C9 0E 01 18 02 0E 00
          SOMA = 374
26240 C5 CD 7F 67 C1 47 FE 08
          SOMA = 1158
26248 38 0C D6 09 2F 0D 06 7F
          SOMA = 484
26256 28 0E C8 08 18 0A 08 78
          SOMA = 553
26264 0D 20 05 07 07 07 06 C7
          SOMA = 276
26272 4F 3A 3C 75 A9 A0 A9 02
          SOMA = 883
26280 3C 75 C9 CD 7F 67 6F 06
          SOMA = 963
26288 00 11 ED 75 19 7E 32 0D
          SOMA = 818
26296 77 C9 05 00 18 0A 06 01
          SOMA = 367
26304 18 05 05 02 18 02 05 03
          SOMA = 73
26312 3A 3B 75 A8 E6 FC A8 32
          SOMA = 1103
26320 3B 75 C9 CD A5 67 06 40
          SOMA = 922
26328 18 0E 05 80 18 0A 06 04
          SOMA = 216
26336 18 05 05 10 18 02 05 08
          SOMA = 92
26344 3A 3B 75 A8 32 3B 75 C3
          SOMA = 825
26352 94 6B 00 00 2A 3B 75 3A
          SOMA = 532
26360 F5 75 67 22 F2 66 F6 08
          SOMA = 1099
26368 32 F5 75 7D F5 80 32 3B
          SOMA = 1022

```

```

26376 75 C9 2A F2 65 7D 32 3B
          SOMA = 939
26384 75 7C 32 F5 75 C9 3E FE
          SOMA = 1173
26392 DB FE F5 E1 47 3E 7F DB
          SOMA = 1423
26400 FE F5 E2 A0 47 3E 61 DB
          SOMA = 1357
26408 FE F5 E0 A0 3C C9 CD 3A
          SOMA = 1408
26416 5B CD 04 74 CD 93 69 CD
          SOMA = 1094
26424 15 67 C8 18 F1 CD 3A 68
          SOMA = 950
26432 CD 04 74 CD 93 69 CD 15
          SOMA = 1009
26440 67 28 F2 CD F6 6B C3 CF
          SOMA = 1345
26448 6B DD 21 5A 67 CD B6 68
          SOMA = 1051
26456 CD 62 58 CD 74 67 FE 08
          SOMA = 1093
26464 30 F9 32 F7 75 D3 FE C3
          SOMA = 1372
26472 0D 6B 09 03 42 4F 52 44
          SOMA = 427
26480 41 20 3F 27 CD CF 6B CD
          SOMA = 923
26488 7F 67 FE FF 28 F6 C9 ED
          SOMA = 1453
26496 4B 23 77 78 F5 20 06 00
          SOMA = 633
26504 0F 04 30 FC 78 FE 06 D8
          SOMA = 915
26512 06 0B 79 F5 20 0F 05 30
          SOMA = 484
26520 FC 78 FE 06 30 03 3E FF
          SOMA = 1000
26528 C9 FE 0A D8 AF C9 3A F6
          SOMA = 1351
26536 75 0F D0 3F 17 32 F6 75
          SOMA = 841
26544 21 00 FB 11 00 F8 01 00
          SOMA = 550
26552 03 ED B0 C9 06 01 18 02
          SOMA = 650
26560 05 02 3A 3B 75 E6 40 C0
          SOMA = 729
26568 3A F5 75 21 0D 6B E5 E6
          SOMA = 1034
26576 01 20 D3 C5 21 00 F8 11
          SOMA = 739
26584 00 FB 01 00 03 ED B0 3A
          SOMA = 726
26592 F5 75 F5 01 32 F5 75 C1
          SOMA = 1216
26600 05 20 25 21 01 F8 11 02
          SOMA = 376

```



```

26608 F8 38 38 28 36 78 01 1E
      SOMA = 606
26615 00 ED B0 23 ED A0 2E 00
      SOMA = 891
26624 01 1F 00 ED B0 21 00 78
      SOMA = 726
26632 11 40 F8 01 C0 02 ED 80
      SOMA = 937
26640 09 21 00 F8 11 00 40 01
      SOMA = 884
26648 18 10 7E E6 BF 82 77 23
      SOMA = 1019
26656 7E E6 BF B3 77 23 10 78
      SOMA = 1138
26664 06 10 7A 53 5F 0D 20 8D
      SOMA = 801
26672 09 CD B6 68 DD 7E 00 D6
      SOMA = 1253
26680 04 47 48 E5 CD 62 68 E1
      SOMA = 1008
26688 CD 59 65 CD 59 65 E5 79
      SOMA = 1140
26696 90 C5 30 CD 72 68 2C CD
      SOMA = 1062
26704 62 68 E1 10 EE C8 CD 31
      SOMA = 1136
26712 68 C5 CD 74 67 C1 B9 D8
      SOMA = 1319
26720 18 F7 3E 47 08 DD 23 DD
      SOMA = 889
26728 7E 00 FE 27 C8 CD 72 68
      SOMA = 1042
26736 18 F3 C5 E5 6F 26 00 00
      SOMA = 803
26744 29 29 11 00 3C 19 EB E1
      SOMA = 944
26752 E5 CD 8F 68 25 CD E7 64
      SOMA = 1254
26760 08 77 08 E1 2C C1 C9 06
      SOMA = 804
26768 08 1A 77 24 13 10 FA C9
      SOMA = 875
26776 0A 08 4C 49 4D 50 41 27
      SOMA = 426
26784 4E 41 44 41 27 54 45 4C
      SOMA = 544
26792 41 20 20 27 41 54 52 49
      SOMA = 472
26800 27 54 55 44 4F 27 DD 45
      SOMA = 685
26808 00 DD 4E 01 04 04 21 00
      SOMA = 341
26816 40 3E 70 08 3E 23 CD 72
      SOMA = 662
26824 68 10 F9 CD 55 65 3E 23
      SOMA = 857
26832 CD 72 68 3E 47 08 DD 45
      SOMA = 855

```

```

26840 00 3E 20 CD 72 68 10 F9
      SOMA = 782
26848 3E 70 08 3E 23 CD 72 68
      SOMA = 702
26856 CD 55 65 0D 20 E0 DD 48
      SOMA = 951
26864 00 04 04 3E 23 CD 72 68
      SOMA = 828
26872 10 F9 21 42 40 DD 23 C9
      SOMA = 885
26880 3A 3B 75 E6 20 C4 97 69
      SOMA = 949
26888 3A 3B 75 E6 04 CA 08 69
      SOMA = 874
26896 CD 23 69 D8 CD 57 69 D8
      SOMA = 1174
26904 3A 3B 75 E6 08 CA 08 6A
      SOMA = 786
26912 C3 2D 6A 2A 2E 77 CD 21
      SOMA = 791
26920 65 7D 07 07 07 E6 F8 5F
      SOMA = 820
26928 79 93 38 50 07 4F 0F 7F
      SOMA = 820
26936 0F 0F E6 1F 2A 38 78 BD
      SOMA = 886
26944 30 42 78 92 38 3E BC 30
      SOMA = 734
26952 3B 07 47 3A 3B 75 E6 08
      SOMA = 610
26960 C8 C5 00 C8 01 A7 C9 3A
      SOMA = 1033
26968 F6 76 E6 08 C0 C5 2A 02
      SOMA = 1035
26976 74 CD 21 65 78 92 38 19
      SOMA = 802
26984 FE 20 30 15 7D E6 1F 07
      SOMA = 748
26992 07 07 5F 73 25 38 E0 7E
      SOMA = 909
27000 38 30 06 CD 99 75 C1 37
      SOMA = 833
27008 C9 C1 A7 C9 3A 3B 76 E6
      SOMA = 1227
27016 80 37 C0 CD 63 6D CD 16
      SOMA = 1015
27024 75 37 C9 ED 4B F8 78 ED
      SOMA = 1288
27032 5B 3B 76 3A F6 76 E6 04
      SOMA = 924
27040 C2 DF 70 78 EE 20 E6 FC
      SOMA = 1404
27048 F6 01 32 3B 76 D5 3E 3F
      SOMA = 812
27056 32 3C 76 C5 CD 08 69 C1
      SOMA = 936
27064 D1 3A 3B 76 AB E6 FC AB
      SOMA = 1268

```

```

27072 5F ED 53 3B 76 C9 CD 57
      SOMA = 1085
27080 69 D8 CD F4 65 79 E6 40
      SOMA = 1286
27088 20 22 3E FE 0F 10 FD 47
      SOMA = 737
27096 7E CB 41 20 01 A0 CB 49
      SOMA = 883
27104 20 02 A8 2F 77 C9 C5 CD
      SOMA = 971
27112 3E 66 CD EF 69 C1 C9 78
      SOMA = 1228
27120 E6 40 28 E4 3A 3C 76 47
      SOMA = 869
27128 7E CB 41 20 01 AF CB 49
      SOMA = 878
27136 20 01 A8 77 C9 CD F4 65
      SOMA = 1071
27144 79 E6 40 20 0F 3E 7E 0F
      SOMA = 666
27152 10 FD 47 CD D8 69 47 CD
      SOMA = 1142
27160 F1 64 70 C9 7D E6 DE 6F
      SOMA = 1342
27168 CD F4 69 23 77 11 20 00
      SOMA = 757
27176 19 77 2D 77 C9 CD F4 65
      SOMA = 1059
27184 79 E6 40 20 13 3E 1E 0F
      SOMA = 573
27192 10 FD 47 CD D8 69 06 03
      SOMA = 875
27200 4F CD F1 64 71 10 FA C9
      SOMA = 1205
27208 7D E6 9C 6F CD F4 69 01
      SOMA = 1177
27216 03 00 5D 54 13 3E 04 E6
      SOMA = 494
27224 ED B0 D1 21 20 00 19 E8
      SOMA = 947
27232 D5 01 04 00 3D 20 F1 D1
      SOMA = 761
27240 C9 DD 21 98 68 CD 56 68
      SOMA = 1106
27248 21 0D 6B E5 A7 C8 3D 28
      SOMA = 860
27256 06 3D 28 11 CD 8D 6A 21
      SOMA = 609
27264 00 E0 11 01 E0 01 FF 17
      SOMA = 745
27272 36 00 ED B0 C9 21 00 F8
      SOMA = 949
27280 11 01 F8 3A 3C 76 77 01
      SOMA = 622
27288 FF 02 ED B0 3A F8 76 E6
      SOMA = 1322
27296 FE 32 F6 76 C9 DD 21 EE
      SOMA = 1361

```

```

27304 6A CD 56 68 21 0D 6B E5
      SOMA = 883
27312 A7 C8 3D 28 06 3D 28 13
      SOMA = 894
27320 CD C8 6A 21 00 E0 01 00
      SOMA = 772
27328 18 7E 2F 77 23 08 78 B1
      SOMA = 669
27336 20 F7 C9 CD A6 67 21 00
      SOMA = 987
27344 F8 01 00 03 7E 5F 0F 07
      SOMA = 803
27352 0F AB E6 07 AB 67 78 07
      SOMA = 811
27360 07 07 AA E6 38 AA 77 23
      SOMA = 794
27368 0B 78 B1 20 E7 C9 0A 08
      SOMA = 790
27376 49 4E 56 45 52 64 27 4E
      SOMA = 689
27384 41 44 41 27 54 45 4C 41
      SOMA = 531
27392 20 20 27 41 54 52 49 27
      SOMA = 446
27400 54 55 44 4F 27 3A 3B 76
      SOMA = 690
27408 E6 DF 32 3B 76 47 E6 04
      SOMA = 985
27416 28 14 78 21 00 40 ED 58
      SOMA = 605
27424 2E 77 ED 4B 38 76 E6 08
      SOMA = 889
27432 CA C0 6C C3 36 6C 21 00
      SOMA = 892
27440 E0 11 00 40 01 00 1B ED
      SOMA = 570
27448 B0 C9 3A 3B 76 E5 14 FE
      SOMA = 1116
27456 10 C0 2A 2E 77 CD E7 64
      SOMA = 951
27464 E5 3A 37 76 77 3A 3B 76
      SOMA = 814
27472 E6 08 01 0F 00 28 03 01
      SOMA = 898
27480 07 00 5D 54 13 ED B0 E1
      SOMA = 841
27488 3A 3B 76 E6 08 01 10 06
      SOMA = 501
27496 28 03 01 08 05 C5 E5 11
      SOMA = 500
27504 20 00 EB 19 EB 06 00 ED
      SOMA = 770
27512 B0 E1 11 20 00 19 C1 10
      SOMA = 684
27520 EC 3A 37 76 4F C6 03 A9
      SOMA = 916
27528 E6 07 A9 D6 08 E6 3F 32
      SOMA = 971

```

```

27535 37 75 AF C9 3A 38 75 E5
                SOMA = 1014
27544 05 01 10 50 28 03 01 08
                SOMA = 173
27552 30 ED 43 38 75 ED 5B 00
                SOMA = 2000
27560 77 7B E5 1F 3D 81 FE 20
                SOMA = 9978
27565 38 07 7B E5 E0 C6 20 00
                SOMA = 1015
27575 5F CD C0 64 80 3D FE C0
                SOMA = 1227
27584 38 05 3E C0 90 CD D0 84
                SOMA = 973
27592 ED 53 2E 77 C3 0D 6B CD
                SOMA = 1005
27600 0F 6C CD F6 6B CD 1C 00
                SOMA = 1022
27608 08 CD F6 6B 3A 20 77 57
                SOMA = 1054
27615 E5 FD 32 20 77 AA 5F 3A
                SOMA = 1007
27624 27 77 57 E5 FE 32 27 77
                SOMA = 937
27632 AA B3 32 28 77 C9 CD 93
                SOMA = 1111
27640 69 3A 2D 77 A7 C8 6F 3A
                SOMA = 853
27648 F5 75 E5 04 28 03 2A 20
                SOMA = 727
27655 77 10 FE 2D 20 EB C9 20
                SOMA = 801
27664 20 77 11 21 77 01 07 00
                SOMA = 328
27672 38 1F ED B0 21 20 77 01
                SOMA = 883
27680 FE 7F 1E 00 ED 78 2F D6
                SOMA = 931
27688 77 B3 5F 23 C8 08 38 7A
                SOMA = 939
27696 7B 32 26 77 A7 C9 E5 D5
                SOMA = 1142
27704 C5 C5 D5 E5 41 C5 D5 1A
                SOMA = 1337
27712 4F 05 04 C8 11 9F 5F C8
                SOMA = 755
27720 11 9F AB E5 0F AB 77 23
                SOMA = 917
27728 10 F1 D1 13 C1 10 E5 E1
                SOMA = 1149
27735 05 03 C5 E5 5D 64 14 05
                SOMA = 838
27744 00 C5 21 C8 21 ED B0 E1
                SOMA = 1110
27752 24 C1 10 EE CD F1 64 E8
                SOMA = 1254
27760 E1 CD F1 64 EB C1 10 C1
                SOMA = 1408

```

```

27768 C1 E1 CD E7 64 C5 A0 57
                SOMA = 1399
27775 5D E1 CD E7 64 C5 38 CB
                SOMA = 1315
27784 38 CB 38 CB 21 CB 21 C5
                SOMA = 984
27792 D5 E5 41 1A 77 23 77 23
                SOMA = 841
27800 77 23 77 23 13 10 F4 05
                SOMA = 593
27808 03 E1 C5 11 20 00 EB 19
                SOMA = 734
27815 E5 E5 05 00 ED B0 CB 38
                SOMA = 1142
27824 E1 C1 10 EE 11 20 00 19
                SOMA = 745
27832 EB C1 09 EB C1 10 D0 C9
                SOMA = 1290
27840 E5 D5 C5 C5 E5 D5 E5 41
                SOMA = 1572
27848 D5 C5 1A 4F 05 04 AF CB
                SOMA = 903
27855 01 17 17 10 FA 5F 0F B3
                SOMA = 602
27864 77 23 05 04 AF CB 01 17
                SOMA = 866
27872 17 10 FA 5F 0F B3 77 23
                SOMA = 732
27880 C1 D1 1C 10 DB E1 5D 54
                SOMA = 1067
27888 14 C5 21 ED B0 E1 CD F1
                SOMA = 1340
27896 64 EB E1 24 CD F1 64 C1
                SOMA = 1335
27904 10 C1 C1 E1 CD E7 64 C5
                SOMA = 1361
27912 A0 57 5D E1 CD E7 64 78
                SOMA = 1221
27920 0F 0F 0F E5 1F 47 C5 E5
                SOMA = 803
27928 D5 41 C5 01 20 00 1A 77
                SOMA = 653
27936 09 77 2C 77 ED 42 77 2C
                SOMA = 757
27944 1C C1 10 EE E1 0E 20 09
                SOMA = 755
27952 EB E1 0E 40 09 C1 10 DE
                SOMA = 978
27960 C9 2A 2E 77 CD 21 65 7D
                SOMA = 872
27968 07 07 07 E5 F8 5F 3A 38
                SOMA = 711
27975 75 21 20 18 E5 05 20 01
                SOMA = 478
27984 29 19 ED 4B F8 75 22 F8
                SOMA = 1025
27992 75 3A 38 75 E5 20 C5 C3
                SOMA = 1010

```



```

28000 97 69 09 ED 4B F8 76 3A SOMA = 1193
28008 3B 76 E6 08 28 14 79 08 SOMA = 810
28016 1F 30 01 AF E5 F8 4F 78 SOMA = 932
28024 06 17 30 01 AF E5 F8 47 SOMA = 1010
28032 18 12 79 D6 3F 30 01 2A FF SOMA = 684
28040 E6 F8 4F 78 D6 2F 30 01 SOMA = 987
28048 AF E6 F8 47 CD 2D 66 28 22 SOMA = 1110
28056 2E 77 03 94 6B 11 FF 77 SOMA = 1142
28064 18 21 11 00 FF 18 10 2A 22 SOMA = 908
28072 01 FF 18 17 11 FF 00 00 00 SOMA = 909
28080 12 11 01 00 18 0D 11 77 77 SOMA = 945
28088 01 18 06 11 00 01 18 03 03 SOMA = 78
28096 11 01 01 21 0D 8B E3 05 05 SOMA = 512
28104 3A 3B 76 E6 40 28 72 2A 22 SOMA = 905
28112 A7 28 3B F2 F3 8D 21 09 09 SOMA = 903
28120 F8 11 00 77 01 20 00 00 00 SOMA = 904
28128 B0 11 00 F8 01 E0 02 00 00 SOMA = 905
28136 B0 21 00 77 01 20 00 00 00 SOMA = 908
28144 B0 18 1B 21 FF FA 11 1A 1A SOMA = 613
28152 77 01 20 00 ED B8 11 77 77 SOMA = 845
28160 FA 01 E0 02 ED B8 21 1F 1F SOMA = 952
28168 77 01 20 00 ED B8 D1 77 77 SOMA = 905
28176 A7 08 F2 2B 6E 21 01 77 77 SOMA = 1044
28184 11 00 F8 06 18 C5 1A 01 01 SOMA = 519
28192 1F 00 ED B0 12 23 13 01 01 SOMA = 709
28200 10 F3 09 21 FE FA 11 77 77 SOMA = 1259
28208 FA 06 18 C5 1A 01 1F 01 01 SOMA = 905
28216 ED B8 12 1B 2B C1 10 77 77 SOMA = 901
28224 09 7A A7 2B 53 F2 71 6E 6E SOMA = 1078

```

```

28232 21 00 E0 11 00 77 01 20 SOMA = 426
28240 00 ED B0 21 00 E1 11 00 SOMA = 888
28248 E0 E5 01 20 00 ED B0 D1 SOMA = 1108
28256 6B 62 CD F1 64 20 F2 21 SOMA = 1058
28264 00 77 01 20 00 ED B0 18 SOMA = 889
28272 27 21 FF F7 11 1F 77 01 SOMA = 742
28280 20 00 ED B8 11 FF F7 21 SOMA = 1005
28288 FF F6 E5 01 20 00 ED B8 SOMA = 1184
28296 E1 5D 54 CD 0D 65 20 77 22 SOMA = 905
28304 21 1F 77 01 20 00 ED B8 SOMA = 607
28312 D1 7B A7 C8 F2 B9 6E 21 SOMA = 1269
28320 1F E0 06 C0 C5 E5 06 00 07 SOMA = 917
28328 CB 16 2B 10 FB E1 CB 1A 1A SOMA = 993
28336 CB 06 CD F1 64 C1 10 00 00 SOMA = 1200
28344 09 21 00 E0 06 C0 C5 E5 06 SOMA = 1082
28352 06 20 CB 1E 23 10 FB E1 10 SOMA = 798
28360 CB 16 CB 0E CD F1 64 C1 10 SOMA = 1181
28368 10 E0 C9 78 B6 77 7C 77 7C SOMA = 1244
28376 E0 67 78 A3 B6 77 7C 77 7C SOMA = 1265
28384 5F 67 C9 78 A6 C9 3A 77 77 SOMA = 1199
28392 BF FE FF C0 79 0F 0F 0F 0F SOMA = 1058
28400 AC E6 E0 AC 06 7D D9 8F 8F SOMA = 1259
28408 08 67 E3 E5 D9 C9 21 00 00 SOMA = 1018
28416 3C 18 18 11 03 4E 56 4D 4D SOMA = 368
28424 45 52 4F 20 20 50 41 44 44 SOMA = 507
28432 52 41 4F 3F 27 CD 32 70 70 SOMA = 895
28440 DA 0D 6B 22 FC 76 D9 E5 E5 SOMA = 1188
28448 CD F4 66 21 FF FF 22 FE FE SOMA = 1382
28456 BF E5 AF 32 3B 76 CD 08 08 SOMA = 1038

```



```

28464 70 ED 4B F8 76 CD F4 85 SOMA = 1340
28472 48 0D 3E 01 0F 10 FD 47 SOMA = 11190
28480 7C E6 07 ED 5B FC 76 83 SOMA = 11190
28488 5F 7A CE 00 57 1A 5F 85 SOMA = 11190
28496 00 CD 43 65 38 08 CD 83 SOMA = 11190
28504 6E 28 F6 CD 32 55 CD D3 SOMA = 11168
28512 6E CD F1 64 28 10 CD E3 SOMA = 11144
28520 6E 28 04 CB 8A 18 07 CB SOMA = 7229
28528 4A CC E6 6E CB CA 2A 2A SOMA = 11107
28536 77 CD 0D 65 28 10 CD E3 SOMA = 9928
28544 6E 28 04 CB 82 18 07 CB SOMA = 7221
28552 42 CC E6 6E CB C2 2A 0A SOMA = 1091
28560 77 CD 32 65 38 05 CD E3 SOMA = 9928
28568 6E 28 03 E1 7C E6 E0 07 SOMA = 11155
28576 07 07 4F 3C 47 7C E6 17 SOMA = 6099
28584 F8 40 57 FE 58 38 8B CD SOMA = 11155
28592 0A 67 E1 D9 C3 0D 6B 13 SOMA = 9928
28600 06 53 41 4C 56 41 2F 0A SOMA = 1110
28608 45 43 55 50 45 52 41 0A SOMA = 6096
28616 54 45 4D 50 4F 52 41 0A SOMA = 6018
28624 49 41 4D 45 4E 64 4B 07 SOMA = 6094
28632 50 52 4F 56 49 53 4F 0A SOMA = 6044
28640 49 41 4D 45 4E 64 4B 07 SOMA = 6094
28648 DD 21 B7 6F CD 66 68 0A SOMA = 9999
28656 00 0F 0F 0F C6 80 67 0A SOMA = 6075
28664 CD E6 6F EB 18 10 CD E6 SOMA = 12860
28672 6F 3A F6 76 E6 FE 32 7A SOMA = 1313
28680 76 18 1C 11 00 80 21 00 SOMA = 346
28688 E0 01 00 18 ED 80 3A 76 SOMA = 996

```

```

28696 76 E6 01 01 00 03 28 0D SOMA = 406
28704 26 F8 18 09 21 00 40 11 SOMA = 436
28712 00 E0 01 00 1B ED 80 C3 SOMA = 860
28720 0D 66 DD 21 03 6F CD 86 SOMA = 875
28728 68 CD 62 68 CD 74 67 87 SOMA = 1070
28736 67 87 6F 26 00 3A 28 77 SOMA = 636
28744 FE 02 3F D8 A7 11 4D 76 SOMA = 914
28752 28 03 11 9D 76 19 C9 CD SOMA = 756
28760 32 70 DA 0D 6B E5 ED 4B SOMA = 1041
28768 F8 76 CD 2D 66 7C E6 F8 SOMA = 1320
28776 67 D1 06 08 7E 12 13 24 SOMA = 525
28784 10 FA C3 0D 6B DD 21 A0 SOMA = 995
28792 70 CD B6 68 CD 62 68 CD SOMA = 1215
28800 2E 67 CD 3D 67 CD A7 76 SOMA = 1002
28808 E6 DF D6 41 38 F4 FE 15 SOMA = 1307
28816 D2 0D 6B 26 00 87 87 87 SOMA = 773
28824 6F ED 5B 78 5C 19 18 8D SOMA = 892
28832 06 03 55 44 47 3F 27 11 SOMA = 352
28840 A6 61 3A 28 77 A7 28 0A SOMA = 698
28848 11 D0 61 3D 28 03 11 78 SOMA = 691
28856 61 21 20 77 06 05 0E 08 SOMA = 314
28864 CB 1E 38 0B 13 10 F9 03 SOMA = 619
28872 0D 06 05 20 F3 AF C9 1A SOMA = 701
28880 C9 3A F6 76 E6 FB 32 F6 SOMA = 1400
28888 76 3A 3B 76 E6 20 C8 3A SOMA = 873
28896 3B 76 EE 20 32 3B 76 E6 SOMA = 904
28904 20 DD 21 00 A0 28 0F 3A SOMA = 559
28912 F6 76 E6 02 DD 21 3D 76 SOMA = 1029
28920 28 04 DD 21 45 76 3A 3B SOMA = 602

```

```

28928 76 ED 4B F8 76 E6 04 20
      SOMA = 1062
28936 09 CD F4 65 DD E5 D1 03
      SOMA = 1413
28944 6F 66 CD 23 69 D8 28 03
      SOMA = 882
28952 CD F4 65 06 08 C5 DD 6E
      SOMA = 1076
28960 00 06 04 C5 C8 03 9F 4F
      SOMA = 651
28968 C8 03 9F A9 E6 0F A9 06
      SOMA = 954
28976 04 77 24 10 FC 7C D6 04
      SOMA = 769
28984 67 C1 2C 10 E6 C1 DD 23
      SOMA = 1035
28992 7D D6 04 6F 7C C6 04 67
      SOMA = 883
29000 CD F6 64 10 D0 C9 CD F4
      SOMA = 1425
29008 65 06 08 C5 DD 4E 00 DD
      SOMA = 632
29016 23 06 04 AF C8 11 17 17
      SOMA = 486
29024 10 FA 47 0F B0 77 06 04
      SOMA = 657
29032 AF C8 11 17 17 10 FA 47
      SOMA = 778
29040 0F B0 4E 2C 77 24 77 2D
      SOMA = 632
29048 71 CD F1 64 C1 18 D4 C9
      SOMA = 1281
29056 3A 3B 76 07 07 D8 07 DC
      SOMA = 692
29064 93 69 3A F6 76 F6 04 32
      SOMA = 974
29072 F6 76 ED 4B F8 76 76 E6
      SOMA = 1392
29080 F8 47 79 E6 F8 4F ED 43
      SOMA = 1301
29088 F8 76 CD 20 72 CD 2E 67
      SOMA = 1071
29096 3E 10 32 2C 77 CD CF 68
      SOMA = 810
29104 CD A7 70 FE FF CA D1 70
      SOMA = 1515
29112 A7 28 ED CD D1 71 3A 2C
      SOMA = 1073
29120 77 D6 18 38 08 FE 20 30
      SOMA = 755
29128 E1 3E 20 18 DD 3E 50 18
      SOMA = 730
29136 D9 F5 CD D9 70 F1 FE 00
      SOMA = 1523
29144 38 56 6F 26 00 3A F6 76
      SOMA = 713
29152 E6 02 ED 58 38 5C 28 12
      SOMA = 754

```

```

29160 7D FE 20 28 0D FE 41 D8
      SOMA = 999
29168 FE 55 D0 D6 41 6F ED 58
      SOMA = 1266
29176 7B 5C 29 29 29 19 E5 ED
      SOMA = 829
29184 4B F8 76 CD 2D 65 D1 D5
      SOMA = 1215
29192 CD 6F 68 DD E1 ED 4B F8
      SOMA = 1458
29200 76 79 C6 08 4F 78 47 C5
      SOMA = 912
29208 CD FE 70 C1 ED 43 F8 76
      SOMA = 1434
29216 CD 2D 66 11 00 A0 01 D8
      SOMA = 508
29224 00 7E 12 24 13 10 FA C9
      SOMA = 686
29232 DD 21 00 A0 FE 09 20 D8
      SOMA = 720
29240 3A F6 76 EE 02 32 F6 76
      SOMA = 1076
29248 C3 2E 67 FE 05 D8 ED 4B
      SOMA = 1131
29256 F8 76 20 05 79 D6 08 18
      SOMA = 770
29264 C3 FE 08 28 BC FE 07 78
      SOMA = 1066
29272 20 08 D6 08 30 B8 D6 40
      SOMA = 772
29280 18 B4 06 08 FE C0 38 AE
      SOMA = 1066
29288 D6 C0 18 AA C5 3A 08 77
      SOMA = 882
29296 90 ED 43 07 77 32 07 77
      SOMA = 750
29304 CD 84 72 C1 79 48 47 61
      SOMA = 1053
29312 C8 32 07 77 11 00 00 C5
      SOMA = 890
29320 CD 95 72 C1 78 A7 C8 11
      SOMA = 1155
29328 00 FF ED 44 47 3A 29 77
      SOMA = 849
29336 A7 20 2A C5 D5 CD A9 72
      SOMA = 1139
29344 D1 C1 79 A7 C8 ED 44 4F
      SOMA = 1274
29352 1D 2A FA 76 7C 80 47 9F
      SOMA = 821
29360 AA C0 78 FE C0 D0 7D 61
      SOMA = 1390
29368 4F 9F AB C0 C5 CD 08 69
      SOMA = 1116
29376 C1 CD E6 69 C9 3A 07 77
      SOMA = 1116
29384 A7 C8 79 A7 28 D6 C5 D5
      SOMA = 1324

```

```

29392 CD A9 72 D1 C1 C5 D5 1D
                SOMA = 1329
29400 79 ED 44 4F CD A9 72 D1
                SOMA = 1202
29408 C1 0D 18 E1 3E FF 18 01
                SOMA = 797
29416 AF 32 29 77 CD 0B 70 CD
                SOMA = 918
29424 F4 66 ED 4B F8 76 2A FA
                SOMA = 1316
29432 76 78 94 30 02 ED 44 F5
                SOMA = 986
29440 79 95 30 02 ED 44 CD 60
                SOMA = 986
29448 73 F1 E5 CD 60 73 D1 19
                SOMA = 1235
29456 CD 74 73 7C A7 20 46 7A
                SOMA = 951
29464 B3 28 01 2C DD 21 00 77
                SOMA = 637
29472 7D DD 77 00 A7 20 08 01
                SOMA = 673
29480 00 00 CD A9 72 18 2E 11
                SOMA = 575
29488 B5 00 CD 63 73 7C DD 77
                SOMA = 1064
29496 01 DD 7E 01 CD 60 73 E5
                SOMA = 994
29504 DD 7E 00 CD 60 73 C1 A7
                SOMA = 1123
29512 ED 42 CD 74 73 45 DD 4E
                SOMA = 1107
29520 01 CD 6C 72 DD 7E 01 DD
                SOMA = 997
29528 35 01 A7 20 DC C3 0A 57
                SOMA = 781
29536 5F 16 00 21 00 00 06 08
                SOMA = 164
29544 CB 3F 30 01 19 CB 23 08
                SOMA = 781
29552 12 10 F5 C9 11 01 00 A7
                SOMA = 665
29560 ED 52 06 04 13 13 18 F7
                SOMA = 668
29568 19 CB 3A CB 1B EB C9 3A
                SOMA = 1010
29576 3A 76 18 02 3E FF 6F D9
                SOMA = 847
29584 E5 CD 0B 70 CD F4 66 2A
                SOMA = 1150
29592 FA 76 ED 4B F8 76 11 01
                SOMA = 1064
29600 01 78 94 28 05 30 04 ED
                SOMA = 603
29608 44 15 15 47 79 95 28 05
                SOMA = 496
29616 30 04 ED 44 1D 1D 4F B0
                SOMA = 670

```

```

29624 28 42 79 B8 D5 38 06 16
                SOMA = 708
29632 00 68 41 18 03 1E 00 69
                SOMA = 331
29640 60 D9 ED 4B FA 76 D9 78
                SOMA = 1330
29648 CB 3F 85 38 03 BC 38 07
                SOMA = 709
29656 94 4F D9 D1 D5 18 04 4F
                SOMA = 973
29664 D5 D9 D1 7A 80 47 79 83
                SOMA = 1212
29672 4F CB 0D 30 0A E5 C5 CD
                SOMA = 984
29680 08 69 C1 CD E6 69 E1 D9
                SOMA = 1288
29688 79 10 D7 D1 CD 0A 67 E1
                SOMA = 1104
29696 D9 C9 00 40 2A 02 74 3A
                SOMA = 700
29704 3B 76 E6 04 28 03 2A 97
                SOMA = 647
29712 65 CD E7 64 7C F8 E0 67
                SOMA = 1334
29720 7E E6 20 3E 07 20 02 3E
                SOMA = 553
29728 38 21 53 75 11 00 9B 01
                SOMA = 462
29736 1C 00 ED B0 EB 11 1D 9B
                SOMA = 677
29744 77 01 1B 00 ED 80 6F 0F
                SOMA = 686
29752 0F 0F AD E6 07 AD 6F 67
                SOMA = 327
29760 22 35 9B 22 28 9B EE 06
                SOMA = 715
29768 32 30 9B 21 00 9B ED 4B
                SOMA = 793
29776 F8 76 C5 79 0E 03 CD 7F
                SOMA = 1033
29784 75 23 C1 3E AF 90 0E 03
                SOMA = 743
29792 30 05 ED 44 36 2D 23 0D
                SOMA = 506
29800 CD 7F 76 3A F6 76 E6 04
                SOMA = 1105
29808 20 05 3A 3B 76 E6 03 07
                SOMA = 512
29816 07 EB 6F 26 00 01 3F 75
                SOMA = 572
29824 09 01 04 00 ED B0 11 0E
                SOMA = 458
29832 9B 3A F6 76 E6 05 FE 04
                SOMA = 1071
29840 30 08 3A 3B 76 E6 40 07
                SOMA = 592
29848 07 07 07 6F 26 00 01 6F
                SOMA = 592

```



```

29855 75 09 01 04 00 ED 80 3A
      SOMA = 8802
29854 3B 75 07 38 0B 2A 28 8B
      SOMA = 4388
29872 7C EE 05 5F 67 22 28 9B
      SOMA = 810
29880 3A 3C 75 47 E5 07 C5 30
      SOMA = 790
29888 32 17 9B E5 07 FE 04 17
      SOMA = 746
29895 0F 07 17 0F 07 17 32 33
      SOMA = 161
29904 9B 75 E5 38 FE 20 30 00
      SOMA = 97
29912 F5 07 32 31 9B 0F 0F 07
      SOMA = 552
29920 E5 07 C5 30 32 15 9B CB
      SOMA = 912
29928 00 30 0B 3A 36 9B EE 07
      SOMA = 896
29936 32 35 9B CB 00 30 0B 3A
      SOMA = 76
29944 35 9B EE 47 32 35 9B 3A
      SOMA = 833
29952 3B 75 2E 32 E5 0C FE 0B
      SOMA = 777
29960 38 02 2E 34 E5 04 20 0B
      SOMA = 424
29968 2E 31 7D 32 14 9B 3A F5
      SOMA = 749
29975 75 E5 10 C0 2A 02 74 DD
      SOMA = 937
29984 21 00 9B 01 04 07 E5 0B
      SOMA = 856
29992 DD 7E 1C 0B DD 7E 00 CD
      SOMA = 935
30000 72 58 DD 23 10 F2 C1 E1
      SOMA = 1150
30008 CD 59 55 0D 20 E8 C9 53
      SOMA = 955
30016 45 54 20 56 4F 52 20 55
      SOMA = 948
30024 45 53 20 53 41 4C 54 54
      SOMA = 975
30032 45 58 54 20 20 20 20 20
      SOMA = 413
30040 20 20 20 20 20 20 20 20
      SOMA = 307
30048 41 20 20 20 20 20 4D 20
      SOMA = 934
30055 20 50 20 49 42 46 20 54
      SOMA = 489
30064 45 4C 41 41 54 52 49 20
      SOMA = 545
30072 20 20 20 47 52 41 46 E5
      SOMA = 513
30080 05 00 21 95 75 09 5E E1
      SOMA = 533

```

```

30088 05 2F 93 04 30 FC 53 70
      SOMA = 747
30095 23 0D 20 EB C9 00 01 0A
      SOMA = 527
30104 54 2A 02 74 7C FE 40 20
      SOMA = 734
30112 05 21 50 50 18 03 21 00
      SOMA = 305
30120 40 22 02 74 CD 0D 5B C3
      SOMA = 735
30128 16 75 3A F5 75 EE 10 32
      SOMA = 885
30136 F5 75 18 F0 0D 07 54 49
      SOMA = 885
30144 50 4F 20 40 49 4E 2F 27
      SOMA = 504
30152 42 49 54 20 52 45 53 49
      SOMA = 558
30160 54 27 42 49 54 20 53 49
      SOMA = 530
30168 54 27 27 20 2A 2A 2A 2A
      SOMA = 362
30176 2A 2A 2A 2A 27 27 DD 21
      SOMA = 500
30184 BC 75 CD 31 68 21 C2 40
      SOMA = 854
30192 E5 CD 55 58 3E 87 3D C3
      SOMA = 1081
30200 58 E1 2C 05 08 C5 E5 CD
      SOMA = 1002
30208 2E 57 CD 74 57 FE 02 30
      SOMA = 877
30216 F5 5F 07 07 F5 43 08 0E
      SOMA = 754
30224 30 55 E1 CD 72 68 E5 CD
      SOMA = 1253
30232 E7 54 35 57 E1 C1 10 DD
      SOMA = 1175
30240 05 05 21 C3 58 7E 23 E5
      SOMA = 721
30248 04 0F 0F 0F CB 11 10 F5
      SOMA = 530
30255 79 32 3A 75 C3 0D 5B 03
      SOMA = 555
30264 08 28 AA 00 07 FF 51 31
      SOMA = 738
30272 99 99 51 51 FF 51 7E 58
      SOMA = 1175
30280 5A 5A 55 7E 51 55 AA 55
      SOMA = 877
30288 AA 55 AA 55 AA AA 00 AA
      SOMA = 1020
30295 00 AA 00 AA 00 FF 00 FF
      SOMA = 850
30304 00 FF 00 FF 00 58 11 22
      SOMA = 897
30312 44 55 10 20 40 21 90 48
      SOMA = 555

```



```

30320 24 12 09 64 42 AA AA 44
      SOMA = 588
30328 44 AA AA 11 11 EE 22 33
      SOMA = 765
30336 99 EE 22 33 99 80 80 60
      SOMA = 1011
30344 FF 08 08 08 FF 88 44 20
      SOMA = 772
30352 22 22 44 88 88 22 50 30
      SOMA = 1008
30360 08 22 50 88 05 08 04 7C
      SOMA = 396
30368 A4 25 3E 20 10 43 A5 5A
      SOMA = 633
30376 34 20 5A A5 02 55 58 55
      SOMA = 800
30384 55 55 55 55 53 CC CC 33
      SOMA = 834
30392 33 CC CC 33 33 38 70 50
      SOMA = 553
30400 C1 83 07 0E 1C 22 FA 8A
      SOMA = 795
30408 AA AA AA A8 AF E5 07 38
      SOMA = 1145
30416 AB BA 83 E0 A7 88 55 22
      SOMA = 1134
30424 11 44 22 55 88 08 1C 3E
      SOMA = 438
30432 7F FF 7F 3E 1C A0 BE 62
      SOMA = 1079
30440 FA 0A EB 28 AF 00 01 08
      SOMA = 719
30448 10 18 20 30 40 50 00 00
      SOMA = 264

```

LISTAGEM "DISASSEMBLADA" DOS CÓDIGOS HEXADECIMAIS DO PROGRAMA

[The page contains extremely faint, illegible markings.]

Year	1990	1991	1992	1993	1994	1995	1996	1997	1998	1999	2000	2001	2002	2003	2004	2005	2006	2007	2008	2009	2010	2011	2012	2013	2014	2015	2016	2017	2018	2019	2020	2021	2022	2023	2024	2025	2026	2027	2028	2029	2030	2031	2032	2033	2034	2035	2036	2037	2038	2039	2040	2041	2042	2043	2044	2045	2046	2047	2048	2049	2050	2051	2052	2053	2054	2055	2056	2057	2058	2059	2060	2061	2062	2063	2064	2065	2066	2067	2068	2069	2070	2071	2072	2073	2074	2075	2076	2077	2078	2079	2080	2081	2082	2083	2084	2085	2086	2087	2088	2089	2090	2091	2092	2093	2094	2095	2096	2097	2098	2099	2100
1990	1991	1992	1993	1994	1995	1996	1997	1998	1999	2000	2001	2002	2003	2004	2005	2006	2007	2008	2009	2010	2011	2012	2013	2014	2015	2016	2017	2018	2019	2020	2021	2022	2023	2024	2025	2026	2027	2028	2029	2030	2031	2032	2033	2034	2035	2036	2037	2038	2039	2040	2041	2042	2043	2044	2045	2046	2047	2048	2049	2050	2051	2052	2053	2054	2055	2056	2057	2058	2059	2060	2061	2062	2063	2064	2065	2066	2067	2068	2069	2070	2071	2072	2073	2074	2075	2076	2077	2078	2079	2080	2081	2082	2083	2084	2085	2086	2087	2088	2089	2090	2091	2092	2093	2094	2095	2096	2097	2098	2099	2100	

[illegible]

Year	1990	1991	1992	1993	1994	1995	1996	1997	1998	1999	2000	2001	2002	2003	2004	2005	2006	2007	2008	2009	2010	2011	2012	2013	2014	2015	2016	2017	2018	2019	2020	2021	2022	2023	2024	2025	2026	2027	2028	2029	2030	2031	2032	2033	2034	2035	2036	2037	2038	2039	2040	2041	2042	2043	2044	2045	2046	2047	2048	2049	2050	2051	2052	2053	2054	2055	2056	2057	2058	2059	2060	2061	2062	2063	2064	2065	2066	2067	2068	2069	2070	2071	2072	2073	2074	2075	2076	2077	2078	2079	2080	2081	2082	2083	2084	2085	2086	2087	2088	2089	2090	2091	2092	2093	2094	2095	2096	2097	2098	2099	2100
1990	1991	1992	1993	1994	1995	1996	1997	1998	1999	2000	2001	2002	2003	2004	2005	2006	2007	2008	2009	2010	2011	2012	2013	2014	2015	2016	2017	2018	2019	2020	2021	2022	2023	2024	2025	2026	2027	2028	2029	2030	2031	2032	2033	2034	2035	2036	2037	2038	2039	2040	2041	2042	2043	2044	2045	2046	2047	2048	2049	2050	2051	2052	2053	2054	2055	2056	2057	2058	2059	2060	2061	2062	2063	2064	2065	2066	2067	2068	2069	2070	2071	2072	2073	2074	2075	2076	2077	2078	2079	2080	2081	2082	2083	2084	2085	2086	2087	2088	2089	2090	2091	2092	2093	2094	2095	2096	2097	2098	2099	2100	

```

nop
nop
ld l,e
ld h,h
add hl,sp
ld l,l
ld (hl),b
ld h,h
cp 85
ret nz
ld h,a
nop
nop
ld (hl),l
ld h,h
ld a,d
ld h,h
ld a,a
ld h,h
ld l,c
ld l,d
nop
nop
nop
nop
and 55
or d
ld (hl),l
ld d,c
ld h,a
ld (hl),b
ld h,l
out (55),a
nop
nop
add a,a
ld (hl),a
call po,8A72
ld h,(hl)
jp nz,8E65
ld h,(hl)
add a,55
nop
nop
xor e
ld h,(hl)
xor e
ld h,(hl)
xor e
ld h,(hl)
xor e
ld h,(hl)
xor e
ld h,(hl)
xor e
ld h,(hl)
xor e
ld h,(hl)

```



```

5737 CD1667 call 6716
5738 08 ret z
5739 18F1 jr 672E
5740 CD03A6B call 6B3A
5741 CD0474 call 7404
5742 CD9369 call 6993
5743 CD1667 call 6716
5744 28FF20 jr z, 673D
5745 CD0F66B call 6BFB
5746 03CF66B jp 6BCF
5747 000216A ld ix, 676A
5748 000866B call 6B6B
5749 002866B call 6B28
5750 CD7467 call 6774
5751 7E08 cp 08
5752 330FF9 jr nc, 675B
5753 030FF776 ld (76F7), a
5754 030FE out (FE), a
5755 030D6B jp 6B0D
5756 0000 add hl, bc
5757 0040 inc bc
5758 0040 ld b, d
5759 0040 ld c, a
5760 0040 ld d, d
5761 0040 ld b, h
5762 0041 ld b, c
5763 003F jr nz, 67B2
5764 7E00 daa
5765 0000 call 6B0F
5766 7E07 call 677F
5767 7E00 cp FF
5768 7E00 jr z, 6774
5769 7E00 ret
5770 7E00 ld bc, (7723)
5771 7E03 or 03
5772 7E04 ld b, 00
5773 7E04 ld c, 00
5774 7E04 inc b
5775 7E04 jr nc, 6788
5776 7E04 ld a, b
5777 7E04 cp 06
5778 7E04 ret c
5779 7E04 ld b, 0B
5780 7E04 ld c, c
5781 7E04 or 00
5782 7E04 inc c
5783 7E04 dec b
5784 7E04 jr nc, 6795
5785 7E04 ld a, b
5786 7E04 cp 06
5787 7E04 jr nc, 67A1
5788 7E04 ld a, FF
5789 7E04 ret
5790 7E04 cp 0A
5791 7E04 ret, c
5792 7E04 xort, c
5793 7E04 ret

```

```

57A6 3AF676 ld a, (76F6)
57A7 0F rrc a
57A8 D0 ret nc
57A9 3F cc f
57AA 17 rla
57AB 32F676 ld (76F6), a
57AC 2100F8 ld hl, F800
57AD 1100F8 ld de, F800
57AE 010003 ld bc, 0300
57AF 0DB0 ld ir
57B0 C9 ret
57B1 0601 ld b, 01
57B2 1802 jr 67C2
57B3 0602 ld b, 02
57B4 3A3B76 ld a, (763B)
57B5 E840 and 40
57B6 C0 ret nz
57B7 3AF676 ld a, (76F6)
57B8 210D6B ld hl, 6B0D
57B9 0000 push hl
57BA 0601 and 01
57BB 20D3 jr nz, 67A6
57BC 03 push bc
57BD 2100F8 ld hl, F800
57BE 1100F8 ld de, F800
57BF 010003 ld bc, 0300
57C0 0DB0 ld ir
57C1 3AF676 ld a, (76F6)
57C2 0601 or 01
57C3 32F676 ld (76F6), a
57C4 C1 pop bc
57C5 06 dec b
57C6 2026 jr nz, 6811
57C7 2101F8 ld hl, F801
57C8 1100F8 ld de, F800
57C9 33638 ld (hl), 38
57CA 0000 dec hl
57CB 33676 ld (hl), 76
57CC 011E00 ld bc, 001E
57CD 0DB0 ld ir
57CE 0603 inc hl
57CF 00DA0 ld i, 00
57D0 2E00 ld l, 00
57D1 011F00 ld bc, 001F
57D2 0DB0 ld ir
57D3 2100F8 ld hl, F800
57D4 1140F8 ld de, F840
57D5 01C002 ld bc, 02C0
57D6 0DB0 ld ir
57D7 C9 ret
57D8 2100F8 ld hl, F800
57D9 110040 ld de, 4000
57DA 011810 ld bc, 1018
57DB 7E ld a, (hl)
57DC 06BF and BF
57DD 0000 or d
57DE 77 ld (hl), a
57DF 23 inc hl

```



```

00000000 ld a,0e
00000001 xor a,0e
00000002 and 40
00000003 or 0170
00000004 ld (763B),a
00000005 push de
00000006 ld a,30F
00000007 ld (763C),a
00000008 push bc
00000009 call 908
0000000A pop bc
0000000B pop de
0000000C ld a,(763B)
0000000D xor a,70
0000000E and 70
0000000F xor a,70
00000010 ld e,a
00000011 ld (763B),de
00000012 ret
00000013 call 6957
00000014 ret c
00000015 call 65F4
00000016 ld a,c
00000017 and 40
00000018 jr nz,69F4
00000019 ld a,FE
0000001A rrc a
0000001B djnz 69D4
0000001C ld a,(hl)
0000001D bit 0,c
0000001E jr nz,69DE
0000001F and b,c
00000020 bit 1,c
00000021 jr nz,69E4
00000022 xor b
00000023 call (hl),a
00000024 ret
00000025 push bc
00000026 call 908
00000027 call 908
00000028 pop bc
00000029 ret
0000002A ld a,c
0000002B and 40
0000002C jr nz,69D8
0000002D ld a,(763C)
0000002E ld b,a
0000002F ld e,(hl)
00000030 bit 0,c
00000031 jr nz,69FE
00000032 xor a
00000033 bit 1,c
00000034 jr nz,6A03
00000035 xor b
00000036 ld (hl),a
00000037 ret

```

```

00000038 call 65F4
00000039 ld a,c
0000003A and 40
0000003B jr nz,6A1C
0000003C ld a,7E
0000003D rrc a
0000003E djnz 6A0F
0000003F ld b,a
00000040 call 69D8
00000041 ld b,a
00000042 call 64F1
00000043 ld (hl),b
00000044 ret
00000045 ld a,l
00000046 and DE
00000047 ld l,a
00000048 call 69F4
00000049 inc hl
0000004A ld (hl),a
0000004B ld de,0020
0000004C add hl,de
0000004D ld (hl),a
0000004E dec l
0000004F ld (hl),a
00000050 ret
00000051 ld a,c
00000052 call 65F4
00000053 ld a,c
00000054 and 40
00000055 jr nz,6A48
00000056 ld a,1E
00000057 rrc a
00000058 djnz 6A37
00000059 ld b,a
0000005A call 69D8
0000005B ld b,03
0000005C ld c,a
0000005D call 64F1
0000005E ld (hl),c
0000005F djnz 6A41
00000060 ret
00000061 ld a,l
00000062 and 9C
00000063 ld l,a
00000064 call 69F4
00000065 ld bc,0003
00000066 ld a,l
00000067 ld d,h
00000068 inc de
00000069 ld a,04
0000006A push hl
0000006B ld ir
0000006C pop de
0000006D ld hl,0020
0000006E add hl,de
0000006F ex de,hl
00000070 push de
00000071 ld bc,0004
00000072 dec a

```



```

00000000 ld de, (772E)
00000001 and bc, (783B)
00000002 and 08
00000003 jmp 8000
00000004 jnc 8038
00000005 ld hl, E000
00000006 ld de, 4000
00000007 ld bc, 1800
00000008 ld r
00000009 ret
0000000A ld a, (783B)
0000000B and 14
0000000C cp 10
0000000D ret nz
0000000E ld hl, (772E)
0000000F call 84E7
00000010 push hl
00000011 ld a, (7837)
00000012 ld (hly), a
00000013 ld a, (783B)
00000014 and 08
00000015 ld bc, 000F
00000016 jr z, 888A
00000017 ld bc, 0007
00000018 ld e, l
00000019 ld d, h
0000001A inc de
0000001B ld ir
0000001C ld B0
0000001D pop hl
0000001E ld a, (783B)
0000001F and 08
00000020 ld bc, 0B10
00000021 jr z, 888D
00000022 ld bc, 0508
00000023 push bc
00000024 push hl
00000025 ld de, 0020
00000026 ex de, hl
00000027 add hl, de
00000028 ex de, hl
00000029 ld b, 00
0000002A ld ir
0000002B pop hl
0000002C ld de, 0020
0000002D add hl, de
0000002E pop bc
0000002F djnz 888D
00000030 ld a, (7837)
00000031 ld c, a
00000032 add a, 03
00000033 xor c
00000034 and 07
00000035 xor c
00000036 sub 08
00000037 and 3F
00000038 ld (7837), a
00000039 xor a
0000003A ret

```

[illegible]

[illegible]

APENDICE A

CONVERSÕES DE VALORES DECIMAIS, BINÁRIOS E HEXADECIMAIS

<i>Hexadecimal</i>	<i>Decimal</i>	<i>Binário</i>
00	0	00000000
01	1	00000001
02	2	00000010
03	3	00000011
04	4	00000100
05	5	00000101
06	6	00000110
07	7	00000111
08	8	00001000
09	9	00001001
0A	10	00001010
0B	11	00001011
0C	12	00001100
0D	13	00001101
0E	14	00001110
0F	15	00001111
10	16	00010000
11	17	00010001
12	18	00010010
13	19	00010011
14	20	00010100
15	21	00010101
16	22	00010110

Hexadecimal	Decimal	Binário
17	23	00010111
18	24	00011000
19	25	00011001
1A	26	00011010
1B	27	00011011
1C	28	00011100
1D	29	00011101
1E	30	00011110
1F	31	00011111
20	32	00100000
21	33	00100001
22	34	00100010
23	35	00100011
24	36	00100100
25	37	00100101
26	38	00100110
27	39	00100111
28	40	00101000
29	41	00101001
2A	42	00101010
2B	43	00101011
2C	44	00101100
2D	45	00101101
2E	46	00101110
2F	47	00101111
30	48	00110000
31	49	00110001
32	50	00110010
33	51	00110011
34	52	00110100
35	53	00110101
36	54	00110110
37	55	00110111
38	56	00111000
39	57	00111001
3A	58	00111010
3B	59	00111011
3C	60	00111100
3D	61	00111101
3E	62	00111110
3F	63	00111111

Hexadecimal	Decimal	Binário
40	64	01000000
41	65	01000001
42	66	01000010
43	67	01000011
44	68	01000100
45	69	01000101
46	70	01000110
47	71	01000111
48	72	01001000
49	73	01001001
4A	74	01001010
4B	75	01001011
4C	76	01001100
4D	77	01001101
4E	78	01001110
4F	79	01001111
50	80	01010000
51	81	01010001
52	82	01010010
53	83	01010011
54	84	01010100
55	85	01010101
56	86	01010110
57	87	01010111
58	88	01011000
59	89	01011001
5A	90	01011010
5B	91	01011011
5C	92	01011100
5D	93	01011101
5E	94	01011110
5F	95	01011111
60	96	01100000
61	97	01100001
62	98	01100010
63	99	01100011
64	100	01100100
65	101	01100101
66	102	01100110
67	103	01100111
68	104	01101000

<i>Hexadecimal</i>	<i>Decimal</i>	<i>Binário</i>
69	105	01101001
6A	106	01101010
6B	107	01101011
6C	108	01101100
6D	109	01101101
6E	110	01101110
6F	111	01101111
70	112	01110000
71	113	01110001
72	114	01110010
73	115	01110011
74	116	01110100
75	117	01110101
76	118	01110110
77	119	01110111
78	120	01111000
79	121	01111001
7A	122	01111010
7B	123	01111011
7C	124	01111100
7D	125	01111101
7E	126	01111110
7F	127	01111111
80	128	10000000
81	129	10000001
82	130	10000010
83	131	10000011
84	132	10000100
85	133	10000101
86	134	10000110
87	135	10000111
88	136	10001000
89	137	10001001
8A	138	10001010
8B	139	10001011
8C	140	10001100
8D	141	10001101
8E	142	10001110
8F	143	10001111
90	144	10010000
91	145	10010001

<i>Hexadecimal</i>	<i>Decimal</i>	<i>Binário</i>
92	146	10010010
93	147	10010011
94	148	10010100
95	149	10010101
96	150	10010110
97	151	10010111
98	152	10011000
99	153	10011001
9A	154	10011010
9B	155	10011011
9C	156	10011100
9D	157	10011101
9E	158	10011110
9F	159	10011111
A0	160	10100000
A1	161	10100001
A2	162	10100010
A3	163	10100011
A4	164	10100100
A5	165	10100101
A6	166	10100110
A7	167	10100111
A8	168	10101000
A9	169	10101001
AA	170	10101010
AB	171	10101011
AC	172	10101100
AD	173	10101101
AE	174	10101110
AF	175	10101111
B0	176	10110000
B1	177	10110001
B2	178	10110010
B3	179	10110011
B4	180	10110100
B5	181	10110101
B6	182	10110110
B7	183	10110111
B8	184	10111000
B9	185	10111001
BA	186	10111010

<i>Hexadecimal</i>	<i>Decimal</i>	<i>Binário</i>
BB	187	10111011
BC	188	10111100
BD	189	10111101
BE	190	10111110
BF	191	10111111
C0	192	11000000
C1	193	11000001
C2	194	11000010
C3	195	11000011
C4	196	11000100
C5	197	11000101
C6	198	11000110
C7	199	11000111
C8	200	11001000
C9	201	11001001
CA	202	11001010
CB	203	11001011
CC	204	11001100
CD	205	11001101
CE	206	11001110
CF	207	11001111
D0	208	11010000
D1	209	11010001
D2	210	11010010
D3	211	11010011
D4	212	11010100
D5	213	11010101
D6	214	11010110
D7	215	11010111
D8	216	11011000
D9	217	11011001
DA	218	11011010
DB	219	11011011
DC	220	11011100
DD	221	11011101
DE	222	11011110
DF	223	11011111
E0	224	11100000
E1	225	11100001
E2	226	11100010
E3	227	11100011

E4	228	11100100
E5	229	11100101
E6	230	11100110
E7	231	11100111
E8	232	11101000
E9	233	11101001
EA	234	11101010
EB	235	11101011
EC	236	11101100
ED	237	11101101
EE	238	11101110
EF	239	11101111
F0	240	11110000
F1	241	11110001
F2	242	11110010
F3	243	11110011
F4	244	11110100
F5	245	11110101
F6	246	11110110
F7	247	11110111
F8	248	11111000
F9	249	11111001
FA	250	11111010
FB	251	11111011
FC	252	11111100
FD	253	11111101
FE	254	11111110
FF	255	11111111

APÊNDICE B

CÓDIGOS DE OPERAÇÃO DO Z80
ORDENADOS POR MNEMÔNICAS

<i>Código de Operação</i>	<i>Códigos Hexadecimais</i>	<i>Códigos Decimais</i>
ADC A, (HL)	8E	142
ADC A, (IX + d)	DD8Edd	221, 142, xx
ADC A, (IY + d)	FD8Edd	253, 142, xx
ADC A, A	8F	143
ADC A, B	88	136
ADC A, C	89	137
ADC A, D	8A	138
ADC A, E	8B	139
ADC A, H	8C	140
ADC A, L	8D	141
ADC A, xx	CExx	206, xx
ADC HL, BC	ED4A	237, 74
ADC HL, DE	ED5A	237, 90
ADC HL, HL	ED6A	237, 106
ADC HL, SP	ED7A	237, 122
ADD A, (HL)	86	134
ADD A, (IX + d)	DD86dd	221, 134, xx
ADD A, (IY + d)	FD86dd	253, 134, xx
ADD A, A	87	135
ADD A, B	80	128
ADD A, C	81	129
ADD A, D	82	130

<i>Código de Operação</i>	<i>Códigos Hexadecimais</i>	<i>Códigos Decimais</i>
ADD A, E	83	131
ADD A, H	84	132
ADD A, L	85	133
ADD A, xx	C6xx	198, xx
ADD HL, BC	09	9
ADD HL, DE	19	25
ADD HL, HL	29	41
ADD HL, SP	39	57
ADD IX, BC	DD09	221, 9
ADD IX, DE	DD19	221, 25
ADD IX, IX	DD29	221, 41
ADD IX, SP	DD39	221, 57
ADD IY, BC	FD09	253, 9
ADD IY, DE	FD19	253, 25
ADD IY, IX	FD29	253, 41
ADD IY, SP	FD39	253, 57
AND (HL)	A6	166
AND (IX + d)	DDA6dd	221, 166, xx
AND (IY + d)	FDA6dd	253, 166, xx
AND A	A7	167
AND B	A0	160
AND C	A1	161
AND D	A2	162
AND E	A3	163
AND H	A4	164
AND L	A5	165
AND xx	E6xx	230, xx
BIT 0, (HL)	CB46	203, 70
BIT 0, (IX + d)	DDCBdd46	221, 203, xx, 70
BIT 0, (IY + d)	FDCBdd46	253, 203, xx, 70
BIT 0, A	CB47	203, 71
BIT 0, B	CB40	203, 64
BIT 0, C	CB41	203, 65
BIT 0, D	CB42	203, 66
BIT 0, E	CB43	203, 67
BIT 0, H	CB44	203, 68
BIT 0, L	CB45	203, 69

<i>Código de Operação</i>	<i>Códigos Hexadecimais</i>	<i>Códigos Decimais</i>
BIT 1, (HL)	CB4E	203, 70
BIT 1, (IX + d)	DDCBdd4E	221, 203, xx, 70
BIT 1, (IY + d)	FDCBdd4E	253, 203, xx, 70
BIT 1, A	CB4F	203, 79
BIT 1, B	CB48	203, 72
BIT 1, C	CB49	203, 73
BIT 1, D	CB4A	203, 74
BIT 1, E	CB4B	203, 75
BIT 1, H	CB4C	203, 76
BIT 1, L	CB4D	203, 77
BIT 2, (HL)	CB56	203, 86
BIT 2, (IX + d)	DDCBdd56	221, 203, xx, 86
BIT 2, (IY + d)	FDCBdd56	253, 203, xx, 86
BIT 2, A	CB57	203, 87
BIT 2, B	CB50	203, 80
BIT 2, C	CB51	203, 81
BIT 2, D	CB52	203, 82
BIT 2, E	CB53	203, 83
BIT 2, H	CB54	203, 84
BIT 2, L	CB55	203, 85
BIT 3, (HL)	CB5E	203, 94
BIT 3, (IX + d)	DDCBdd5E	221, 203, xx, 94
BIT 3, (IY + d)	FDCBdd5E	253, 203, xx, 94
BIT 3, A	CB5F	203, 95
BIT 3, B	CB58	203, 88
BIT 3, C	CB59	203, 89
BIT 3, D	CB5A	203, 90
BIT 3, E	CB5B	203, 91
BIT 3, H	CB5C	203, 92
BIT 3, L	CB5D	203, 93
BIT 4, (HL)	CB66	203, 102
BIT 4, (IX + d)	DDCBdd66	221, 203, xx, 102
BIT 4, (IY + d)	FDCBdd66	253, 203, xx, 102
BIT 4, A	CB67	203, 103
BIT 4, B	CB60	203, 96
BIT 4, C	CB61	203, 97
BIT 4, D	CB62	203, 98
BIT 4, E	CB63	203, 99

<i>Código de Operação</i>	<i>Códigos Hexadecimais</i>	<i>Códigos Decimais</i>
BIT 4, H	CB64	203, 100
BIT 4, L	CB65	202, 101
BIT 5, (HL)	CB6E	203, 110
BIT 5, (IX + d)	DDCBdd6E	221, 203, xx, 110
BIT 5, (IY + d)	FDCBdd6E	253, 203, xx, 110
BIT 5, A	CB6F	203, 111
BIT 5, B	CB68	203, 104
BIT 5, C	CB69	203, 105
BIT 5, D	CB6A	203, 106
BIT 5, E	CB6B	203, 107
BIT 5, H	CB6C	203, 108
BIT 5, L	CB6D	203, 109
BIT 6, (HL)	CB76	203, 118
BIT 6, (IX + d)	DDCBdd76	221, 203, xx, 118
BIT 6, (IY + d)	FDCBdd76	253, 203, xx, 118
BIT 6, A	CB77	203, 119
BIT 6, B	CB70	203, 112
BIT 6, C	CB71	203, 113
BIT 6, D	CB72	203, 114
BIT 6, E	CB73	203, 115
BIT 6, H	CB74	203, 116
BIT 6, L	CB75	203, 117
BIT 7, (HL)	CB7E	203, 116
BIT 7, (IX + d)	DDCBdd7E	221, 203, xx, 116
BIT 7, (IY + d)	FDCBdd7E	253, 203, xx, 116
BIT 7, A	CB7F	203, 127
BIT 7, B	CB78	203, 120
BIT 7, C	CB79	203, 121
BIT 7, D	CB7A	203, 122
BIT 7, E	CB7B	203, 123
BIT 7, H	CB7C	203, 124
BIT 7, L	CB7D	203, 125
CALL C, xxxx	DCxxxx	220, xx, xx
CALL M, xxxx	FCxxxx	252, xx, xx
CALL NC, xxxx	D4xxxx	212, xx, xx
CALL NZ, xxxx	C4xxxx	196, xx, xx
CALL P, xxxx	F4xxxx	244, xx, xx

<i>Código de Operação</i>	<i>Códigos Hexadecimais</i>	<i>Códigos Decimais</i>
CALL PE, xxxx	ECxxxx	236, xx, xx
CALL PO, xxxx	E4xxxx	228, xx, xx
CALL Z, xxxx	CCxxxx	204, xx, xx
CP (HL)	BE	190
CP (IX + d)	DDBEdd	221, 190, xx
CP (IY + d)	FDBEd	253, 190, xx
CP A	BF	191
CP B	B8	184
CP C	B9	185
CP D	BA	186
CP E	BB	187
CP H	BC	188
CP L	BD	189
CP xx	FExx	254, xx
CPD	EDA9	237, 169
CPDR	EDB9	237, 185
CPI	EDA1	237, 161
CPIR	EDB1	237, 177
CPL	2F	47
DAA	27	39
DEC (HL)	35	53
DEC (IX + d)	DD35dd	221, 53, xx
DEC (IY + d)	FD35dd	253, 53, xx
DEC A	3D	61
DEC B	05	5
DEC BC	0b	11
DEC C	0D	13
DEC D	15	21
DEC DE	1B	27
DEC E	1D	29
DEC H	25	37
DEC HL	2B	43
DEC IX	DD2B	221, 43
DEC IY	FD2B	253, 43
DEC L	2D	45
DEC SP	3B	59
DI	F3	243

<i>Código de Operação</i>	<i>Códigos Hexadecimais</i>	<i>Códigos Decimais</i>
DJNZ xx	10xx	16, xx
EI	FB	251
EX (SP), HL	E3	227
EX (SP), IX	DDE3	221, 227
EX (SP), IY	FDE3	253, 227
EX AF, AF'	08	8
EX DE, HL	EB	235
EXX	D9	217
HALT	76	118
IM 0	ED46	237, 70
IM 1	ED56	237, 86
IM 2	ED5E	237, 94
IN A, (C)	ED 78	237, 120
IN A, (xx)	DBxx	219, xx
IN B, (C)	ED40	237, 64
IN C, (C)	ED48	237, 72
IN D, (C)	ED50	237, 80
IN E, (C)	ED58	237, 88
IN H, (C)	ED60	237, 96
IN L, (C)	ED68	237, 104
INC (HL)	34	52
INC (IX + d)	DD34dd	221, 52, xx
INC (IY + d)	FD34dd	253, 52, xx
INC A	3C	60
INC B	04	4
INC BC	03	3
INC C	0C	12
INC D	14	20
INC DE	13	19
INC E	1C	28
INC H	24	36
INC HL	23	35
INC IX	DD23	221, 35
INC IY	FD23	253, 35
INC L	2C	44

<i>Código de Operação</i>	<i>Códigos Hexadecimais</i>	<i>Códigos Decimais</i>
INC SP	33	51
IND	EDAA	237, 170
INDR	EDBA	237, 186
INI	EDA2	237, 162
INIR	EDB2	237, 178
JP (HL)	E9	233
JP (IX)	DDE9	221, 233
JP (IY)	FDE9	253, 233
JP C, xxxx	DAxxxx	218, xx, xx
JP M, xxxx	FAxxxx	250, xx, xx
JP NC, xxxx	D2xxxx	210, xx, xx
JP NZ, xxxx	C2xxxx	194, xx, xx
JP P, xxxx	F2xxxx	242, xx, xx
JP PE, xxxx	EAxxxx	234, xx, xx
JP PO, xxxx	E2xxxx	226, xx, xx
JP xxxx	C3xxxx	195, xx, xx
JP Z, xxxx	CAxxxx	202, xx, xx
JR C, xx	38xx	56, xx
JR NC, xx	30xx	48, xx
JR NZ, xx	20xx	32, xx
JR xx	18xx	24, xx
JR Z, xx	28xx	40, xx
LD (BC), A	02	2
LD (DE), A	12	18
LD HL, (xxxx)	2Axxxx	42, xx, xx
LD (HL), A	77	119
LD (HL), B	70	112
LD (HL), C	71	113
LD (HL), D	72	114
LD (HL), E	73	115
LD (HL), H	74	116
LD (HL), L	75	117
LD (HL), xx	36xx	54, xx
LD (IX + d), A	DD77dd	221, 119, xx
LD (IX + d), B	DD70dd	221, 112, xx
LD (IX + d), C	DD71dd	221, 113, xx
LD (IX + d), D	DD72dd	221, 114, xx
LD (IX + d), E	DD73dd	221, 115, xx
LD (IX + d), H	DD74dd	221, 116, xx

<i>Código de Operação</i>	<i>Códigos Hexadecimais</i>	<i>Códigos Decimais</i>
LD (IX + d), L	DD75dd	221, 117, xx
LD (IX + d), xx	DD36ddxx	221, 54, xx, xx
LD (IY + d), A	FD77dd	253, 119, xx
LD (IY + d), B	FD70dd	253, 112, xx
LD (IY + d), C	FD71dd	253, 113, xx
LD (IY + d), D	FD72dd	253, 114, xx
LD (IY + d), E	FD73dd	253, 115, xx
LD (IY + d), H	FD74dd	253, 116, xx
LD (IY + d), L	FD75dd	253, 117, xx
LD (IY + d), xx	FD36ddxx	253, 54, xx, xx
LD (xxxx), A	32xxxx	50, xx, xx
LD (xxxx), BC	ED43xxxx	237, 67, xx, xx
LD (xxxx), DE	ED53xxxx	237, 83, xx, xx
LD (xxxx), HL	22xxxx	34, xx, xx
LD (xxxx), IX	DD22xxxx	221, 34, xx, xx
LD (xxxx), IY	Fd22xxxx	253, 34, xx, xx
LD (xxxx), SP	ED73xxxx	237, 115, xx, xx
LD A, (BC)	0A	10
LD A, (de)	1A	26
LD A, (HL)	7E	126
LD A, (IX + d)	DD7Edd	221, 126, xx
LD A, (IY + d)	FD7Edd	253, 126, xx
LD A, (xxxx)	3Axxxx	58, xx, xx
LD A, A	7F	127
LD A, B	78	120
LD A, C	79	121
LD A, D	7A	122
LD A, E	7B	123
LD A, H	7C	124
LD A, I	ED57	237, 87
LD A, L	7D	125
LD A, R	ED5F	237, 85
LD A, xx	3Exx	62, xx
LD B, (HL)	46	70
LD B, (IX + d)	DD46dd	221, 70, xx
LD B, (IY + d)	FD46dd	253, 70, xx
LD B, A	47	71
LD B, B	40	64

<i>Código de Operação</i>	<i>Códigos Hexadecimais</i>	<i>Códigos Decimais</i>
LD B, C	41	65
LD B, D	42	66
LD B, E	43	67
LD B, H	44	68
LD B, L	45	69
LD B, xx	06xx	6, xx
LD BC, (xxxx)	ED4Bxxxx	237, 75, xx, xx
LD BC, xxxx	01xxxx	1, xx, xx
LD C, (HL)	4E	78
LD C, (IX + d)	DD4Edd	221, 78, xx
LD C, (IY + d)	FD4Edd	253, 78, xx
LD C, A	4F	79
LD C, B	48	72
LD C, C	49	73
LD C, D	4A	74
LD C, E	4B	75
LD C, H	4C	76
LD C, L	4D	77
LD C, xx	0Exx	14, xx
LD D, (HL)	56	86
LD D, (IX + d)	DD56dd	221, 86, xx
LD D, (IY + d)	FD56dd	253, 86, xx
LD D, A	57	87
LD D, B	50	80
LD D, C	51	81
LD D, D	52	82
LD D, E	53	83
LD D, H	54	84
LD D, L	55	85
LD D, xx	16xx	22, xx
LD DE, (xxxx)	ED5Bxxxx	237, 91, xx, xx
LD DE, xxxx	11xxxx	17, xx, xx
LD E, (HL)	5E	94
LD E, (IX + d)	DD5Edd	221, 94, xx
LD E, (IY + d)	FD5Edd	253, 94, xx
LD E, A	5F	95
LD E, B	58	88
LD E, C	59	89
LD E, D	5A	90

<i>Código de Operação</i>	<i>Códigos Hexadecimais</i>	<i>Códigos Decimais</i>
LD E, E	5B	91
LD E, H	5C	92
LD E, L	5D	93
LD E, xx	1Exx	30, xx
LD H, (HL)	66	102
LD H, (IX + d)	DD66dd	221, 102, xx
LD H, (IY + d)	FD66dd	253, 102, xx
LD H, A	67	103
LD H, B	60	96
LD H, C	61	97
LD H, D	62	98
LD H, E	63	99
LD H, L	64	100
LD H, xx	65	101
LD HL, xxxx	26xx	38, xx
	21xxxx	33, xx, xx
LD I, A	ED47	237, 71
LD IX, xxxx	DD21xxxx	221, 33, xx, xx
LD IX (xxxx)	DD2Axxxx	221, 42, xx, xx
LD IY, xxxx	FD21xxxx	253, 33, xx, xx
LD IY, (xxxx)	FD2Axxxx	253, 42, xx, xx
LD L, (HL)	6E	110
LD L, (IX + d)	DD6Edd	221, 110, xx
LD L, (IY + d)	FD6Edd	253, 110, xx
LD L, A	6F	111
LD L, B	68	104
LD L, C	69	105
LD L, D	6A	106
LD L, E	6B	107
LD L, H	6C	108
LD L, L	6D	109
LD L, xx	2Exx	46, xx
LD R, A	ED4F	237, 79
LD SP, (xxxx)	ED7Bxxxx	237, 123, xx, xx
LD SP, HL	F9	249
LD SP, IX	DDF9	221, 249

<i>Código de Operação</i>	<i>Códigos Hexadecimais</i>	<i>Códigos Decimais</i>
LD SP, IY	FDF9	253, 249
LD SP, xxxx	31xxxx	49, xx, xx
LDD	EDA8	237, 168
LDDR	EDB8	237, 184
LDI	EDA0	237, 160
LDIR	EDB0	237, 176
NEG	ED44	237, 68
NOP	00	0
OR (HL)	B6	182
OR (IX + d)	DDB6dd	221, 182, xx
OR (IY + d)	FDB6dd	253, 182, xx
OR A	B7	183
OR B	B0	176
OR C	B1	177
OR D	B2	178
OR E	B3	179
OR H	B4	180
OR L	B5	181
OR xx	F6xx	246, xx
OTDR	EDBB	237, 187
OTIR	EDB3	237, 179
OUT (C), A	ED79	237, 121
OUT (C), B	ED41	237, 65
OUT (C), C	ED49	237, 73
OUT (C), D	ED51	237, 81
OUT (C), E	ED59	237, 89
OUT (C), H	ED61	237, 97
OUT (C), L	ED69	237, 105
OUT (xx), A	D3xx	211, xx
OUTD	EDAB	237, 171
OUTI	EDA3	237, 163
POP AF	F1	241
POP BC	C1	193
POP DE	D1	209

POP HL	E1	225
POP IX	DDE1	221, 225
POP IY	FDE1	253, 225
PUSH AF	F5	245
PUSH BC	C5	197
PUSH DE	D5	213
PUSH HL	E5	229
PUSH IX	DDE5	221, 229
PUSH IY	FDE5	253, 229
RES 0, (HL)	CB86	203, 134
RES 0, (IX + d)	DDCBdd86	221, 203, xx, 134
RES 0, (IY + d)	FDCBdd86	253, 203, xx, 134
RES 0, A	CB87	203, 135
RES 0, B	CB80	203, 128
RES 0, C	CB81	203, 129
RES 0, D	CB82	203, 130
RES 0, E	CB83	203, 131
RES 0, H	CB84	203, 132
RES 0, L	CB85	203, 133
RES 1, (HL)	CB8E	203, 142
RES 1, (IX + d)	DDCBdd8E	221, 203, xx, 142
RES 1, (IY + d)	FDCBdd8E	253, 203, xx, 142
RES 1, A	CB8F	203, 143
RES 1, B	CB88	203, 136
RES 1, C	CB89	203, 137
RES 1, D	CB8A	203, 138
RES 1, E	CB8B	203, 139
RES 1, H	CB8C	203, 140
RES 1, L	CB8D	203, 141
RES 2, (HL)	CB96	203, 150
RES 2, (IX + d)	DDCBdd96	221, 203, xx, 150
RES 2, (IY + d)	FDCBdd96	253, 203, xx, 150
RES 2, A	CB97	203, 151
RES 2, B	CB90	203, 144
RES 2, C	CB91	203, 145
RES 2, D	CB92	203, 146
RES 2, E	CB93	203, 147
RES 2, H	CB94	203, 148

RES 2, L	CB95	203, 149
RES 3, (HL)	CB9E	203, 158
RES 3, (IX + d)	DDCBdd9E	221, 203, xx, 158
RES 3, (IY + d)	FDCBdd9E	253, 203, xx, 158
RES 3, A	CB9F	203, 159
RES 3, B	CB98	203, 152
RES 3, C	CB99	203, 153
RES 3, D	CB9A	203, 154
RES 3, E	CB9B	203, 155
RES 3, H	CB9C	203, 156
RES 3, L	CB9D	203, 157
RES 4, (HL)	CBA6	203, 166
RES 4, (X + d)	DDCBddA6	221, 203, xx, 166
RES 4, (IY + d)	FDCBddA6	253, 203, xx, 166
RES 4, A	CBA7	203, 167
RES 4, B	CBA0	203, 160
RES 4, C	CBA1	203, 161
RES 4, D	CBA2	203, 162
RES 4, E	CBA3	203, 163
RES 4, H	CBA4	203, 164
RES 4, L	CBA5	203, 165
RES 5, (HL)	CBAE	203, 174
RES 5, (IX + d)	DDCBddAE	221, 203, xx, 174
RES 5, (IY + d)	FDCBddAE	253, 203, xx, 174
RES 5, A	CBAF	203, 175
RES 5, B	CBA8	203, 168
RES 5, C	CBA9	203, 169
RES 5, D	CBAA	203, 170
RES 5, E	CBAB	203, 171
RES 5, H	CBAC	203, 172
RES 5, L	CBAD	203, 173
RES 6, (HL)	CBB6	203, 182
RES 6, (IX + d)	DDCBddB6	221, 203, xx, 182
RES 6, (IY + d)	FDCBddB6	253, 203, xx, 182
RES 6, A	CBB7	203, 183
RES 6, B	CBB0	203, 176
RES 6, C	CBB1	203, 177

RES 6, D	CBB2	203, 178
RES 6, E	CBB3	203, 179
RES 6, H	CBB4	203, 180
RES 6, L	CBB5	203, 181
RES 7, (HL)	CBBE	203, 190
RES 7, (IX + d)	DDCBddBE	221, 203, xx, 190
RES 7, (IY + d)	FDCBddBE	253, 203, xx, 190
RES 7, A	CBBF	203, 191
RES 7, B	CBB8	203, 184
RES 7, C	CBB9	203, 185
RES 7, D	CBBA	203, 186
RES 7, E	CBBB	203, 187
RES 7, H	CBBC	203, 188
RES 7, L	CBBD	203, 189
RET	C9	201
RET C	D8	216
RET M	F8	248
RET NC	D0	208
RET NZ	C0	192
RET P	F0	240
RET PE	E8	232
RET PO	E0	224
RET Z	C8	200
RETI	ED4D	237, 77
RETN	ED45	237, 69
RL (HL)	CB16	203, 22
RL (IX + d)	DDCBdd16	221, 203, xx, 22
RL (IY + d)	FDCBdd16	253, 203, xx, 22
RL A	CB17	203, 23
RL B	CB10	203, 16
RL C	CB11	203, 17
RL D	CB12	203, 18
RL E	CB13	203, 19
RL H	CB14	203, 20
RL L	CB15	203, 21
RLA	17	23
RLC (HL)	CB06	203, 6

RLC (IX + d)	DDCBdd06	221, 203, xx, 6
RLD (IX + d)	FDCBdd06	253, 203, xx, 6
RLC A	CB07	203, 7
RLC B	CB00	203, 0
RLC C	CB01	203, 1
RLC D	CB02	203, 2
RLC E	CB03	203, 3
RLC H	CB04	203, 4
RLC L	CB05	203, 5
RLCA	07	7
RLD	ED6F	237, 111
RR (HL)	CB1E	203, 30
RR (IX + d)	DDCBdd1E	221, 203, xx, 1E
RR (IX + d)	FDCBdd1E	253, 203, xx, 1E
RR A	CB1F	203, 31
RR B	CB18	203, 24
RR C	CB19	203, 25
RR D	CB1A	203, 26
RR E	CB1B	203, 27
RR H	CB1C	203, 27
RR L	CB1D	203, 28
RRA	1F	31
RRC (HL)	CB0E	203, 14
RRC (IX + d)	DDCBdd0E	221, 203, xx, 14
RRC (IX + d)	FDCBdd0E	253, 203, xx, 14
RRC A	CB0F	203, 15
RRC B	CB08	203, 8
RRC C	CB09	203, 9
RRC D	CB0A	203, 10
RRC E	CB0B	203, 11
RRC H	CB0C	203, 12
RRC L	CB0D	203, 13
RRCA	0F	15
RRD	ED67	237, 103
RST 0	C7	199
RST 10h	D7	215
RST 18h	DF	223

RST 20h	E7	231
RST 28h	EF	239
RST 30h	F7	247
RST 38h	FF	255
RST 8	CF	207
SBC A, (HL)	9E	158
SBC A, (IX + d)	DD9Edd	221, 158, xx
SBC A, (IX + d)	FD9Edd	253, 158, xx
SBC A, A	9F	159
SBC A, B	98	152
SBC A, C	99	153
SBC A, D	9A	154
SBC A, E	9B	155
SBC A, H	9C	156
SBC A, L	9D	157
SBC A, xx	DExx	222, xx
SBC HL, BC	ED42	237, 66
SBC HL, DE	ED52	237, 82
SBC HL, HL	ED62	237, 98
SBC HL, SP	ED72	237, 114
SCF	37	55
SET 0, (HL)	CBC6	203, 198
SET 0, (IX + d)	DDCBddC6	221, 203, xx, 198
SET 0, (IX + d)	FDCBddC6	253, 203, xx, 198
SET 0, A	CBC7	203, 199
SET 0, B	CBC0	203, 192
SET 0, C	CBC1	203, 193
SET 0, D	CBC2	203, 194
SET 0, E	CBC3	203, 195
SET 0, H	CBC4	203, 196
SET 0, L	CBC5	203, 197
SET 1, (HL)	CBCE	203, 206
SET 1, (IX + d)	DDCBddCE	221, 203, xx, 206
SET 1, (IX + d)	FDCBddCE	253, 203, xx, 206
SET 1, A	CBCF	203, 207
SET 1, B	CBC8	203, 200

SET 1, C	CBC9	203, 201
SET 1, D	CBCA	203, 202
SET 1, E	CBCB	203, 203
SET 1, H	CBCC	203, 204
SET 1, L	CBCE	203, 205
SET 2, (HL)	CBD6	203, 214
SET 2, (IX + d)	DDCBddD6	221, 203, xx, 214
SET 2, (IY + d)	FDCBddD6	253, 203, xx, 214
SET 2, A	CBD7	203, 215
SET 2, B	CBD0	203, 208
SET 2, C	CBD1	203, 209
SET 2, D	CBD2	203, 210
SET 2, E	CBD3	203, 211
SET 2, H	CBD4	203, 212
SET 2, L	CBD5	203, 213
SET 3, (HL)	CBDE	203, 222
SET 3, (IX + d)	DDCBddDE	221, 203, xx, 222
SET 3, (IY + d)	FDCBddDE	253, 203, xx, 222
SET 3, A	CBDF	203, 223
SET 3, B	CBD8	203, 216
SET 3, C	CBD9	203, 217
SET 3, D	CBDA	203, 218
SET 3, E	CBDB	203, 219
SET 3, H	CBDC	203, 220
SET 3, L	CBDD	203, 221
SET 4, (HL)	CBE6	203, 230
SET 4, (IX + d)	DDCBddE6	221, 203, xx, 230
SET 4, (IY + d)	FDCBddE6	253, 203, xx, 230
SET 4, A	CBE7	203, 231
SET 4, B	CBE0	203, 224
SET 4, C	CBE1	203, 225
SET 4, D	CBE2	203, 226
SET 4, E	CBE3	203, 227
SET 4, H	CBE4	203, 228
SET 4, L	CBE5	203, 229
SET 5, (HL)	CBEE	203, 238
SET 5, (IX + d)	DDCBddEE	221, 203, xx, 238

SET 5, A	CBEF	203, 239
SET 5, B	CBE8	203, 232
SET 5, C	CBE9	203, 233
SET 5, D	CBEA	203, 234
SET 5, E	CBEB	203, 235
SET 5, H	CBFC	203, 236
SET 5, L	CBED	203, 237
SET 6, (HL)	CBF6	203, 246
SET 6, (IX + d)	DDCBddF6	221, 203, xx, 246
SET 6, (IY + d)	FDCBddF6	253, 203, xx, 246
SET 6, A	CBF7	203, 247
SET 6, B	CBF0	203, 240
SET 6, C	CBF1	203, 241
SET 6, D	CBF2	203, 242
SET 6, E	CBF3	203, 243
SET 6, H	CBF4	203, 244
SET 6, L	CBF5	203, 245
SET 7, (HL)	CBFE	203, 254
SET 7, (IX + d)	DDCBddFE	221, 203, xx, 254
SET 7, (IY + d)	FDCBddFE	253, 203, xx, 254
SET 7, A	CBFF	203, 255
SET 7, B	CBF8	203, 248
SET 7, C	CBF9	203, 249
SET 7, D	CBFA	203, 250
SET 7, E	CBFB	203, 251
SET 7, H	CBFC	203, 252
SET 7, L	CBFD	203, 253
SLA (HL)	CB26	203, 38
SLA (IX + d)	DDCBdd26	221, 203, xx, 38
SLA (IY + d)	FDCBdd26	253, 203, xx, 38
SLA A	CB27	203, 39
SLA B	CB20	203, 32
SLA C	CB21	203, 33
SLA D	CB22	203, 34
SLA E	CB23	203, 35
SLA H	CB24	203, 36
SLA L	CB25	203, 37

SRA (HL)	CB2E	203, 46
SRA (IX + d)	DDCBdd2E	221, 203, xx, 46
SRA (IY + d)	FDCBdd2E	253, 203, xx, 46
SRA A	CB2F	203, 47
SRA B	CB28	203, 40
SRA C	CB29	203, 41
SRA D	CB2A	203, 42
SRA E	CB2B	203, 43
SRA H	CB2C	203, 44
SRA L	CB2D	203, 45

SRL (HL)	CB3E	203, 62
SRL (IX + d)	DDCBdd3E	221, 203, xx, 62
SRL (IY + d)	FDCBdd3E	253, 203, xx, 62
SRL A	CB3F	203, 63
SRL B	CB38	203, 56
SRL C	CB39	203, 57
SRL D	CB3A	203, 58
SRL E	CB3B	203, 59
SRL H	CB3C	203, 60
SRL L	CB3D	203, 61

SUB (HL)	96	150
SUB (IX + d)	DD96dd	221, 150, xx
SUB (IY + d)	FD96dd	253, 150, xx
SUB A	97	151
SUB B	90	144
SUB C	91	145
SUB D	92	146
SUB E	93	147
SUB H	94	148
SUB L	95	149

XOR (HL)	AE	174
XOR (IX + d)	DDAEdd	221, 174, xx
XOR (IY + d)	FDAEdd	253, 174, xx
XOR A	AF	175
XOR B	A8	168
XOR C	A9	169
XOR D	AA	170

XOR E	AB	171
XOR H	AC	172
XOR L	AD	173
XOR xx	EEx	238, xx

Gráfica Palas Athena
Associação "Palas Athena" do Brasil
Rua José Bento, 394
Fone: 279-6288 — CEP 01523
Cambuci — São Paulo