TOP SECRET

MSX1
MSX2
MSX2+
MSX turbo R

Desenvolvimento
Arquitetura Interna
BIOS
Math-Pack
Memória Mapeada
Megaram
Vídeo
VDP V9938
VDP V9958
FM-OPLL
MSX-AUDIO
MSXDOS1
MSXDOS2
Winchesters
Relógio

EDISON ANTONIO PIRES DE MORAES

MSX é marca registrada da Gradiente Eletrônica
TASWORD é marca registrada da Tasman Ltd
LX-300 é marca registrada da Epson do Brasil
MSDOS e PC são marcas registradas da IBM Corp
MSXDOS é marca registrada da Microsoft Corp
MSXDOS2 é marca registrada da ASCII Corp

NOTA DO AUTOR

Este não é um livro indicado para iniciantes em micros MSX, mas sim para programadores e usuários que já conhecem um pouco da arquitetura do MSX.

Esta obra descreve detalhes do MSX até agora desconhecidos pela maioria dos usuários brasileiros, embora o próprio autor reconheça que ainda faltam alguns detalhes importantes.

Este livro foi escrito com muito carinho e dedicação, durante três anos de exaustiva pesquisa, inclusive até no próprio micro MSX, para chegar a um resultado que o autor considera satisfatório.

Após a pesquisa e escrita, vieram a edição e diagramação, feitas pelo próprio autor, usando o processador de textos TASWORD MSX2 modificado, e a impressão, feita em uma impressora EPSON LX-300, o que consumiu mais seis meses de trabalho duro, mas também com muito carinho.

Este livro é fruto do carinho que o autor tem pelos micros MSX, que o considera um dos melhores já construídos.

Apenas para complementar, este não é um livro que prima pelo português impecável, visto não se tratar de obra de caráter literário, mas sim essencialmente técnico. A língua oficial do livro é o Portinglês, mas também pode-se encontrar um pouco de Portunhol, Portfrancês e Portjaponês. O autor espera não ser criticado pela profusão de línguas existente nesta obra. Por mais paradoxal que possa parecer, a mistura de idiomas foi feita com o intuito de facilitar a compreensão de quem vier a ler o livro.

Agradecimento

Agradeço ao meu grande
amigo RICARDO SUZUKI,
sem a ajuda do qual
a conclusão desta obra
não teria sido possível.

ÍNDICE

Capítulo 1

INTRODUÇÃO AO SISTEMA

O sistema MSX foi criado em 1.983 e anunciado oficialmente em junho desse mesmo ano pela Microsoft, detentora do padrão na época (MSX é a sigla de **M**icro**S**oft e**X**tended). O MSX foi criado com arquitetura aberta, podendo qualquer empresa fabricá-lo sem ter que pagar "royalties".

As especificações previam que todos os micros MSX seriam compatíveis em pontos estratégicos, e que todas as versões que viessem a ser criadas posteriormente manteriam a compatibilidade com o padrão original.

Atualmente, já existe a quarta versão, o MSX turbo R, e na prática a compatibilidade tem se mantido. De fato, sempre há pequenas alterações em função do desenvolvimento tecnológico ou da não utilização de certos recursos pelos programadores e pelos usuários em geral. Assim, do MSX1 para o MSX2, a expansão de memória em slots, de manuseio complicado, foi substituída por uma expansão chamada "Memory Mapper" ou "Memória Mapeada". Do MSX2 para o MSX2+, a RAM principal passou a se constituir pelos primeiros 64 Kbytes da Memória Mapeada, economizando com isso um slot, além de ter algumas funções do VDP (processador de vídeo) alteradas. Já do MSX2+ para o MSX turbo R, as mudanças foram mais radicais: eliminou-se a interface de cassete, que tornou-se totalmente obsoleta e introduziu-se uma nova CPU de 16 bits, a R800, totalmente compatível com o Z80, porém incrivelmente mais rápida que este.

Assim, apesar dessas pequenas alterações que teoricamente destruiriam a compatibilidade, na prática o MSX turbo R é compatível com todos os modelos anteriores. Veja abaixo algumas características e diferenças principais entre os quatro modelos MSX que existem atualmente.

	Junho/83 MSX1	Maio/85 MSX2	Outubro/88 MSX2+	Outubro/90 MSX turbo R
CPU	Z80 3,58MHz	Z80 3,58MHz	Z80 3,58MHz	Z80 3,58MHz R800 28MHz
RAM mínima	8 Kbytes	64 Kbytes	64 Kbytes	256 Kbytes
RAM máxima	1 Mbyte	4 Mbytes	4 Mbytes	4 Mbytes
VRAM	16 Kbytes	64/128 Kb	128 Kbytes	128 Kbytes
VDP	TMS9918	V9938	V9958	V9958
ROM stand.	32K Main	32K Main 16K SUB-ROM	32K Main 32K SUB-ROM 16K DOS1	32K Main 48K SUB-ROM 16K DOS1 48K DOS2
Interf.CAS	Standard	Standard	Standard	-
Interf.imp.	Standard	Standard	Standard	Standard

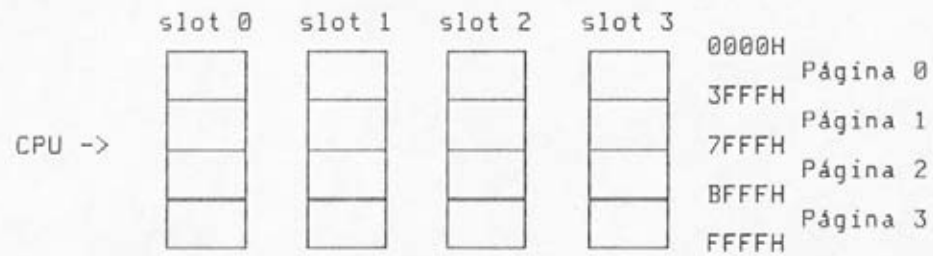
Slots ext.	1 ou 2	2	2	2
PSG	Standard	Standard	Standard	Standard
MSX-AUDIO	-	Opcional	Opcional	Opcional
FM SOUND	-	Opcional	Standard	Standard
PCM	-	-	-	Standard
Disk-drive	Opcional	Opcional	Standard 3½ DD	Standard 3½ DD
MSX-BASIC	Ver. 1.0	Ver. 2.0	Ver. 3.0	Ver. 4.0
MSX-DOS	Ver. 1.0	Ver. 1.0 2.0 opc.	Ver. 1.0 2.0 opc.	Ver. 1.0 2.0 stand.

1 - SLOTS E CARTUCHOS

A CPU Z80A que é usada nos micros MSX pode endereçar diretamente apenas 64 Kbytes de memória. Nos micros MSX, entretanto, usando a técnica de slots e páginas, o Z80 pode acessar até 1 Megabyte. Cuidado para não confundir a técnica de slots com a expansão de memória "Memory Mapper" que se utiliza de outro artifício para que cada slot possa acessar até 4 Megabytes. A introdução da nova CPU R800, de 16 bits, nos modelos MSX turbo R, não altera a técnica de slots e páginas.

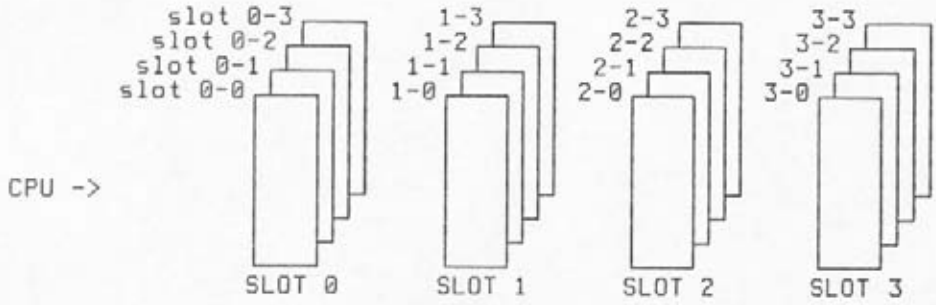
1.1 - SLOTS

Há dois tipos de slots: os slots primários e os slots secundários. Os slots primários são em número de quatro e estão conectados diretamente à CPU. Cada slot é dividido em quatro partes de 16 Kbytes, perfazendo 64 Kbytes, denominadas "páginas". Cada página ocupa o mesmo espaço de endereçamento da CPU e por isso apenas quatro páginas podem ficar ativas ao mesmo tempo, ainda que em slots diferentes.

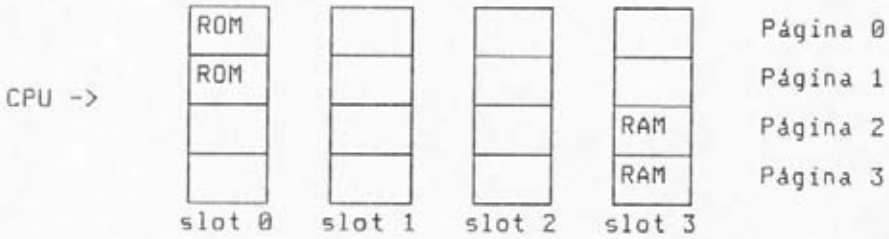


Cada slot primário pode suportar até quatro slots secundários. A escolha das páginas continua sendo possível da mesma forma que nos slots primários: apenas quatro páginas podem ficar ativas ao mesmo tempo, ainda que em slots primários e secundários diferentes.

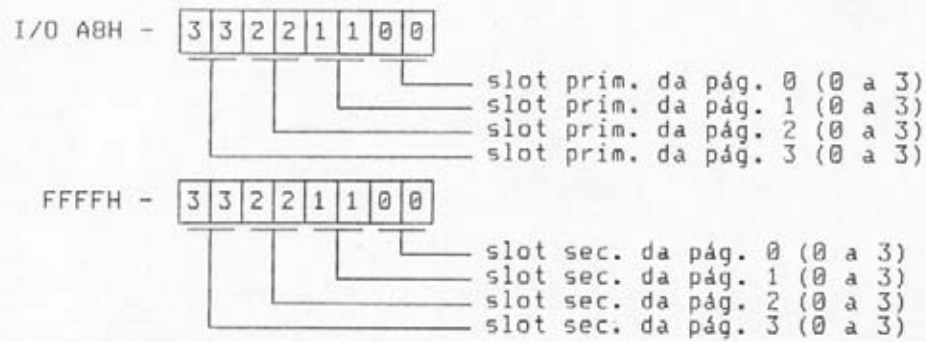
Veja na página seguinte como ficam estruturados os slots primários e secundários. Observe que podem haver até 16 slots ao todo. Dos 16 slots possíveis, 8 são reservados para a expansão do sistema e os outros 8 para o usuário.



Seleção inicial de memória pela CPU:



A seleção de slots e páginas é diferente para slots primários e secundários. Para os slots primários, ela é feita pela porta de I/O A8H e para os slots secundários é feita pelo registrador de slot secundário que nada mais é que o endereço FFFFH de memória. Não é recomendável que se troque slots e páginas diretamente e é necessário um cuidadoso planejamento para se chavear páginas e slots. Para utilizar rotinas em outras páginas é recomendável sempre usar o BIOS, que além de ser mais seguro e garantir a compatibilidade, simplifica muito a operação de slots e páginas.

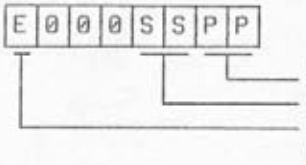


Para obter o valor correto do slot secundário no endereço FFFFH, é necessário fazer uma inversão após a leitura (instrução NOT do BASIC ou CPL em Assembly). Atente para o detalhe de que o valor só é invertido quando LIDO. Quando for ESCRITO, o valor não é invertido.

Os slots onde ficam instaladas a Main-ROM, a SUB-ROM e a RAM dependem de cada máquina. Em muitos casos, é necessário saber onde estão instaladas as memórias básicas do MSX, como no ca-

so de estar rodando um programa sob o DOS e ser necessário acessar a Main-ROM, por exemplo. Os slots onde estão armazenados as posições da Main-ROM e a SUB-ROM são especificados nas seguintes variáveis de sistema:

EXPTBL (FCC1H) - Slot da Main-ROM.
EXBRSA (FAF8H) - Slot da SUB-ROM (0 para o MSX1).



1.2 - CHAMADAS INTER-SLOTS

Quando um programa está rodando em um determinado slot e deve chamar alguma rotina em outro slot, este está fazendo uma chamada inter-slot.

Existem três casos mais comuns de chamadas inter-slot:

1- Chamada do BIOS na Main-ROM a partir do MSX-DOS;
2- Chamada do BIOS na SUB-ROM a partir do BASIC;
3- Chamada do BIOS na Main-ROM e na SUB-ROM a partir de sofware em cartucho ou RAM.

Para facilitar e assegurar a compatibilidade, existe um grupo de rotinas do BIOS denominado "chamadas inter-slot", sendo que algumas destas rotinas também estão disponíveis para o MSX-DOS, para que este possa acessar todas as rotinas do BIOS.

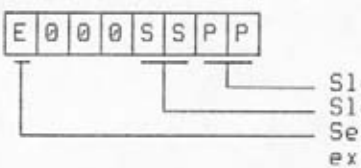
As rotinas "inter-slot" do BIOS são as seguintes:

RDSLT (000CH) - Lê um byte em qualquer slot.
WRSLT (0014H) - Escreve um byte em qualquer slot.
CALSLT (001CH) - Chama uma rotina em qualquer slot.
ENASLT (0024H) - Troca páginas e slots.
CALLF (0030H) - Chama uma rotina em qualquer slot.
RSLREG (0138H) - Lê o registrador de slot primário.
WSLREG (013BH) - Escreve no registrador de slot primário.
SUBROM (015CH) - Chama uma rotina na SUB-ROM.
EXTROM (015FH) - Chama uma rotina na SUB-ROM.

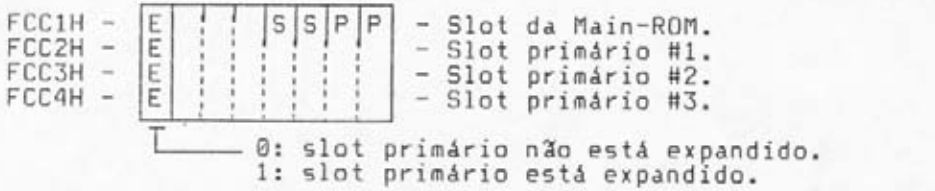
A descrição completa de cada rotina pode ser vista no capítulo 2, na seção "BIOS EM ROM".

As áreas de trabalho que contêm as variáveis de sistema relativas ao slots são as seguintes (a descrição completa das variáveis de sistema pode ser vista no capítulo 3):

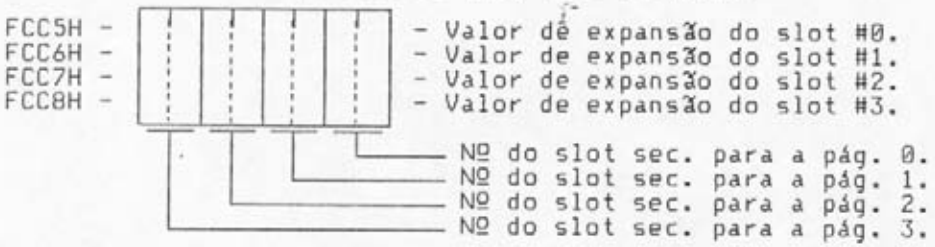
EXBRSA (FAF8H,1) - Slot da SUB-ROM.



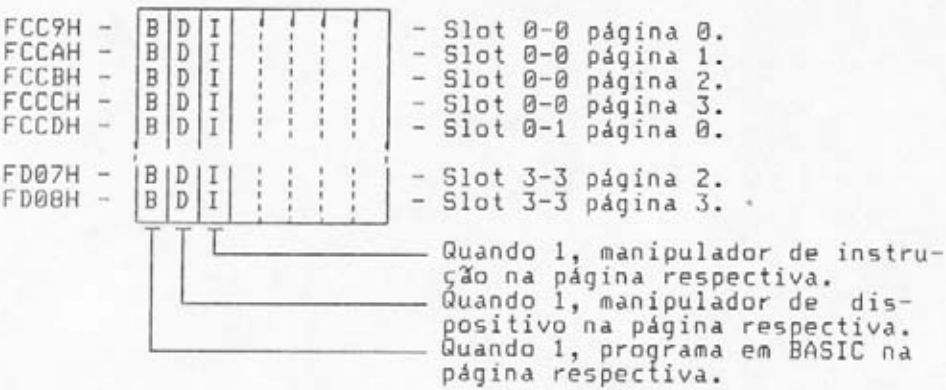
EXPTBL (FCC1H,4) - Indica se slot primário está expandido ou não.



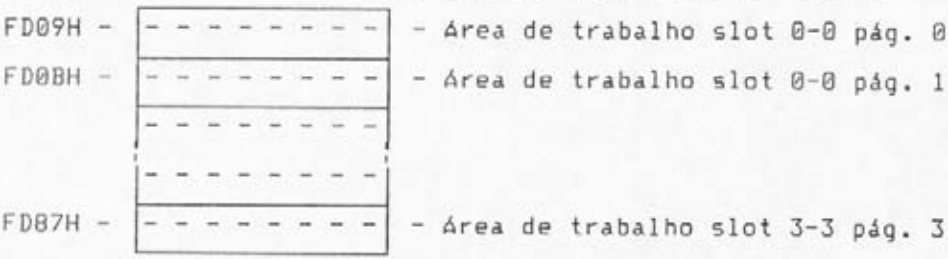
SLTTBL (FCC5H,4) - Área onde ficam registrados os valores de expansão de cada slot primário.



SLTATR (FCC9H,64) - Guarda a existência de rotinas em qualquer página e slot.



SLTWRK (FD09H,128) - Área de trabalho dos slots e páginas, reservando dois bytes para cada página.

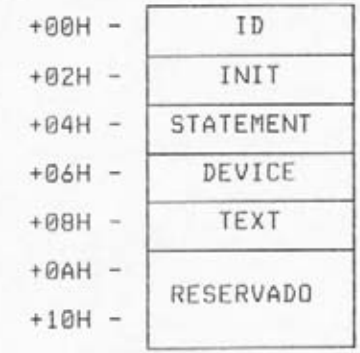


2 - DESENVOLVENDO SOFTWARE EM CARTUCHO

Normalmente, os micros MSX possuem dois slots externos onde podem ser encaixados cartuchos contendo softwares, interfa-

ces, etc. Programas em BASIC ou Assembler podem ser facilmente armazenados em cartuchos contendo uma ROM ou EPROM.

Os cartuchos devem ter obrigatoriamente os primeiros 16 bytes reservados para o header. O header pode iniciar nos endereços 4000H ou 8000H, portanto somente nas páginas 1 ou 2. Os cartuchos não podem ocupar a área de endereçamento das páginas 0 e 3. Quando o micro é resetado, as informações contidas no header do cartucho são automaticamente reconhecidas para que o MSX execute corretamente as rotinas contidas no mesmo. A composição do header do cartucho é a seguinte:



Obs.: a área reservada deve ser obrigatoriamente preenchida com bytes 00H.

ID - São dois bytes de identificação. No caso de cartuchos ROM, esses bytes devem ter o código "AB" (41H,42H) e no caso de cartuchos para a SUB-ROM, os bytes devem ser "CD" (43H,44H).

INIT - Quando é necessário inicializar a área de trabalho ou I/O, esses dois bytes devem conter o endereço da rotina de inicialização, caso contrário deve conter o valor 0000H. Depois que a rotina de inicialização foi executada, a instrução RET retorna o controle ao micro. Todos os registradores podem ser modificados, exceto o registrador SP. Programas em assembler também devem ser executados diretamente pelos bytes INIT.

STATEMENT - Quando o cartucho deve ser acessado pela instrução CALL do BASIC, esses dois bytes devem conter o endereço da rotina de expansão, caso contrário deve conter o valor 0000H.

A instrução CALL tem o seguinte formato:

CALL <nome da instrução de expansão> (argumentos)

O nome da instrução pode ter até 15 caracteres. A abreviação da instrução CALL é o caractere sublinhado "_" e pode ser usado no lugar de CALL sem problemas.

Quando o interpretador BASIC encontra um comando CALL, o nome da instrução de expansão é colocado na variável de sistema PROCNM (FD89H) e o controle transferido para a rotina cujo início é indicado pelos bytes STATEMENT. é esta rotina que deve reconhecer o nome da instrução em PROCNM. O registrador HL aponta exatamente para o primeiro caractere após a instrução expandida. Veja o exemplo abaixo:

```
CALL COMANDO (0,1,2):A=0
              ↑
              HL
```

```
PROCNM -> |C|O|M|A|N|D|O|0|
                          |___ Fim do nome da instrução
                               expandida (byte 00H).
```

Quando a rotina de expansão não reconhece o comando, ela deve manter o valor de HL, setar a flag CY (CY=1) e devolver o controle ao interpretador (instrução RET). O interpretador vai então procurar outros cartuchos de expansão de comandos, se houver mais de um, e o procedimento será o mesmo. Se ao final a instrução não for reconhecida como válida, a flag CY ficará setada e será exibida e mensagem "Syntax error" (erro de sintaxe). Veja o exemplo abaixo:

```
CALL COMANDO (0,1,2):A=0
              ↑
              HL
Flag CY=1
```

Já se o comando for reconhecido como válido, a rotina correspondente será executada e no retorno ao interpretador, a flag CY deve estar resetada (CY=0) e o registrador HL deve apontar para o primeiro sinalizador após o argumento da instrução expandida. O sinalizador pode ser o valor 00H (fim de linha) ou 3AH (dois pontos, separador de instruções). O processamento continuará normalmente. Veja o exemplo abaixo:

```
CALL COMANDO (0,1,2):A=0
                     ↑
                     HL
Flag CY=0
```

DEVICE - Esses dois bytes podem apontar para uma rotina de expansão de dispositivos, no caso do cartucho conter um dispositivo de I/O; caso contrário esses bytes devem ser 0000H. A rotina para o dispositivo de expansão deve estar entre 4000H e 7FFFH. Um cartucho pode ter até quatro dispositivos, cujo nome pode ter até 15 caracteres.

Quando o interpretador encontra um dispositivo indefinido, ele armazena o nome em PROCNM (FD89H), coloca o valor FFH no registrador A e passa o controle para o cartucho que tenha uma expansão de dispositivo.

Para criar uma rotina de expansão de dispositivos, identifique o descritor de arquivo em PROCNM primeiro, e se não for o dispositivo correto, retorne ao interpretador com a flag CY setada em 1. Veja o exemplo abaixo:

```
OPEN "XYZ:" ...  ->  nome do dispositivo
Registrador A=FFH
Flag CY=1
PROCNM -> |X|Y|Z|0|
                |___ Fim do descritor de arquivo (00H)
```

Já se o descritor de dispositivo for reconhecido, a rotina deve ser processada e o número de identificação do dispositivo (device ID), que varia de 0 a 3, deve ser colocado no re-

gistrador A; depois sete a flag CY em 0 e retorne ao interpretador.

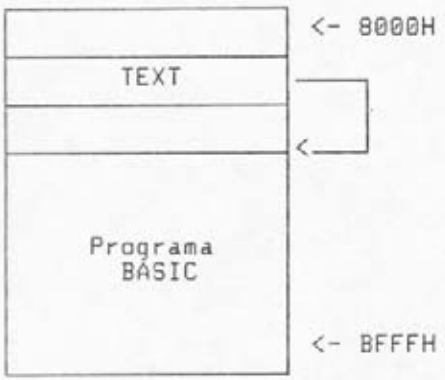
O interpretador procura cartucho após cartucho, e se ao final o nome do dispositivo não for reconhecido (ou seja, a flag CY sempre for 1), a mensagem de erro "Bad file name" (nome de arquivo errado) será mostrada.

Quando a operação de I/O atual é processada, o interpretador coloca o nome do dispositivo (device ID; 0 a 3) na variável de sistema DEVICE (FD99H) e seta o dispositivo requerido em A (veja a tabela abaixo) para depois chamar a rotina de expansão de dispositivo.

Reg.A	Dispositivo	Reg.A	Dispositivo
0	OPEN	10	Função LOC
2	CLOSE	12	Função LOF
4	Acesso aleatório	14	Função EOF
6	Saída seqüencial	16	Função FPOS
8	Entrada seqüencial	18	Caractere "backup"

TEXT - Esses dois bytes apontam para um programa BASIC gravado em cartucho, autoexecutável quando o micro for resetado. Se não houver programa BASIC, esses dois bytes devem conter 0000H. O tamanho do programa não pode ultrapassar 16 Kbytes (8000H a BFFFH).

O interpretador examina o conteúdo de TEXT, e se esse conter um endereço, inicia a execução do programa BASIC contido no endereço indicado. O primeiro byte apontado por TEXT deve ser 00H, que indica o início do texto BASIC.



Capítulo 2

A MEMÓRIA ROM

A memória ROM é vital para o funcionamento do micro. No caso do MSX, ela incorpora a rotina de inicialização, o BIOS, a tabela inicial de caracteres, o MSXDOS (DOS Kernel), etc.

Além disso, existem alguns bytes no inicio da ROM que contêm algumas informações importantes que podem ser úteis ao programador. Esses bytes são:

0004H/0005H - Endereço do conjunto de caracteres na ROM.
0006H - Porta de leitura de dados do VDP.
0007H - Porta de escrita de dados no VDP.

002BH - 

Tipo do gerador de caracteres:
0 = japonês
1 = internacional
2 = coreano
Formato da data:
0 = ano/mês/dia
1 = mês/dia/ano
2 = dia/mês/ano
Freqüência de interrupção:
0 = 60 Hz
1 = 50 Hz

002CH - 

Tipo de teclado:
0 = japonês
1 = internacional
2 = francês
3 = reino unido
4 = alemão
Versão do BASIC:
0 = japonês
1 = internacional

002DH - Versão do hardware:
00H = MSX1
01H = MSX2
02H = MSX2+
03H = MSX turbo R

002EH - 

MSX-MIDI:
0 = sem MSX-MIDI
1 = MSX-MIDI inclusa (turbo R)

1 - BIOS EM ROM

Praticamente todo programa, seja em assembler ou linguagem de alto nível, incluindo o próprio Interpretador BASIC residente no MSX, requer um conjunto de funções primárias para poder operar. Essas funções incluem acionadores de tela, impresso-

ras, drives e outras funções relacionadas ao hardware. No MSX essas funções primárias são realizadas pelas rotinas do BIOS, que significa "Basic Input/Output System", ou Sistema Básico de Entrada e Saída.

Esse capitulo fornece a descrição de 137 rotinas do BIOS disponíveis ao usuário, se o micro for um MSX turbo R. Para versões anteriores, o número de rotinas disponíveis diminui, mas isto estará descrito detalhadamente.

Existem dois tipos de rotinas do BIOS: as que estão na Main-ROM e as que estão na SUB-ROM. Para o MSX1 não existe SUB-ROM; para o MSX2 há 16K de SUB-ROM (Página 0); para o MSX2+ há 32K de SUB-ROM (Páginas 0 e 1) e para o MSX turbo R há 48K de SUB-ROM (Páginas 0, 1 e 2). As rotinas da Main-ROM e da SUB-ROM usam diferentes seqüências de chamada. Para a Main-ROM pode ser usada uma instrução CALL ou RST. As chamadas para a SUB-ROM serão descritas posteriormente.

As rotinas estão listadas conforme a seguinte notação:

```
LABEL  (Endereço de chamada)
  Função:  descreve a função da rotina
  Entrada: descreve os parâmetros para chamada
  Saída:   descreve os parâmetros de retorno da rotina
  Registradores: descreve os registradores da UCP modificados
                 pela rotina.
```

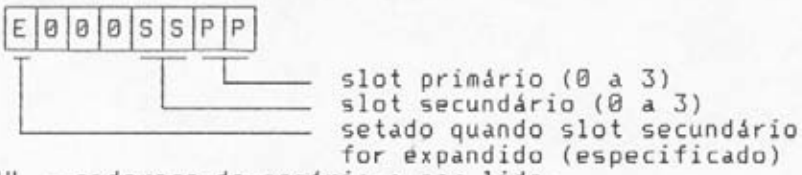
1.1 - ROTINAS RST

Das rotinas RST listadas, de *RST 00H* até *RST 28H* são rotinas usadas pelo interpretador BASIC. A *RST 30H* é usada para chamadas inter-slot e a *RST 38H* é usada para interrupções de hardware. Porém, deve ser ressalvado que nem todas as rotinas desse grupo podem ser chamadas por instruções RST, devendo, nesse caso usar instruções CALL.

```
CHKRAM (0000H)
  Função:  Testa a RAM na partida e inicializa as variáveis de
           sistema. Uma chamada a esta rotina provocará um reset
           por software.
  Entrada: Nenhuma.
  Saída:   Nenhuma.
  Registradores: Todos.

SYNCHR (0008H)
  Função:  Testa se o caractere apontado por (HL) é o especifi-
           cado. Se não for, gera "Syntax error" (Erro de sinta-
          xe), caso contrário chama CHRGTR (0010H).
  Entrada: Coloque o caractere a ser testado em (HL) e o caracte-
           re para comparação após a instrução RST (parâmetro em
          linha). Veja o exemplo abaixo:
                    LD   HL,CARACT
                    RST  008H
                    DEFB 'A'
                    :
                    DEFB 'B'
  Saída:   HL é incrementado em um e A recebe (HL). Quando o ca-
           ractere testado for numérico, a flag CY é setada; o
          fim de declaração (00H ou 3AH) seta a flag Z.
```

Registradores: AF, HL.

RDSLT (000CH)
Função: Lê um byte em qualquer slot. As interrupções são desabilitadas durante a leitura.
Entrada: A contém o indicador se slot.



HL - endereço de memória a ser lido.
Saída: A contém o valor do byte lido.
Registradores: AF, BC, DE.

CHRGTR (0010H)
Função: Lê um caracter ou um token do texto BASIC.
Entrada: (HL) aponta para o caracter atual do texto.
Saída: HL é incrementado em um e A recebe (HL). Quando o caractere for numérico, a flag CY é setada; o fim de declaração (00H ou 3AH) seta a flag Z.
Registradores: AF, HL.

WRSLT (0014H)
Função: Escreve um byte na RAM em qualquer slot. As interrupções ficam desabilitadas durante a escrita.
Entrada: A - indicador de slot (igual a RDSLT - 000CH).
HL - endereço para a escrita.
E - byte a ser escrito.
Saída: Nenhuma.
Registradores: AF, BC, D.

OUTDO (0018H)
Função: Saída para o dispositivo atual.
Entrada: A - caractere a sair.
Se PRTFLG (F416H) for diferente de 0, o caractere é enviado à impressora.
Se PTRFIL (F864H) for diferente de 0, o caractere é enviado ao arquivo especificado por PTRFIL.
Saída: Nenhuma.
Registradores: Nenhum.

CALSLT (001CH)
Função: Chama rotina em qualquer slot (chamada inter-slot).
Entrada: Especificar o byte ID de slot (igual a RDSLT - 000CH) nos 8 bits mais altos de IY. IX deve conter o endereço a ser chamado.
Saída: Depende da rotina chamada.
Registradores: Depende da rotina chamada.

DCOMPR (0020H)
Função: Compara HL com DE.
Entrada: HL, DE.
Saída: Seta flag Z se HL=DE; seta flag CY se HL<DE.
Registradores: AF.

ENASLT (0024H)
  Função: Habilita uma página de qualquer slot. As interrupções são desativadas durante a habilitação.
  Entrada: A - Indicador de slot (igual a RDSLT - 000CH).
           HL - Qualquer endereço da página a ser habilitada.
  Saída: Nenhuma.
  Registradores: Todos.

GETYPR (002BH)
  Função: Obtém o tipo do operando em DAC ou indicado por VALTYP.
  Entrada: Nenhuma.
  Saída: Inteiro - A=FFH; flags M, NZ e C.
         String - A=00H; flags P, Z e C.
         Simples precisão - A=01H; flags P, NZ e C.
         Dupla precisão - A=05H; flags P, NZ e NC.
  Registradores: AF.

CALLF (0030H)
  Função: Chama rotina em qualquer slot. Ela se diferencia de CALSLT por usar parâmetros em linha, ao invés de carregar diretamente os registradores, a fim de caber dentro dos HOOKS. A seqüência de chamada é a seguinte:

```
RST  030H  ;chama CALLF
DEFB n     ;n é o ID de slot (igual a RDSLT-000CH)
DEFW nn    ;nn é o endereço a ser chamado.
```

  Entrada: Pelo método já descrito.
  Saída: Depende da rotina chamada.
  Registradores: Depende da rotina chamada.

KEYINT (0038H)
  Função: Executa rotina de interrupção e varredura do teclado.
  Entrada: Nenhuma.
  Saída: Nenhuma.
  Registradores: Nenhum.

1.2 - ROTINAS PARA INICIALIZAÇÃO DE I/O

INITIO (003BH)
  Função: Inicializar o PSG e a porta de status Centronics.
  Entrada: Nenhuma.
  Saída: Nenhuma.
  Registradores: Todos.

INIFNK (003EH)
  Função: Inicializa o conteúdo das teclas de função.
  Entrada: Nenhuma.
  Saída: Nenhuma.
  Registradores: Todos.

1.3 - ROTINAS DE ACESSO AO VDP

DISSCR (0041H)
  Função: Desabilita a saída de vídeo.
  Entrada: Nenhuma.
  Saída: Nenhuma.
  Registradores: AF, BC.

ENASCR (0044H)
  Função: Habilita a saída de vídeo.
  Entrada: Nenhuma.
  Saída: Nenhuma.
  Registradores: AF, BC.

WRTVDP (0047H)
  Função: Escreve dados nos registradores do VDP.
  Entrada: B - byte a ser escrito.
  C - registrador que receberá o dado. Pode variar de 0 a 7 para MSX1, de 0 a 23 e 32 a 46 para MSX2 e de 0 a 23 / 25 a 27 / 32 a 46 para o MSX2+ ou superior.
  Saída: Nenhuma.
  Registradores: AF, BC.

RDVRM (004AH)
  Função: Lê um byte da VRAM. Essa rotina acessa somente os 14 bits mais baixos do bus de endereços da VRAM (16 Kbytes para o VDP TMS 9918 do MSX1). Para acessar toda a VRAM, use a rotina NRDVRM (0174H).
  Entrada: HL - endereço da VRAM a ser lido.
  Saída: A - contém o byte lido.
  Registradores: AF.

WRTVRM (004DH)
  Função: Escreve um byte na VRAM. Essa rotina acessa somente os 14 bits mais baixos do bus de endereços da VRAM (16 Kbytes para o VDP TMS9918 do MSX1). Para acessar toda a VRAM, use a rotina NWRVRM (0177H).
  Entrada: HL - endereço da VRAM a ser escrito.
  A - byte a ser escrito.
  Saída: Nenhuma.
  Registradores: AF.

SETRD (0050H)
  Função: Prepara a VRAM para leitura seqüencial usando a função de auto-incremento de endereço do VDP. é um meio de leitura mais rápido do que usando um loop com a rotina RDVRM. Essa rotina acessa somente os 14 bits mais baixos do bus de endereços da VRAM (16 Kbytes para o VDP TMS9918 do MSX1). Para acessar toda a VRAM, use a rotina NSETRD (016EH).
  Entrada: HL - endereço da VRAM para início da leitura.
  Saída: Nenhuma.
  Registradores: AF.

SETWRT (0053H)
  Função: Prepara a VRAM para escrita seqüencial usando a função de auto-incremento de endereço do VDP. As características são as mesmas de SETRD. Para acessar toda a VRAM, use a rotina NSTWRT (0171H).
  Entrada: HL - endereço da VRAM para início da escrita.
  Saída: Nenhuma.
  Registradores: AF.

FILVRM (0056H)
  Função: Preenche um bloco da VRAM com um único byte de dados. Essa rotina acessa somente os 14 bits mais baixos do bus de endereços da VRAM (16 Kbytes para o VDP TMS

9918 do MSX1). Para acessar toda a VRAM, use a rotina BIGFIL (016BH).
Entrada: HL - endereço da VRAM para início da escrita.
BC - quantidade de bytes (comprimento).
A - byte a ser escrito.
Saída: Nenhuma.
Registradores: AF, BC.

LDIRMV (0059H)
Função: Transfere um bloco de memória da VRAM para a RAM.
Entrada: HL - endereço fonte na VRAM.
DE - endereço de destino na RAM
BC - tamanho do bloco (comprimento).
Obs.: todos os bits de endereço são válidos.
Saída: Nenhuma.
Registradores: Todos.

LDIRVM (005CH)
Função: Transfere um bloco de memória da RAM para a VRAM.
Entrada: HL - endereço fonte na RAM.
DE - endereço de destino na VRAM.
BC - tamanho do bloco (comprimento).
Obs.: todos os bits de endereço são válidos.
Saída: Nenhuma.
Registradores: Todos.

CHGMOD (005FH)
Função: Troca os modos de tela. No caso de micros MSX2 ou superior, a paleta de cores não é inicializada. Para inicializá-la, use a rotina CHGMDP (01B5H/SUBROM).
Entrada: A - modo screen (0 a 3 para MSX1, 0 a 8 para MSX2 e 0 a 12 para MSX2+ e MSX turbo R).
Saída: Nenhuma.
Registradores: Todos.

CHGCLR (0062H)
Função: Troca as cores da tela. No modo texto 40 ou 80 colunas, a cor da borda é sempre igual à cor de fundo.
Entrada: FORCLR (F3E9H) - cor do primeiro plano.
BAKCLR (F3EAH) - cor de fundo.
BDRCLR (F3EBH) - cor da borda.
Saída: Nenhuma.
Registradores: Todos.

NMI (0066H)
Função: Executa a rotina NMI (Non-Maskable Interrupt - Interrupção não mascarável).
Entrada: Nenhuma.
Saída: Nenhuma.
Registradores: Nenhum.

CLRSPR (0069H)
Função: Inicializa todos os sprites. A tabela de padrões dos sprites é limpa (preenchida com zeros), os números dos sprites são inicializados com a série 0 ≈ 31 e a cor dos sprites é igualada à cor de fundo. A localização vertical dos sprites é colocada em 209 (screens 0 a 3) ou 217 (screens 4 a 8 e 10 a 12).
Entrada: SCRMOD (FCAFH) deve conter o modo screen.
Saída: Nenhuma.

Registradores: Todos.

INITXT (006CH)
Função: Inicaliza a tela no modo texto 1 (40 x 24). Nesta rotina, a paleta de cores não é inicializada. Para inicializá-la, chame INIPLT (0141H/SUBROM).
Entrada: TXTNAM (F3B3H) - endereço da tabela de nomes dos caracteres.
TXTCGP (F3B7H) - endereço da tabela geradora de padrões dos caracteres.
LINL40 (F3AEH) - largura das linhas em caracteres.
Saída: Nenhuma.
Registradores: Todos.

INIT32 (006FH)
Função: Inicializa a tela no modo gráfico 1 (32 x 24). Nesta rotina, a paleta de cores não é inicializada. Para inicalizá-la, chame INIPLT (0141H/SUBROM).
Entrada: T32NAM (F3BDH) - endereço da tabela de nomes dos caracteres.
T32COL (F3BFH) - endereço da tabela de cores dos caracteres.
T32CGP (F3C1H) - endereço da tabela de padrões dos caracteres.
T32ATR (F3C3H) - endereço da tabela de atributos dos sprites.
T32PAT (F3C5H) - endereço da tabela de padrões dos sprites.
Saída: Nenhuma.
Registradores: Todos.

INIGRP (0072H)
Função: Inicializa a tela no modo gráfico de alta resolução (screen 2). Nessa rotina, a paleta de cores não é inicializada. Para inicializá-la, chame INIPLT (0141H/SUBROM).
Entrada: GRPNAM (F3C7H) - endereço da tabela de nomes dos caracteres.
GRPCOL (F3C9H) - endereço da tabela de cores dos caracteres.
GRPCGP (F3CBH) - endereço da tabela de padrões dos caracteres.
GRPATR (F3CDH) - endereço da tabela de atributos dos sprites.
GRPPAT (F3CFH) - endereço da tabela de padrões dos sprites.
Saída: Nenhuma.
Registradores: Todos.

INIMLT (0075H)
Função: Inicializa a tela no modo multicor (screen 3). Nessa rotina, a paleta de cores não é inicializada. Para inicializá-la, chame INIPLT (0141H/SUBROM).
Entrada: MLTNAM (F3D1H) - endereço da tabela de nomes dos caracteres.
MLTCOL (F3D3H) - endereço da tabela de cores dos caracteres.
MLTCGP (F3D5H) - endereço da tabela de padrões dos caracteres.

MLTATR (F3D7H) - endereço da tabela de atributos dos sprites.
MLTPAT (F3D9H) - endereço da tabela de padrões dos sprites.

Saída: Nenhuma.
Registradores: Todos.

SETTXT (0078H)
Função: Coloca apenas o VDP no modo texto 1 (40 x 24).
Entrada: Igual a INITXT (006CH).
Saída: Nenhuma.
Registradores: Todos.

SET32 (007BH)
Função: Coloca apenas o VDP no modo gráfico 1 (32 x 24).
Entrada: Igual a INIT32 (006FH).
Saída: Nenhuma.
Registradores: Todos.

SETGRP (007EH)
Função: Coloca apenas o VDP no modo gráfico 2 (screen 2).
Entrada: Igual a INIGRP (0072H).
Saída: Nenhuma.
Registradores: Todos.

SETMLT (0081H)
Função: Coloca apenas o VDP no modo multicor (screen 3).
Entrada: Igual a INIMLT (0075H).
Saída: Nenhuma.
Registradores: Todos.

CALPAT (0084H)
Função: Retorna o endereço da tabela geradora do padrão de um sprite.
Entrada: A - número do sprite.
Saída: HL - endereço na VRAM.
Registradores: AF, DE, HL.

CALATR (0087H)
Função: Retorna o endereço da tabela de atributos de um sprite.
Entrada: A - número do sprite.
Saída: HL - endereço na VRAM.
Registradores: AF, DE, HL.

GSPSIZ (008AH)
Função: Retorna o tamanho atual dos sprites.
Entrada: Nenhuma.
Saída: A - tamanho do sprite em bytes. A flag CY é setada se o tamanho for 16 x 16 e resetada caso contrário.
Registradores: AF.

GRPPRT (008DH)
Função: Apresenta um caractere numa tela gráfica.
Entrada: A - código ASCII do caractere. Quando a screen for de 5 a 8 ou de 10 a 12, coloque o código de operação lógica em LOGOPR (FB02H).
Saída: Nenhuma.
Registradores: Nenhum.

1.4 - ROTINAS DE ACESSO AO PSG

GICINI (0090H)
  Função: Inicializa o PSG para o comando PLAY do BASIC. O volume das três vozes é colocado em 0 e o registrador 7 inicializado com B8H, ativando os geradores de som e desativando o gerador de ruído branco.
  Entrada: Nenhuma.
  Saída: Nenhuma.
  Registradores: Todos.

WRTPSG (0093H)
  Função: Escreve dados nos registradores do PSG.
  Entrada: A - número do registrador para escrita.
  E - byte a ser escrito.
  Saída: Nenhuma.
  Registradores: Nenhum.

RDPSG (0096H)
  Função: Lê o conteúdo dos registradores do PSG.
  Entrada: A - número do registrador do PSG a ser lido.
  Saída: A - valor lido.
  Registradores: Nenhum.

STRTMS (0099H)
  Função: Testa se o comando PLAY está sendo executado. Se não estiver, inicia a execução, desempilhando as filas musicais.
  Entrada: Nenhuma.
  Saída: Nenhuma.
  Registradores: Todos.

1.5 - ROTINAS DE ACESSO AO TECLADO, TELA E IMPRESSORA

CHSNS (009CH)
  Função: Verifica o buffer do teclado.
  Entrada: Nenhuma.
  Saída: A flag Z é setada se o buffer estiver vazio, caso contrário é resetada.
  Registradores: AF.

CHGET (009FH)
  Função: Entrada de um caractere pelo teclado, com espera.
  Entrada: Nenhuma.
  Saída: A - código ASCII do caractere.
  Registradores: AF.

CHPUT (00A2H)
  Função: Apresenta um caractere na tela.
  Entrada: A - código ASCII do caractere a ser apresentado.
  Saída: Nenhuma.
  Registradores: Nenhum.

LPTOUT (00A5H)
  Função: Envia um caractere para a impressora.
  Entrada: A - código ASCII do caractere a ser enviado.
  Saída: A flag CY é setada se a operação falhar.
  Registradores: F.

LPTSTT (00A8H)
  Função: Retorna o status da impressora.
  Entrada: Nenhuma.
  Saída: Quando A=255 e a flag Z estiver resetada, a impressora está pronta. Quando A=0 e a flag Z estiver setada, a impressora não está pronta para receber dados.
  Registradores: AF.

CNVCHR (00ABH)
  Função: Converte caractere com cabeçalho gráfico.
  Entrada: A - código ASCII do caractere.
  Saída: A flag CY é resetada se não houver cabeçalho gráfico; as flags CY e Z são setadas e o código convertido colocado em A; se a flag CY é setada e a flag Z resetada, o código não convertido é colocado em A.
  Registradores: AF.

PINLIN (00AEH)
  Função: Coleta uma linha de texto do console e a armazena em um buffer especificado até que a tecla RETURN ou CTRL/STOP seja pressionada.
  Entrada: Nenhuma.
  Saída: HL - endereço de início do buffer menos 1. Se a flag CY estiver setada, foi pressionada CTRL/STOP.
  Registradores: Todos.

INLIN (00B1H)
  Função: Mesma de PINLIN, exceto que AUTFLG (F6AAH) é setada.
  Entrada: Nenhuma.
  Saída: Mesma de PINLIN.
  Registradores: Todos.

QINLIN (00B4H)
  Função: Executa INLIN apresentando "?" e um espaço.
  Entrada: Nenhuma.
  Saída: Mesma de PINLIN.
  Registradores: Todos.

BREAKX (00B7H)
  Função: Verifica diretamente as teclas CTRL/STOP. Nessa rotina, as interrupções são desabilitadas.
  Entrada: Nenhuma.
  Saída: A flag CY é setada se CTRL/STOP estiverem pressionadas.
  Registradores: AF.

ISCNTC (00BAH)
  Função: Verifica as teclas CTRL/STOP ou STOP. É usada principalmente pelo interpretador BASIC. Se CTRL/STOP estiverem pressionadas, o controle é devolvido ao interpretador; se STOP for pressionada, paraliza a execução de um programa, até CTRL/STOP ou STOP serem pressiomadas novamente.
  Entrada: Nenhuma.
  Saída: Nenhuma.
  Registradores: AF.

CKCNTC (00BDH)
  Função: Mesma de ISCNTC, exceto que o programa BASIC não poderá ser continuado pela instrução CONT.

```
  Entrada: Nenhuma.
  Saída:   Nenhuma.
  Registradores: AF.

BEEP   (00C0H)
  Função:  Gera um beep.
  Entrada: Nenhuma.
  Saída:   Nenhuma.
  Registradores: Todos.

CLS    (00C3H)
  Função:  Limpa a tela.
  Entrada: A flag Z deve estar setada.
  Saída:   Nenhuma.
  Registradores: AF, BC, DE.

POSIT  (00C6H)
  Função:  Move o cursor nas telas de texto.
  Entrada: H - coordenada X (horizontal) do cursor.
           L - coordenada Y (vertical) do cursor.
  Saída:   Nenhuma.
  Registradores: AF.

FNKSB  (00C9H)
  Função:  Testa se o display das teclas de função está ligado a-
           través de FNKFLG (FBCEH). Se estiver, desliga e se não
           estiver, liga.
  Entrada: FNKFLG (FBCEH).
  Saída:   Nenhuma.
  Registradores: Todos.

ERAFNK (00CCH)
  Função:  Desliga o display das teclas de função.
  Entrada: Nenhuma.
  Saída:   Nenhuma.
  Registradores: Todos.

DSPFNK (00CFH)
  Função:  Liga o display das teclas de função.
  Entrada: Nenhuma.
  Saída:   Nenhuma.
  Registradores: Todos.

TOTEXT (00D2H)
  Função:  Força a tela para o modo texto (screen 0 ou 1).
  Entrada: Nenhuma.
  Saída:   Nenhuma.
  Registradores: Todos.
```

1.6 - ROTINAS DE ACESSO I/O PARA GAMES

```
GTSTCK (00D5H)
  Função:  Retorna o status do joystick ou teclas do cursor.
  Entrada: A - 0 = teclas do cursor.
               1 = joystick no port 1.
               2 = joystick no port 2.
  Saída:   A - direção do joystick ou teclas de função, con-
               forme a ilustração na página seguinte.
```

Registradores: Todos.

GTTRIG (00D8H)
Função: Retorna o estado dos botões do joystick ou da barra de espaço.
Entrada: A - 0 = barra de espaço.
1 = joystick no port 1, botão A
2 = joystick no port 2, botão A
3 = joystick no port 1, botão B
4 = joystick no port 2, botão B
Saída: A - 0 se o botão testado não estiver pressionado, e 255 se o botão testado estiver pressionado.
Registradores: AF, BC.

GTPAD (00DBH)
Função: Retorna o status do touch-pad (digitalizador) ligado a um dos conectores de joystick.
Entrada: A - código de função (0 a 3 para porta A e 4 a 7 para porta B):
0 ou 4 - retorna o status de atividade.
1 ou 5 - retorna coordenada "X".
2 ou 6 - retorna coordenada "Y".
3 ou 7 - retorna o status da tecla.
Saída: A - status ou valor. Para coordenada X ou Y, varia de 0 a 255; para status de atividade, devolve 255 se o touch-pad estiver sendo tocado e 0 caso contrário; para status de tecla, devolve 255 se esta estiver sendo pressionada e 0 caso contrário.
Registradores: Todos.
Obs.: Esta rotina foi modificada nos modelos MSX turbo R.

GTPDL (00DEH)
Função: Retorna o status do paddle ligado a um dos conectores de joystick.
Entrada: A - identificação do paddle (1 a 12).
1,3,5,7,9,11 - paddles ligados no port 1.
2,4,6,8,10,12 - paddles ligados no port 2.
Saída: A - valor lido (0 a 255).
Registradores: Todos.
Obs.: Esta rotina foi modificada nos modelos MSX turbo R.

1.7 - MISCELANEA

LFTQ (00F6H)
Função: Retorna o número de bytes livres em uma fila musical do PSG.
Entrada: A - número da fila (0, 1, 2).
Saída: HL - espaço livre deixado na fila.
Registradores: AF, BC, HL.

PUTQ (00F9H)
Função: Coloca um byte em uma das três filas musicais do PSG.

```
  Entrada: A - número da fila (0, 1, 2).
           E - byte de dados.
  Saída:   Flag Z setada se a fila estiver cheia.
  Registradores: AF, BC, HL.

CHGCAP (0132H)
  Função:  Altera o estado do LED de CAPS LOCK.
  Entrada: A - 0=apaga o LED; outro valor, acende o LED.
  Saída:   Nenhuma.
  Registradores: AF.

CHGSND (0135H)
  Função:  Altera a saída de som do "click" das teclas.
  Entrada: A - 0=desliga o "click"; outro valor, liga o "click".
  Saída:   Nenhuma.
  Registradores: AF.

RSLREG (0138H)
  Função:  Lê o conteúdo do registrador de slot primário.
  Entrada: Nenhuma.
  Saída:   A - valor lido.
  Registradores: A.

WSLREG (013BH)
  Função:  Escreve um valor no registrador de slot primário.
  Entrada: A - valor a ser escrito.
  Saída:   Nenhuma.
  Registradores: Nenhum.

RDVDP  (013EH)
  Função:  Lê o registrador de status do VDP.
  Entrada: Nenhuma.
  Saída:   A - valor lido.
  Registradores: A.

SNSMAT (0141H)
  Função:  Lê uma linha da matriz do teclado.
  Entrada: A - número da linha do teclado a ser lida.
  Saída:   A - colunas lidas da linha especificada. O bit cor-
               respondente a uma tecla pressionada é 0.
  Registradores: AF, C.

ISFLIO (014AH)
  Função:  Testa quando um dispositivo está ativo.
  Entrada: Nenhuma.
  Saída:   A - 0 se o dispositivo estiver ativo; outro valor se o
               dispositivo estiver inativo.
  Registradores: AF.

OUTDLP (014DH)
  Função:  Saída formatada para a impressora. Difere de LPTOUT
           (00A5H) nos seguintes pontos: se o caractere for um
           código TAB (09H), serão enviados espaços até atingir
           um múltiplo de 8; para impressoras não MSX, caracteres
           gráficos são transformados em caracteres de 1 byte; se
           houver falha, ocorre um erro de I/O.
  Entrada: A - byte a ser enviado para a impressora.
  Saída:   Nenhuma.
  Registradores: F.
```

```
GETVCP (0150H)
  Função: Retorna o endereço do byte 2 no buffer de voz especi-
          ficado (PSG).
  Entrada: A - número da voz (0, 1, 2).
  Saída:   HL - endereço no buffer de voz.
  Registradores: AF, HL.

GETVC2 (0153H)
  Função: Retorna o endereço de qualquer byte no buffer de voz
          especificado pelo número da voz em VOICEN (FB38H).
  Entrada: L - número do byte do bloco (0 a 36).
  Saída:   HL - endereço no buffer de voz.
  Registradores: AF, HL.

KILBUF (0156H)
  Função: Limpa o buffer do teclado.
  Entrada: Nenhuma.
  Saída:   Nenhuma.
  Registradores: HL.

CALBAS (0159H)
  Função: Executa uma chamada inter-slot para qualquer rotina do
          interpretador BASIC.
  Entrada: IX - endereço da rotina a ser chamada.
  Saída:   Depende da rotina chamada.
  Registradores: Depende da rotina chamada.
```

1.8 - ROTINAS PARA ACESSO AO DRIVE

```
PHYDIO (0144H)
  Função: Ler ou gravar um ou mais setores no drive especificado.
  Entrada: HL - endereço da RAM a partir do qual serão colocados
               os setores a ler ou retirados os setores a gravar.
           DE - número do primeiro setor a ser lido ou gravado.
           B -  número de setores a ler ou gravar.
           C -  parâmetro de formatação do disquete:
                0F8H - 80 trilhas, face simples;
                0F9H - 80 trilhas, face dupla;
                0FCH - 40 trilhas, face simples;
                0FDH - 40 trilhas, face dupla.
           A -  número do drive (0=A, 1=B, etc).
           Flag CY - resetada para fazer leitura,
                     setada para fazer gravação.
  Saída:   Flag CY - se estiver setada, houve algum tipo de
                     erro (leitura ou gravação).
  Registradores: Todos.

FORMAT (0147H)
  Função: Formatar um disquete. Ao ser chamada, serão apresenta-
          das uma série de perguntas que deverão ser respondidas
          para iniciar a formatação. Infelizmante, não há um pa-
          drão fixo, e as perguntas são diferentes para cada in-
          terface.
  Entrada: Nenhuma.
  Saída:   Nenhuma.
  Registradores: Todos.
```

1.9 - ENTRADAS ADICIONADAS PARA O MSX2 E MSX2+

SUBROM (015CH)
  Função: Executa uma chamada inter-slot para a SUB-ROM.
  Entrada: IX - endereço da rotina a ser chamada (ao mesmo tempo, salva IX na pilha).
  Saída: Depende da rotina chamada.
  Registradores: O registrador de fundo e IY são reservados.

EXTROM (015FH)
  Função: Executa uma chamada inter-slot para a SUB-ROM.
  Entrada: IX - endereço da rotina a ser chamada.
  Saída: Depende da rotina chamada.
  Registradores: O registrador de fundo e IY são reservados.

EOL (0168H)
  Função: Apaga até o fim da linha.
  Entrada: H - coordenada X (horizontal) do cursor.
  L - coordenada Y (vertical) do cursor.
  Saída: Nenhuma.
  Registradores: Todos.

BIGFIL (016BH)
  Função: Mesma de FILVRM (0056H), com as seguintes diferenças: Na FILVRM, são testadas as screens 0 a 3, e nesse caso, o VDP é acionado para acessar 16 Kbytes apenas, para compatibilidade com o MSX1. Na BIGFIL, o modo não é testado e as ações são levadas para fora pelos parâmetros dados.
  Entrada: HL - endereço na VRAM para início da escrita.
  BC - comprimento (número de bytes a escrever).
  A - dado a ser escrito.
  Saída: Nenhuma.
  Registradores: AF, BC.

NSETRD (016EH)
  Função: Prepara a VRAM para leitura seqüencial, usando a função de auto-incremento de endereço do VDP.
  Entrada: HL - endereço da VRAM a partir do qual os dados serão lidos. Todos os bits são válidos.
  Saída: Nenhuma.
  Registradores: AF.

NSTWRT (0171H)
  Função: Prepara a VRAM para a escrita seqüencial, usando a função de auto-incremento de endereço do VDP.
  Entrada: HL - endereço da VRAM a partir do qual os dados serão escritos. Todos os bits são válidos.
  Saída: Nenhuma.
  Registradores: Nenhum.

NRDVRM (0174H)
  Função: Lê o conteúdo de um byte da VRAM.
  Entrada: HL - endereço na VRAM do byte a ser lido.
  Saída: A - byte lido.
  Registradores: F.

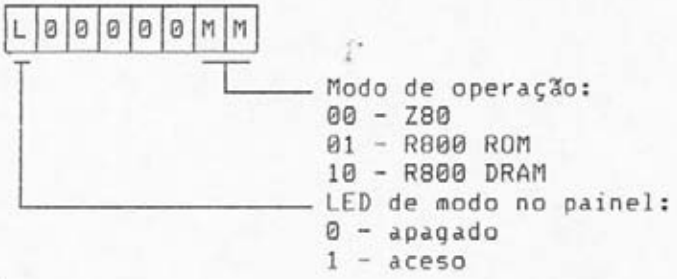
NWRVRM (0177H)
  Função: Escreve um byte de dados na VRAM.
  Entrada: HL - endereço na VRAM do byte a ser escrito.

A - byte a ser escrito.
Saída: Nenhuma.
Registradores: AF.

1.10 - ENTRADAS ADICIONADAS PARA O MSX turbo R

CHGCPU (0180H)
Função: Trocar de microprocessador (modo de operação).

Entrada: A -

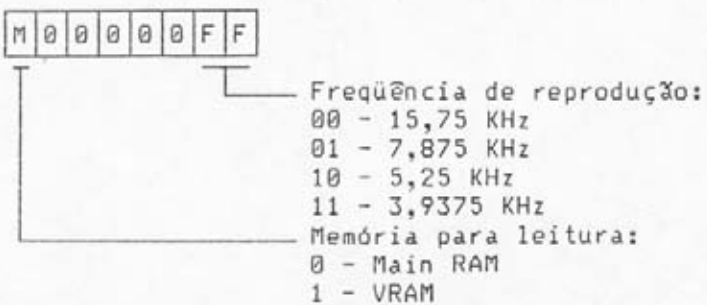


Saída: Nenhuma.
Registradores: AF.

GETCPU (0183H)
Função: Verificar em qual modo o computador está operando.
Entrada: Nenhuma.
Saída: A - 0=Z80; 1=R800 ROM; 2=R800 DRAM.
Registradores: AF.

PCMPLY (0186H)
Função: Reproduzir o som pelo PCM.
Entrada: HL - endereço de início para leitura.
BC - tamanho do bloco a reproduzir (comprimento).

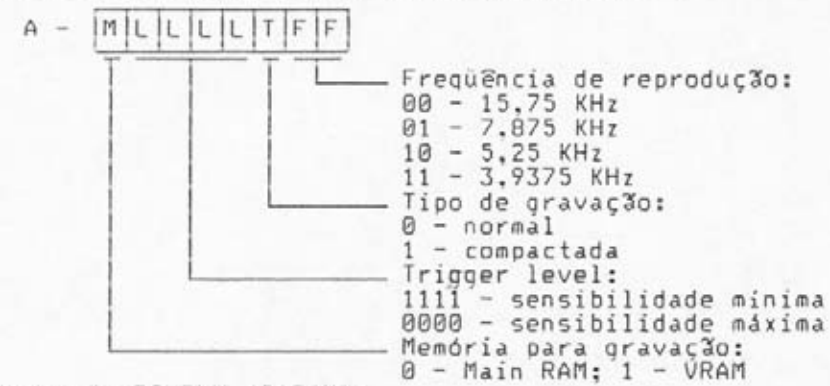
A -



Obs.: Usar a freqüência de 15,75 KHz apenas no modo R800 DRAM.
Saída: Flag CY: resetada - parou.
setada - parou porque houve erro.
A: 1 - tem erro na frequência.
2 - foi pressionada STOP.
Registradores: AF, BC, HL.

PCMREC (0189H)
Função: Digitalizar sons através do PCM.

Entrada: Igual a PCMPLY, exceto para o registrador A:



Saída: Mesma de PCMPLY (0186H).
Registradores: AF, BC, HL.

1.11 - ENTRADAS PARA A SUB-ROM

A seqüência de chamada para as rotinas da SUB-ROM é feita com o auxílio da rotina EXTROM (015FH) ou SUBROM (015CH), carregando IX com o endereço da rotina da SUB-ROM a ser chamada, e procedendo conforme o exemplo abaixo:

```
        LD   IX,INIPLT  ;carrega IX com o endereço da rotina
        CALL EXTROM     ;executa a rotina
        ...             ;retorna da rotina aqui
```

Quando o conteúdo de IX não deve ser destruído, use a seguinte seqüência de chamada:

```
INIPAL: PUSH IX         ;salva IX
        LD   IX,INIPLT  ;carrega IX com o endereço da rotina
        JP   SUBROM     ;executa a rotina
        ...             ;retorno da chamada de INIPAL
```

GRPPRT (0089H)
  Função: Imprime um caractere na tela gráfica (válida somente para as screens 5 a 8 e 10 a 12).
  Entrada: A - Código ASCII do caractere.
  Saída: Nenhuma.
  Registradores: Nenhum.

NVBXLN (00C9H)
  Função: Desenha uma caixa.
  Entrada: Ponto inicial: BC - coordenada X (horizontal).
  DE - coordenada Y (vertical).
  Ponto final: GXPOS (FCB3H) - coordenada X (horizontal)
  GYPOS (FCB5H) - coordenada Y (vertical).
  Cor: ATRBYT (F3F2H) para o atributo.
  Código de operação lógica: LOGOPR (FB02H).
  Saída: Nenhuma.
  Registradores: Todos.

NVBXFL (00CDH)
  Função: Desenha uma caixa pintada.
  Entrada: Mesma de NVBXLN (00C9H).

```
  Saída:   Nenhuma.
  Registradores: Todos.

CHGMOD (00D1H)
  Função:  Troca os modos de tela.
  Entrada: A - modo screen (0 a 8 ou 10 a 12).
  Saída:   Nenhuma.
  Registradores: Todos.

INITXT (00D5H)
  Função:  Inicializa a tela no modo texto (40 x 24).
  Entrada: TXTNAM (F3B3H) - endereço da tabela de nomes dos
                            caracteres.
           TXTCGP (F3B7H) - endereço da tabela geradora de
                            padrões dos caracteres.
  Saída:   Nenhuma.
  Registradores: Todos.

INIT32 (00D9H)
  Função:  Inicializa a tela no modo texto (32 x 24).
  Entrada: T32NAM (F3BDH) - Endereço da tabela de nomes dos
                            caracteres.
           T32COL (F3BFH) - Endereço da tabela de cores dos
                            caracteres.
           T32CGP (F3C1H) - Endereço da tabela de padrões dos
                            caracteres.
           T32ATR (F3C3H) - Endereço da tabela de atributos
                            dos sprites.
           T32PAT (F3C5H) - Endereço da tabela de padrões dos
                            sprites.
  Saída:   Nenhuma.
  Registradores: Todos.

INIGRP (00DDH)
  Função:  Inicializa a tela no modo gráfico de alta resolução
           (screen 2).
  Entrada: GRPNAM (F3C7H) - Endereço da tabela de nomes dos
                            caracteres.
           GRPCOL (F3C9H) - Endereço da tabela de cores dos
                            caracteres.
           GRPCGP (F3CBH) - Endereço da tabela de padrões dos
                            caracteres.
           GRPATR (F3CDH) - Endereço da tabela de atributos
                            dos sprites.
           GRPPAT (F3CFH) - Endereço da tabela de padrões dos
                            caracteres.
  Saída:   Nenhuma.
  Registradores: Todos.

INIMLT (00E1H)
  Função:  Inicializa a tela no modo multicor (screen 3).
  Entrada: MLTNAM (F3D1H) - Endereço da tabela de nomes dos
                            caracteres.
           MLTCOL (F3D3H) - Endereço da tabela de cores dos
                            caracteres.
           MLTCGP (F3D5H) - Endereço da tabela de padrões dos
                            caracteres.
           MLTATR (F3D7H) - Endereço da tabela de atributos
                            dos sprites.
```

MLTPAT (F3D9H) - Endereço da tabela de padrões dos sprites.
Saída: Nenhuma.
Registradores: Todos.

SETTXT (00E5H)
Função: Coloca o VDP no modo texto (40 x 24).
Entrada: Mesma de INITXT (00D5H/SUBROM).
Saída: Nenhuma.
Registradores: Todos.

SET32 (00E9H)
Função: Coloca o VDP no modo texto (32 x 24).
Entrada: Mesma de INIT32 (00D9H/SUBROM).
Saída: Nenhuma.
Registradores: Todos.

SETGRP (00EDH)
Função: Coloca o VDP no modo gráfico de alta resolução (screen 2).
Entrada: Mesma de INIGRP (00DDH/SUBROM)
Saída: Nenhuma.
Registradores: Todos.

SETMLT (00F1H)
Função: Coloca o VDP no modo multicor (screen 3).
Entrada: Mesma de INIMLT (00E1H/SUBROM).
Saída: Nenhuma.
Registradores: Todos.

CLRSPR (00F5H)
Função: Inicializa todos os sprites. A tabela de padrões dos sprites é limpa (preenchida com zeros), os números dos sprites são inicializados com a série 0 ≈ 31 e a cor dos sprites é igualada à cor de fundo. A localização vertical dos sprites é colocada em 217.
Entrada: SCRMOD (FCAFH) deve conter o modo screen.
Saída: Nenhuma.
Registradores: Todos.

CALPAT (00F9H)
Função: Retorna o endereço da tabela geradora do padrão de um sprite (esta rotina é a mesma que CALPAT (0084H) na Main-ROM).
Entrada: A - número do sprite.
Saída: HL - endereço na VRAM.
Registradores: AF, DE, HL.

CALATR (00FDH)
Função: Retorna o endereço da tabela de atributos de um sprite (esta rotina é a mesma que CALATR (0087H) na Main-ROM).
Entrada: A - número do sprite.
Saída: HL - endereço na VRAM.
Registradores: AF, DE, HL.

GSPSIZ (0101H)
Função: Retorna o tamanho atual dos sprites (esta rotina é a mesma que GSPSIZ (008AH) na Main-ROM).
Entrada: Nenhuma.

```
  Saída:   A - tamanho dos sprites em bytes. A flag CY é setada
               se o tamanho for 16 x 16 e resetada caso contrário.
  Registradores: AF.

GETPAT (0105H)
  Função: Retorna o padrão de um caractere.
  Entrada: A - código ASCII do caractere.
  Saída:   PATWRK (FC40H) - padrão do caractere.
  Registradores: Todos.

WRTVRM (0109H)
  Função: Escreve um byte de dados na VRAM.
  Entrada: HL - endereço da VRAM (0000H a FFFFH).
           A  - byte a ser escrito.
  Saída:   Nenhuma.
  Registradores: AF.

RDVRM  (010DH)
  Função: Lê o conteúdo de um byte da VRAM.
  Entrada: HL - endereço da VRAM a ser lido (0000H a FFFFH).
  Saída:   A - byte lido.
  Registradores: AF.

CHGCLR (0111H)
  Função: Troca as cores da tela.
  Entrada: A - modo screen da tela.
           FORCLR (F3E9H) - cor de frente.
           BAKCLR (F3EAH) - cor de fundo.
           BDRCLR (F3EBH) - cor da borda.
  Saída:   Nenhuma.
  Registradores: Todos.

CLSSUB (0115H)
  Função: Limpar a tela.
  Entrada: Nenhuma.
  Saída:   Nenhuma.
  Registradores: Todos.

DSPFNK (011DH)
  Função: Apresenta o conteúdo das teclas de função.
  Entrada: Nenhuma.
  Saída:   Nenhuma.
  Registradores: Todos.

WRTVDP (012DH)
  Função: Escreve dados em um registrador do VDP.
  Entrada: C - número do registrador.
           B - byte a ser escrito.
  Saída:   Nenhuma.
  Registradores: AF. BC.

VDPSTA (0131H)
  Função: Lê o conteúdo de um registrador do VDP.
  Entrada: A - número do registrador a ser lido (0 a 9).
  Saída:   A - dado lido.
  Registradores: F.

SETPAG (013DH)
  Função: Alterna as páginas de vídeo.
```

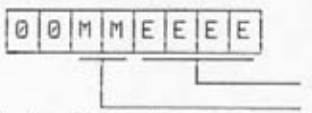
```
   Entrada: DPPAGE (FAF5H) - número da página apresentada no
                             vídeo.
            ACPAGE (FAF6H) - número da página ativa.
   Saída:   Nenhuma.
   Registradores: AF.

INIPLT (0141H)
   Função: Inicializa a paleta de cores (a paleta atual é gravada
           na VRAM.
   Entrada: Nenhuma.
   Saída:   Nenhuma.
   Registradores: AF, BC, DE.

RSTPLT (0145H)
   Função:  Recupera a paleta de cores da VRAM.
   Entrada: Nenhuma.
   Saída:   Nenhuma.
   Registradores: AF, BC, DE.

GETPLT (0149H)
   Função:  Retorna o código de cores da paleta.
   Entrada: A - número da paleta (0 a 15).
   Saída:   B - 4 bits altos para código do vermelho;
            B - 4 bits baixos para código do azul;
            C - 4 bits baixos para código do verde.
   Registradores: AF, DE.

SETPLT (014DH)
   Função:  Modifica o código de cores da paleta.
   Entrada: D - número da paleta (0 a 15).
            A - 4 bits altos para o código do vermelho;
            A - 4 bits baixos para o código do azul;
            E - 4 bits baixos para o código do verde.
   Saída:   Nenhuma.
   Registradfores: AF, DE.

BEEP   (017DH)
   Função:  Gera um beep.
   Entrada: Nenhuma.
   Saída:   Nenhuma.
   Registradores: Todos.

PROMPT (0181H)
   Função:  Apresenta o sinal de prompt.
   Entrada: Nenhuma.
   Saída:   Nenhuma.
   Registradores: Todos.

NEWPAD (01ADH)
   Função:  Lê o estado do mouse ou da light-pen (caneta ótica).
   Entrada: A - deve conter os valores para chamada descritos a-
                baixo (as descrições entre parênteses são valores
                de retorno, sempre em A.
                8 - checa se a light-pen está conectada (se esti-
                    ver, A=255).
                9 - retorna a coordenada X (horizontal) em A.
                10 - retorna a coordenada Y (vertical) em A.
                11 - retorna o estado da chave da light-pen (se
                     estiver pressionada, A=255).
```

12 - checa se o mouse está conectado no port 1 do joystick (se estiver, A=255).
13 - retorna a coordenada na direção X em A.
14 - retorna a coordenada na direção Y em A.
15 - sempre 0.
16 - checa se o mouse está conectado no port 2 do joystick (se estiver, A=255).
17 - retorna a coordenada na direção X em A.
18 - retorna a coordenada na direção Y em A.
19 - sempre 0.

Saída: A - contém os valores de retorno, conforme descrito acima.
Registradores: Todos.
Obs.: esta rotina foi modificada nos modelos MSX turbo R.

CHGMDP (01B5H)
Função: Troca o modo do VDP. A paleta de cores é inicializada.
Entrada: A - modo screen (0 a 8 para MSX2 e 0 a 8 / 10 a 12 para MSX2+ ou superior).
Saída: Nenhuma.
Registradores: Todos.

REDCLK (01F5H)
Função: Lê um dado da memória do relógio.
Entrada: C - endereço da RAM do relógio, conforme abaixo:



Saída: A - dado lido (apenas os 4 bits baixos são válidos).
Registradores: AF.

WRTCLK (01F9H)
Função: Escreve um dado na memória do relógio.
Entrada: A - dado a ser escrito.
C - endereço da RAM do relógio (igual a REDCLK).
Saída: Nenhuma.
Registradores: F.

1.12 - ROTINAS DE TRANSFERÊNCIA DE DADOS (BIT BLOCK TRANSFER)

Este conjunto de rotinas da SUB-ROM foi desenvolvido para a transferência de dados entre a RAM, VRAM e disco, de forma semelhante ao comando COPY do BASIC. Essas rotinas são de fácil execução, tornando disponíveis para programas assembly funções de transferência de dados de forma fácil, rápida e segura.

BLTVV (0191H)
Função: Transfere dados de uma área da VRAM para outra.
Entrada: HL - Deve conter o valor F562H.

SX	(F562H,2)	- coordenada X da fonte.
SY	(F564H,2)	- coordenada Y da fonte.
DX	(F566H,2)	- coordenada X do destino.
DY	(F568H,2)	- coordenada Y do destino.
NX	(F56AH,2)	- número de pontos na direção X.
NY	(F56CH,2)	- número de pontos na direção Y.
CDUMMY	(F56EH,1)	- dummy (não requer dados).
ARGT	(F56FH,1)	- seleciona a direção e expansão da VRAM (igual a R#45 do VDP).

```
            LOGOP  (F570H,1) - código de operação lógica (igual
                               aos códigos do VDP).
  Saída:    A flag CY é resetada.
  Registradores: Todos.
  Obs.: o número após os endereços dados representa a quantidade
        de bytes que a variável de sistema requer. Essa represen-
        tação será usada daqui em diante.
```

As rotinas seguintes requerem que o espaço de memória a ser movido seja alocado da seguinte forma para cada screen:

```
SCREEN 6:
  (pontos na direção X) * (pontos na direção Y) / 4 + 4

SCREENS 5 e 7:
  (pontos na direção X) * (pontos na direção Y) / 2 + 4

SCREENS 8, 10, 11 e 12:
  (pontos na direção X) * (pontos na direção Y) + 4

BLTVM  (0195H)
  Função:  Transfere dados da RAM para a VRAM.
  Entrada: HL - Deve conter o valor F562H.
           DPTR   (F562H,2) - endereço-fonte na RAM.
           DUMMY  (F564H,2) - dummy (não requer dados).
           DX     (F566H,2) - coordenada X de destino.
           DY     (F568H,2) - coordenada Y de destino.
           NX     (F56AH,2) - número de pontos na direção X (não
                              requer dados; já está setada).
           NY     (F56CH,2) - número de pontos na direção Y (não
                              requer dados; já está setada).
           CDUMMY (F56EH,1) - dummy (não requer dados).
           ARGT   (F56FH,1) - seleciona a direção e a expansão
                              da VRAM (igual a R#45 do VDP).
           LOGOP  (F570H,1) - código de operação lógica (igual
                              aos códigos do VDP).
  Saída:   A flag CY é setada se o número de bytes a transferir
           estiver incorreto.
  Registradores: Todos.

BLTMV  (0199H)
  Função:  Transfere dados da VRAM para a RAM.
  Entrada: HL - Deve conter o valor F562H.
           SX     (F562H,2) - coordenada X da fonte.
           SY     (F564H,2) - coordenada Y da fonte.
           DPTR   (F566H,2) - endereço de destino na RAM.
           DUMMY  (F568H,2) - dummy (não requer dados).
           NX     (F56AH,2) - número de pontos na direção X.
           NY     (F56CH,2) - número de pontos na direção Y.
           CDUMMY (F56EH,1) - dummy (não requer dados).
           ARGT   (F56FH,1) - seleciona a expansão e a direção
                              da VRAM (igual a R#45 do VDP).
  Saída:   A flag CY é resetada.
  Registradores: Todos.
```

As rotinas seguintes transferem dados entre a RAM, VRAM e o disco. Para isso, deve-se especificar o nome do arquivo no disco como no exemplo na página seguinte.

```
        LD   HL,FNAME     :pega o end. do nome do arquivo
        LD   (FNPTR),HL   :seta o end. na variável de sistema
        :
        :
FNAME:  DEFB 22H,'B:TESTE.PIC',22H,00H :nome do arquivo
                                        + marca de fim
```

Como estas rotinas também são usadas pelo interpretador BASIC, se ocorrer algum erro durante a transferência, o controle é passado automaticamente ao manipulador de erro, que depois devolve o controle ao interpretador.

Para evitar que isso ocorra, use o hook HERRO (FEFDH) para interceptar o erro antes que este seja transferido ao interpretador. Observe que o código do erro fica no registrador E, podendo ser usado pelo programa assembly.

BLTVD (019DH)
  Função: Transfere dados do disco para a VRAM.
  Entrada: HL - Deve conter o valor F562H.
  - FNPTR (F562H,2) - endereço do nome do arquivo.
  - DUMMY (F564H,2) - dummy (não requer dados).
  - DX (F566H,2) - coordenada X do destino.
  - DY (F568H,2) - coordenada Y do destino.
  - NX (F56AH,2) - número de pontos na direção X (não requer dados; já setada).
  - NY (F56CH,2) - número de pontos na direção Y (não requer dados; já setada).
  - CDUMMY (F56EH,1) - dummy (não requer dados).
  - ARGT (F56FH,1) - seleciona a expansão e a direção da VRAM (igual a R#45 do VDP).
  - LOGOP (F570H,1) - código de operação lógica (igual aos códigos do VDP).

  Saída: A flag CY é setada se houver algum erro nos parâmetros.
  Registradores: Todos.

BLTDV (01A1H)
  Função: Transfere dados da VRAM para o disco.
  Entrada: HL - Deve conter o valor F562H.
  - SX (F562H,2) - coordenada X da fonte.
  - SY (F564H,2) - coordenada Y da fonte.
  - FNPTR (F566H,2) - endereço do nome do arquivo.
  - DUMMY (F568H,2) - dummy (não requer dados).
  - NX (F56AH,2) - número de pontos na direção X.
  - NY (F56CH,2) - número de pontos na direção Y.
  - CDUMMY (F56EH,1) - dummy (não requer dados).

  Saída: A flag CY é resetada.
  Registradores: Todos.

BLTMD (01A5H)
  Função: Carrega dados do disco para a RAM.
  Entrada: HL - Deve conter o valor F562H.
  - FNPTR (F562H,2) - endereço do nome do arquivo.
  - SY (F564H,2) - dummy (não requer dados).
  - SPTR (F566H,2) - endereço inicial dos dados a serem carregados.
  - EPTR (F568H,2) - endereço final dos dados a serem carregados.

  Saída: A flag CY é resetada.
  Registradores: Todos.

```
BLTDM  (01A9H)
  Função:  Grava dados da RAM no disco.
  Entrada: HL - deve conter o valor F562H.
           SPTR   (F562H,2) - endereço inicial dos dados a
                              serem gravados.
           EPTR   (F564H,2) - endereço final dos dados a
                              serem gravados.
           FNPTR  (F566H,2) - endereço do nome do arquivo.
  Saída:   A flag CY é resetada.
  Registradores: Todos.
```

2 - O MATH-PACK (PACOTE MATEMÁTICO)

O Math-Pack *(Pacote Matemático)* é um conjunto de rotinas matemáticas que não pertencem ao BIOS e que constituem o centro das operações matemáticas do MSX-BASIC. Essas rotinas podem ser utilizadas por programas assembly, tornando disponíveis operações com ponto flutuante, aritméticas, logarítmicas e trigonométricas, além de várias funções especiais.

As operações envolvendo números reais com o Math-Pack são realizadas em BCD (Binary Coded Decimal). Os números podem ser inteiros de 2 bytes (-32768 a +32767), de precisão simples (6 dígitos) ocupando 4 bytes ou de precisão dupla (14 dígitos) ocupando 8 bytes, conforme ilustrado na figura abaixo.

Formato BCD para expressar números reais

Exemplo de número de precisão simples:

```
123.456  ->  0,123456E+6    ou
                0   1   2   3
DAC  ->    | 46 | 12 | 34 | 56 |
```

Exemplo de número de precisão dupla:

```
123.456,78901234  ->  0,12345678901234E+6    ou
                        0   1   2   3   4   5   6   7
DAC  ->    | 46 | 12 | 34 | 56 | 78 | 90 | 12 | 34 |
```

Observe que os dígitos que constituem a mantissa são sempre considerados como colocados logo após a vírgula.

Um número real é composto por uma mantissa, um sinal e

um expoente. O sinal da mantissa é representado por 0 (positivo) ou 1 (negativo). O expoente é uma expressão binária de 7 bits que representa uma potência de 10 e pode variar de -63 a +63, conforme a ilustração abaixo.

+/-	expoente							
0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	indefinido (-0?)
x	0	0	0	0	0	0	1	-63ª potência de 10
x	1	0	0	0	0	0	0	0ª potência de 10
x	1	1	1	1	1	1	1	+63ª potência de 10

Para realizar operações com o Math-Pack, existem duas áreas de memória reservadas, que são o "DAC" (Decimal ACumulator, F7F6H) e "ARG" (F847H). Por exemplo, numa multiplicação, o produto dos números contidos em DAC e ARG é calculado e o resultado colocado em DAC.

No DAC, podem ser armazenados números de dupla precisão, simples precisão e inteiros de dois bytes, sendo que nesse último caso os dois bytes que representam o número inteiro são armazenados em DAC+2 e DAC+3. Para que as rotinas do Math-Pack possam distingüir que tipo de número está armazenado em DAC, a variável de sistema VALTYP (F663H) é usada, devendo conter o valor 2 para números inteiros, 4 para números de precisão simples e 8 para números de precisão dupla.

Ao usar rotinas do Math-Pack em assembler, deve-se tomar um cuidado especial. Como são rotinas utilizadas pelo interpretador BASIC, caso ocorra algum erro (como divisão por zero ou overflow, por exemplo), o controle é automaticamente transferido para o manipulador de erro que depois devolve o controle ao interpretador. Para evitar que isso ocorra, use o hook HERRO (FFB1H) para interceptar o erro. Observe que o código de erro fica no registrador E da UCP, podendo também ser usado pelo programa assembly.

Para usar as rotinas do Math-Pack em programas assembly deve-se proceder exatamente da mesma forma como se chama as rotinas do BIOS. Colocam-se os devidos valores em ARG, DAC e VALTYP e eventualmente em algum registrador da UCP e chama-se a rotina desejada através da instrução CALL. A única observação a fazer é que pouquíssimas rotinas preservam algum registrador; portanto sempre salve na pilha os registradores que não devem ser destruídos. A seguir, estão listadas as rotinas do Math-Pack, com as labels, o respectivo endereço e a função que cada uma desempenha.

2.1 - ÁREA DE TRABALHO

VALTYP (F663H) 1 byte
  Formato do número contido em DAC (2, 4 ou 8).

DAC (F7F6H) 16 bytes
  Acumulador de ponto flutuante no formato BCD.

ARG (F847H) 16 bytes
Argumento para uso com DAC.

2.2 - FUNÇÕES MATEMÁTICAS

```
DECSUB (268CH)      DAC  <-  DAC - ARG
DECADD (269AH)      DAC  <-  DAC + ARG
DECMUL (27E6H)      DAC  <-  DAC * ARG
DECDIV (289FH)      DAC  <-  DAC / ARG
SNGEXP (37C8H)      DAC  <-  DAC ^ ARG - Simples precisão
DBLEXP (37D7H)      DAC  <-  DAC ^ ARG - Dupla precisão
COS    (2993H)      DAC  <-  COS (DAC)
SIN    (29ACH)      DAC  <-  SIN (DAC)
TAN    (29FBH)      DAC  <-  TAN (DAC)
ATN    (2A14H)      DAC  <-  ATN (DAC)
SQR    (2AFFH)      DAC  <-  SQR (DAC)
LOG    (2A72H)      DAC  <-  LOG (DAC) - Base Neperiana
EXP    (2B4AH)      DAC  <-  EXP (DAC) - Base Neperiana
```

2.3 - OUTRAS FUNÇÕES

```
DECNRM (26FAH)      Normaliza DAC       *1
RND    (2BDFH)      DAC  <-  RND (DAC)
SIGN   (2E71H)      A  <-  Sinal da mantissa em DAC
ABSFN  (2E82H)      DAC  <-  ABS (DAC)
NEG    (2E8DH)      DAC  <-  NEG (DAC)
SGN    (2E97H)      DAC  <-  SGN (DAC)  *2
```

*1 - Zeros excessivos na mantissa são removidos. Por exemplo,
0,00123 -> 0,123E-2)
*2 - Na função SGN, o resultado é representado por um número
inteiro de 2 bytes.

2.4 - OPERAÇÕES COM NÚMEROS INTEIROS

```
UMULT  (314AH)      DE  <-  DE * BC
ISUB   (3167H)      HL  <-  DE - HL
IADD   (3172H)      HL  <-  DE + HL
IMULT  (3193H)      HL  <-  DE * HL
IDIV   (31E6H)      HL  <-  DE / HL
IMOD   (323AH)      HL  <-  DE mod HL
                    DE  <-  DE / HL
INTEXP (383FH)      DAC <-  DE ^ HL
```

2.5 - CONVERSÃO DE TIPO

```
FRCINT (2F8AH) - Converte DAC para número inteiro de 2 bytes
                 (DAC +2, +3).
FRCSNG (2FB2H) - Converte DAC para número de precisão simples.
FRCDBL (303AH) - Converte DAC para número de precisão dupla.
FIXER  (30BEH) - DAC  <-  SGN (DAC) * INT (ABS (DAC))
```

Obs.: Depois da conversão, VALTYP (F663H) conterá o valor que representa o tipo de número convertido armazenado em DAC (2, 4 ou 8).

2.6 - MOVIMENTO

```
MAF    (2C4DH)      ARG  <-  DAC      Dupla precisão
MAM    (2C50H)      ARG  <-  (HL)     Dupla precisão
MOV8DH (2C53H)     (DE)  <-  (HL)     Dupla precisão
```

MFA	(2C59H)	DAC	<-	ARG	Dupla precisão
MFM	(2C5CH)	DAC	<-	(HL)	Dupla precisão
MMF	(2C67H)	(HL)	<-	DAC	Dupla precisão
MOV8HD	(2C6AH)	(HL)	<-	(DE)	Dupla precisão
XTF	(2C6FH)	(SP)	<->	DAC	Dupla precisão
PHA	(2CC7H)	ARG	<-	(SP)	Dupla precisão
PHF	(2CCCH)	DAC	<-	(SP)	Dupla precisão
PPA	(2CDCH)	(SP)	<-	ARG	Dupla precisão
PPF	(2CE1H)	(SP)	<-	DAC	Dupla precisão
PUSHF	(2EB1H)	DAC	<-	(SP)	Precisão simples
MOVFM	(2EBEH)	DAC	<-	(HL)	Precisão simples
MOVFR	(2EC1H)	DAC	<-	CBED	Precisão simples
MOVRF	(2ECCH)	CBED	<-	DAC	Precisão simples
MOVRMI	(2ED6H)	CBED	<-	(HL)	Precisão simples
MOVRM	(2EDFH)	BCDE	<-	(HL)	Precisão simples
MOVMF	(2EE8H)	(HL)	<-	DAC	Precisão simples
MOVE	(2EEBH)	(HL)	<-	(DE)	Precisão simples
VMOVAM	(2EEFH)	ARG	<-	(HL)	VALTYP
MOVVFM	(2EF2H)	(DE)	<-	(HL)	VALTYP
VMOVE	(2EF3H)	(HL)	<-	(DE)	VALTYP
VMOVFA	(2F05H)	DAC	<-	ARG	VALTYP
VMOVFM	(2F08H)	DAC	<-	(HL)	VALTYP
VMOVAF	(2F0DH)	ARG	<-	DAC	VALTYP
VMOVMF	(2F10H)	(HL)	<-	DAC	VALTYP

Obs.: (HL) e (DE) significam os endereços de memória apontados por HL e DE. Quatro nomes de registradores juntos contêm um número de precisão simples (sinal+expoente; 1º e 2º dígitos; 3º e 4º dígitos; 5º e 6º dígitos). Quando o objeto for VALTYP, o movimento será de acordo com o tipo indicado por VALTYP (F663H), ou seja, 2, 4 ou 8 bytes.

2.7 - COMPARAÇÕES

			esquerdo	direito
ICOMP	(2F4DH)	Inteiro de 2 bytes	DE	HL
DCOMP	(2F21H)	Precisão simples	CBED	DAC
XDCOMP	(2F5CH)	Precisão dupla	ARG	DAC

O resultado da comparação será colocado no registrador A, conforme mostrado abaixo:

A=01H -> esquerdo < direito
A=00H -> esquerdo = direito
A=FFH -> esquerdo > direito

2.8 - OPERAÇÕES DE PONTO FLUTUANTE E I/O

FIN (3299H)
Função: Converte uma string representando um número real para o formato BCD e o armazena em DAC.
Entrada: HL - Endereço do primeiro caractere da string.
A - Primeiro caractere da string.
Saída: DAC - Número real em BCD.
C - FFH - sem ponto decimal;
00H - com ponto decimal.
B - Número de dígitos após o ponto decimal.
D - Número total de dígitos.

```
FOUT   (3425H)
  Função: Converte um número real contido em DAC para uma
          string sem formatar.

PUFOUT (3426H)
  Função: Converte um número real contido em DAC para uma
          string formatando.
  Entrada: A - formato:
               bit 7 - 0-não formatado    1-formatado
               bit 6   0-sem vírgulas     1-com vírgulas a
                                            cada 3 dígitos.
               bit 5 - 0-sem significado  1-preenche espaços
                                            com "*".
               bit 4 - 0-sem significado  1-adiciona "$" antes
                                            do número.
               bit 3 - 0-sem significado  1-coloca sinal de +
                                            para números posi-
                                            tivos.
               bit 2 - 0-sem significado  1-coloca o sinal
                                            depois do número.
               bit 1 - Não utilizado.
               bit 0 - 0-ponto fixo       1-ponto flutuante
           B - Número de dígitos antes do ponto decimal.
           C - Número de dígitos depois do ponto decimal, in-
               cluindo este.
  Saída:   HL - endereço do primeiro caractere da string

FOUTB  (371AH)
  Função: Converte um número inteiro contido em DAC para uma
          expressão binária.

FOUTO  (371EH)
  Função: Converte um número inteiro contido em DAC para uma
          expressão octal.

FOUTH  (3722H)
  Função: Converte um número inteiro contido em DAC para uma
          expressão hexadecimal.
  Entrada: DAC+2 - Número inteiro
           VALTYP = 2
  Saída:   HL - Endereço do primeiro caractere da string.
```

3 - O INTERPRETADOR BASIC

A maior parte do interpretador BASIC reside na página 1 da ROM. A área de texto de um programa BASIC se inicia normalmente no endereço 8000H, que corresponde ao início da página 2, mas pode ser alterada mudando-se a variável de sistema TXTTAB (F676H) que contém inicialmente o valor 8000H e indica o início da área de texto BASIC.

3.1 - AS TOKENS

Para cada palavra reservada do BASIC, existe um código correspondente chamado "token" ou "átomo". Uma token nada mais é que um único byte representando uma palavra reservada do BASIC.

Como se pode concluir, o texto BASIC não é armazenado na forma ASCII, mas em uma forma mais compacta. A finalidade das tokens não é apenas tornar o texto BASIC mais compacto, mas tam-

bém mais rápido; afinal, durante o processamento, o interpretador tem que decodificar apenas um byte, ao invés de toda a seqüência de códigos ASCII que o representaria.

Um comando BASIC, por exemplo, "PRINT A", estará armazenado na área de texto BASIC da seguinte forma:

```
byte 91H - token do comando PRINT
byte 20H - espaço
byte 41H - código ASCII da variável 'A'
```

Já as funções do BASIC são armazenadas de uma forma ligeiramente diferente. As tokens das funções são precedidas por um byte FFH e têm o seu bit 7 setado. Por exemplo, uma função BASIC do tipo "X=SIN(A)" é armazenada da seguinte forma:

```
byte 58H - código ASCII da variável 'X'
byte EFH - token do sinal  "="
byte FFH - identificador de função
byte 89H - token setada da função SIN
byte 28H - código ASCII de '('
byte 41H - código ASCII da variável 'A'
byte 29H - código ASCII de ')'
```

Veja todos os comandos e funções do BASIC com suas respectivas tokens na seção "CHAMANDO COMANDOS EM BASIC".

3.2 - ESTRUTURA DAS LINHAS DE PROGRAMA

A maneira pela qual as linhas são armazenadas na área de texto BASIC é bastante simples.

Os dois primeiros bytes (normalmente 8001H e 8002H) contêm o endereço de início da próxima linha; os dois seguintes contêm o número da linha (que pode variar de 0 65529) e em seguida vêm os bytes que armazenam a linha propriamente dita, podendo ter até 254 bytes, sendo que o último byte deve ser 00H, indicando o fim de linha. Quando for o fim do programa, são acrescidos mais dois bytes 00H, indicando este fato.

Os números são armazenados de uma forma especial, visando a economizar o máximo possível de memória na área de texto. Os números inteiros são tratados de uma forma bastante peculiar, sendo divididos em três grupos: 0 a 9, 10 a 255 e 256 a 32767. Para os números inteiros de 0 a 9, há uma espécie de "token" que o identifica como tal, conforme a tabela abaixo:

0 - 11H	2 - 13H	4 - 15H	6 - 17H	8 - 19H
1 - 12H	3 - 14H	5 - 16H	7 - 18H	9 - 1AH

Para os números inteiros de 10 a 255, é colocado um byte de identificação antes do número, que neste caso é 0FH. Logo após o byte de identificação, vem um byte representando numericamente o valor, de 10 a 255. Para os números inteiros de 256 a 32767, também existe um byte de identificação (1CH) seguido de dois bytes que armazenam o número na forma LSB-MSB. Se um número inteiro for negativo, este será precedido pela token do sinal de "-" (F2H).

Os números de precisão simples são armazenados em quatro bytes, na forma BCD, precedidos pelo byte de identificação 1DH. Os números de precisão dupla são armazenados em oito bytes precedidos pelo byte de identificação 1FH.

Os números armazenados em outras bases (binário, octal e hexadecimal) também têm seus bytes de identificação próprios. Para um número binário, são dois bytes ID, no caso 26H e 42H, ou "&B", sendo o número binário armazenado na forma ASCII. Já os números octais têm como ID o byte 0BH e o número armazenados em dois bytes na forma LSB-MSB. Para os números hexadecimais, o byte ID é 0CH e o número também é armazenado na forma LSB-MSB.

Já os números que referem a linhas de programas (nas instruções GOTO e GOSUB por exemplo) também têm um tratamento bem peculiar. Durante a digitação do programa, o número de linha é armazenado em dois bytes precedidos pelo byte ID 0EH. Quando a linha for executada pela primeira vez, o interpretador mudará o byte ID para 0DH e os dois bytes seguintes conterão o endereço de início da linha respectiva, e não mais o número de linha. Isto é feito para acelerar a execução do programa na próxima vez que for rodado.

3.3 - A ÁREA DE VARIÁVEIS DO BASIC

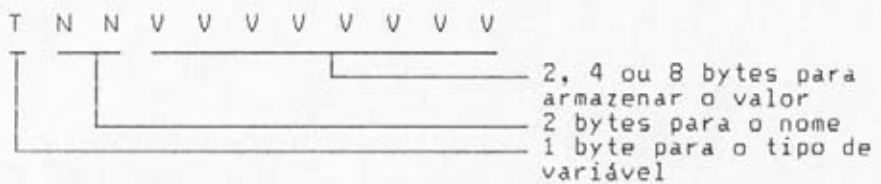
A área de memória logo acima do final do texto BASIC é alocada para armazenar as variáveis do programa. Esta área se inicia no endereço apontado por VARTAB (F6C2H) e termina no endereço apontado por STREND. Cada vez que uma variável é consultada, o interpretador procura a mesma na área delimitada por VARTAB e STREND, e caso não a encontre, assume o valor 0 para variáveis numéricas ou string nula para variáveis alfanuméricas.

Sempre que uma nova linha BASIC é introduzida ou o comando CLEAR é executado, o valor de STREND é igualado ao valor de VARTAB e conseqüentemente todas as variáveis do programa são limpas e ficam nulas.

Existem quatro tipos de variáveis do BASIC:

numéricas inteiras:	ocupam 2 bytes
numéricas de precisão simples:	ocupam 4 bytes
numéricas de precisão dupla:	ocupam 8 bypes
alfanuméricas (strings):	ocupam 3 bytes

As variáveis numéricas possuem a sintaxe de armazenamento ilustrada abaixo:



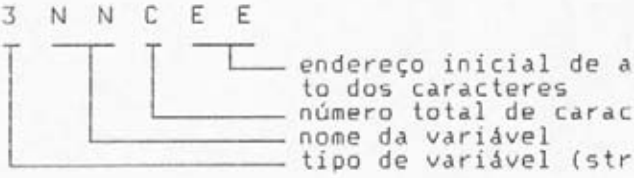
O primeiro byte indica o tipo de variável numérica que está armazenada, sendo que seu valor também já indica o número de bytes ocupados por ela (2, 4 ou 8).

O interpretador assume como default as variáveis de dupla precisão, mas podem ser alteradas pelos comandos DEFINT, DEFSNG, DEFDBL e DEFSTR. Estes comandos possuem uma tabela que se inicia em F6CAH e tem 26 bytes, um para cada letra do alfabeto, que indica que a variável cujo nome se inicia com aquela letra deve assumir o tipo indicado:

02 - inteira
03 - string
04 - simples precisão
08 - dupla precisão

Os sinais de identificação imediata do tipo de variável (%, !, # e $) têm precedência sobre os valores indicados por esta tabela.

As variáveis alfanuméricas (strings) têm uma forma de armazenamento ligeiramente diferente, cuja sintaxe é apresentada abaixo:



A variável de sistema FRETOP (F69BH) armazena o endereço que receberá o último caractere da string que está sendo armazenada.

Se houver uma atribuição direta a uma variável alfanumérica (tipo A$="XYZ"), o endereço que o apontador indicará estará na própria área de texto do programa BASIC e não na área reservada para variáveis string. Entretanto, qualquer operação feita com esta variável que a modifique fará com que ela seja transferida para a área reservada e o apontador conterá o endereço respectivo nesta.

3.4 - CHAMANDO COMANDOS EM BASIC

É possível utilizar as rotinas do interpretador em programas assembly. Observe entretanto que, ao se chamar um comando em BASIC, passamos literalmente a trabalhar em BASIC, devendo-se levar em consideração duas coisas. Primeira: algum erro ou bug acidental que ocorra durante a execução da rotina fará com que o controle seja devolvido automaticamente ao nível de comando BASIC. Para evitar que isso ocorra, use o hook HERRO (FFB1H) para interceptar o erro. O código de erro fica no registrador E, podendo ser utilizado pelo programa assembly. Segunda: um comando em BASIC só deve ser chamado se o algoritmo a ser utilizado for muito complexo, como as instruções CIRCLE, LINE, DRAW, PLAY e outras de execução complexa. Dê sempre preferência às rotinas do BIOS quando estas puderem realizar o mesmo trabalho, pois são muito mais rápidas e utilizam menos memória que as rotinas do BASIC.

Para chamar um comando em BASIC, normalmente basta setar no par HL o endereço de uma falsa linha BASIC, terminada por um byte 0, preferencialmente na forma tokenizada. Porém, alguns

comandos exigem que mais registradores e até variáveis de sistema sejam carregadas, mas são comandos que praticamnte não têm utilidade para o programador assembly. Para obter a forma tokenizada do comando, há um algoritmo simples: basta digitar a linha de programa desejada e depois usar um programa monitor (desassembler) para observar a linha tokenizada. Lembre-se que quando for uma função, a token correspondente será precedida por um byte FFH.

O procedimento para se chamar o comando BASIC, após setar o par HL, deve ser feito através da rotina CALBAS (0159H) do BIOS. Também podem ser usadas as rotinas CALSLT (001CH) ou CALLF (0030H) do BIOS, setando em IY o slot da Main-ROM (págs. 0 e 1).

O passo seguinte é verificar em qual endereço está a rotina que executa o comando desejado. Para isso, existe uma tabela de endereços que se inicia em 392EH, e os endereços por ela apontados seguem em ordem crescente de token do comando.

Abaixo, segue uma tabela com os nomes, tokens e endereços dos comandos BASIC. Para utilizá-la é simples: basta consultar o conteúdo do respectivo endereço. Por exemplo, o comando PLAY tem seu ponto de entrada especificado em 39AEH. Basta consultar o conteúdo desse endereço [LD IX,(39AEH)] para obter o ponto de entrada da rotina que executa o comando PLAY.

Comando	Token	Endereço	Comando	Token	Endereço
AUTO	A9H	3973H	ERROR	A6H	3978H
AND	F6H	3A18H	ERL	E1H	39EEH
ATTR$	E9H	39FEH	ERR	E2H	39F0H
BASE	C9H	39BEH	EQU	F9H	3A1EH
BSAVE	D0H	39CCH	FOR	82H	3920H
BLOAD	CFH	39CAH	FIELD	B1H	398EH
BEEP	C0H	39ACH	FILES	B7H	39AAH
CALL	CAH	39C0H	FN	DEH	39E8H
CLOSE	B4H	3994H	GOTO	89H	393EH
COPY	D6H	39D8H	GO TO	89H	393EH
CONT	99H	395EH	GOSUB	8EH	3948H
CLEAR	92H	3950H	GET	B2H	3990H
CLOAD	9BH	3962H	INPUT	85H	3936H
CSAVE	9AH	3960H	IF	8BH	3942H
CSRLIN	E8H	39FCH	INSTR	E5H	39F6H
CIRCLE	BCH	39A4H	IMP	FAH	3A20H
COLOR	BDH	39A6H	INKEY$	ECH	3A04H
CLS	9FH	396AH	IPL	D5H	39D6H
CMD	D7H	39DAH	KILL	D4H	39D4H
DELETE	A8H	397CH	KEY	CCH	3964H
DATA	84H	3934H	LPRINT	9DH	394CH
DIM	86H	3938H	LLIST	9EH	3968H
DEFSTR	ABH	3982H	LET	88H	393CH
DEFINT	ACH	3984H	LOCATE	D8H	39DCH
DEFSNG	ADH	3986H	LINE	AFH	398AH
DEFDBL	AEH	3988H	LOAD	B5H	3996H
DSKO$	D1H	39CEH	LSET	B8H	399CH
DEF	97H	395AH	LIST	93H	3952H
DSKI$	EAH	3A00H	LFILES	BBH	39A2H
DRAW	BEH	39A8H	MOTOR	CEH	39C8H
ELSE	A1H	396EH	MERGE	B6H	3998H
END	81H	392EH	MOD	FBH	3A22H
ERASE	A5H	3976H	MAX	CDH	39C6H

NEXT	83H	3932H	SCREEN	C5H	39B6H
NAME	D3H	39D2H	SPRITE	C7H	39BAH
NEW	94H	3954H	STOP	90H	394CH
NOT	E0H	39ECH	SWAP	A4H	3974H
OPEN	B0H	398CH	SET	D2H	39D0H
OUT	9CH	3964H	SAVE	BAH	39A0H
ON	95H	3956H	SPC(	DFH	39EAH
OR	F7H	3A1AH	STEP	DCH	39E4H
OFF	EBH	3A02H	STRING$	E3H	39F2H
PRINT	91H	394EH	SPACE$	19H	397EH
PUT	B3H	3992H	SOUND	C4H	39B4H
POKE	98H	395CH	THEN	DAH	39E0H
PSET	C2H	39B0H	TRON	A2H	3970H
PRESET	C3H	39B2H	TROFF	A3H	3972H
POINT	EDH	3A06H	TAB(	DBH	39E2H
PAINT	BFH	39AAH	TO	D9H	39DEH
PLAY	C1H	39AEH	TIME	CBH	39C2H
RETURN	8EH	3948H	USING	E4H	39F4H
READ	87H	393AH	USR	DDH	39E6H
RUN	8AH	3940H	VARPTR	E7H	39FAH
RESTORE	8CH	3944H	VDP	C8H	39BCH
REM	8FH	394AH	VPOKE	C6H	39B8H
RESUME	A7H	397AH	WIDTH	A0H	396CH
RSET	B9H	399EH	WAIT	96H	3958H
RENUM	AAH	3980H	XOR	F8H	3A1CH

Estes são os comandos do BASIC com suas respectivas tokens e endereços apontadores. Existem também as funções do BASIC, cuja token vem precedida por um byte FFH na forma tokenizada, indicador de função. A tabela que contém os endereços das funções inicia em 39DEH e tem a mesma estrutura da tabela de comandos. A tabela abaixo relaciona as funções do BASIC e suas respectivas tokens.

Comando	Token	Endereço	Comando	Token	Endereço
ABS	06H	39E8H	LEN	12H	3A00H
ATN	0EH	39F8H	LEFT$	01H	39DEH
ASC	15H	3A06H	LOF	2DH	3A36H
BIN$	1DH	3A16H	MKI$	2EH	3A38H
CINT	1EH	3A18H	MKS$	2FH	3A3AH
CSNG	1FH	3A1AH	MKD$	30H	3A3CH
CDBL	20H	3A1CH	MID$	03H	39E2H
CVI	28H	3A2CH	OCT$	1AH	3A10H
CVS	29H	3A2EH	POS	11H	39FEH
CVD	2AH	3A30H	PEEK	17H	3A0AH
COS	0CH	39F4H	PDL	24H	3A24H
CHR$	16H	3A08H	PAD	25H	3A26H
DSKF	26H	3A28H	RIGHT$	02H	39E0H
EXP	0BH	39F2H	RND	08H	39ECH
EOF	2BH	3A32H	SGN	04H	39E4H
FRE	0FH	39FAH	SQR	07H	39EAH
FIX	21H	3A1EH	SIN	09H	39EEH
FPOS	27H	3A2AH	STR$	13H	3A02H
HEX$	1BH	3A12H	SPACE$	19H	3A0EH
INT	05H	39E6H	STICK	22H	3A20H
INP	10H	39FCH	STRIG	23H	3A22H
LPOS	1CH	3A14H	TAN	0DH	39F6H
LOG	0AH	39F0H	VAL	14H	3A04H
LOC	2CH	3A34H	VPEEK	18H	3A0CH

No início desta seção, foi dito que a linha BASIC devia estar preferencialmente na forma tokenizada. Entretanto, é possível utilizá-la na forma ASCII para facilitar. O único cuidado, nesse caso, é substituir dez caracteres-chave pelas tokens respectivas, podendo o restante do texto BASIC estar na forma ASCII. Esses caracteres são:

+ F1H	* F3H	^ F5H	' E6H	= EFH
- F2H	/ F4H	\ FCH	> EEH	< F0H

Assim, por exemplo, uma linha de texto BASIC tipo:

```
LINE (10,10)-(50,50),1
```

deve ser colocada na linha em código de máquina da seguinte forma:

```
DEFB '(10,10)'.0F2H.'(50,50),1',00H
```

Marca de fim de linha
Token do caractere "-"
O par HL deve apontar aqui

Se algum dos caracteres acima vier entre aspas no texto BASIC, como nos comandos DRAW ou PLAY, deverá ser mantido em sua forma original. Por exemplo:

PLAY "A-BC+" em assembly ficará:

```
DEFB '"A-BC+"',00H
```

Marca de fim de linha
HL

Para terminar, vamos a um exemplo prático, com a instrução BASIC CIRCLE.

```
          ORG   0C000H
CIRCLE:   EQU   039A4H
INIGRP:   EQU   00072H
CHGET:    EQU   0009FH
CALBAS:   EQU   00159H
          CALL  INIGRP          :tela em SCREEN 2
          LD    HL,LINBAS       :HL aponta p/ falsa linha BASIC
          LD    IX,(CIRCLE)     :IX = endereço de CIRCLE
          CALL  CALBAS          :executa CIRCLE
          CALL  CHGET           :espera tecla ser pressionada
          RET                   :retorna
LINBAS:   DEFB '(128,96),50',00H
```

Capítulo 3

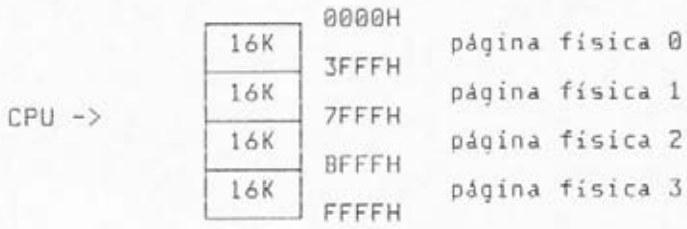
A MEMÓRIA RAM

A CPU Z80 pode acessar diretamente o máximo de apenas 64 Kbytes de memória. Essa quantidade de memória já era insuficiente para várias aplicações mesmo em 1.983 quando foi criado o padrão MSX. Por isso, foi desenvolvido um sistema de slots e páginas que permitia ao Z80 acessar, teoricamente, até o máximo de 1 Megabyte de memória. O sistema de slots e páginas para funcionar como expansão de memória era muito complexo e não chegou a ser utilizado comercialmente.

Em 1.985, com o lançamento do MSX2, foi criado um novo conceito de expansão de memória, a *Memória Mapeada*, de fácil manipulação, que podia ser efetivamente ampliada até 4 Megabytes.

1 - A MEMÓRIA MAPEADA

A memória mapeada usa as portas de I/O do Z80 como complemento ao barramento de endereços. Quatro portas de I/O são usadas, de FCH a FFH, uma para cada *Página Física*. Páginas Físicas são as quatro páginas de 16 Kbytes que podem estar ativas ao mesmo tempo, cada uma em endereços diferentes. Veja a ilustração abaixo:



Assim, para cada página física há uma porta de I/O correspondente:

```
Página física 0  =  porta FCH
Página física 1  =  porta FDH
Página física 2  =  porta FEH
Página física 3  =  porta FFH
```

O valor que pode ser escrito em uma porta do Z80 varia de 0 a 255; assim, pode-se ter até 256 *Páginas Lógicas*. Como cada página tem 16 Kbytes de memória, fazemos 16 Kbytes vezes 256, o que dá um máximo de 4 Megabytes.

No MSX2 é usado um slot com 64 Kbytes de RAM e a memória mapeada deve estar em outro slot. Já do MSX2 Plus em diante, os 64 Kbytes de RAM principal correspondem aos primeiros 64 Kbytes da memória mapeada, ocupando quatro páginas. A seleção inicial das páginas é a seguinte:

```
Página física 0 = página lógica 3 (porta FCH = 3)
Página física 1 = página lógica 2 (porta FDH = 2)
Página física 2 = página lógica 1 (porta FEH = 1)
Página física 3 = página lógica 0 (porta FFH = 0)
```

A troca entre as páginas físicas e lógicas é muito simples. Basta usar uma instrução OUT do Z80 para posicionar a página lógica desejada na página física correspondente. Assim, para a seleção inicial dos 64 Kbytes, as seguintes instruções são usadas:

```
OUT  0FCH,3 :posiciona a pág. lógica 3 na pág. física 0
OUT  0FDH,2 :posiciona a pág. lógica 2 na pág. física 1
OUT  0FEH,1 :posiciona a pág. lógica 1 na pág. física 2
OUT  0FFH,0 :posiciona a pág. lógica 0 na pág. física 3
```

Observe que como as páginas lógicas têm sempre o mesmo número, eventualmente uma página lógica pode estar em *duas ou mais* páginas físicas ao mesmo tempo. Por exemplo, veja as instruções:

```
OUT  0FDH,5
OUT  0FEH,5
```

Elas colocam a página lógica 5 nas páginas físicas 1 e 2.

Observe também que a seleção de slots e páginas físicas tem precedência sobre a seleção de páginas lógicas. Por isso, quando for selecionar uma página lógica, tenha sempre a certeza de que a página física correspondente esteja habilitada.

Normalmente, apenas as páginas físicas 1 e 2 são utilizadas para a seleção de páginas lógicas, uma vez que a página física 0 contém o BIOS e a página física 3 contém a área de trabalho do sistema e não pode ser desligada.

2 - A MEGARAM

Apesar de não ser reconhecida oficialmente pela Microsoft como expansão de memória para o MSX, a Megaram é bastante popular no Brasil.

A Megaram também envolve conceitos de páginas físicas e lógicas, mas sua operação é mais complicada que a da Memória Mapeada.

Cada página lógica da Megaram tem 8 Kbytes e o gerenciamento dessas páginas é feito através da porta 08EH do Z80. Para habilitar a Megaram, devemos primeiro executar a seguinte instrução:

```
OUT  (08EH),A
```

Observe que o valor de A não tem importância. Essa instrução apenas indica à Megaram que ela vai ser utilizada. Observe que cada página lógica da Megaram tem apenas 8 Kbytes; portanto, são necessárias duas páginas lógicas para cada página física. Cada página lógica pode começar em cada um dos seguintes endereços:

```
4000H - 6000H - 8000H - A000H
```

Depois de executada a instrução "OUT (08EH),A", deve-se carregar em A o número desejado da página lógica da Megaram e executar a instrução "LD (xxxxH),A", onde "xxxxH" é o endereço inicial da página lógica na página física. Para colocarmos as pá-

ginas lógicas 0 e 1 da Megaram na página física 1 do micro, devemos executar as seguintes instruções:

```
OUT   (08EH),A    ;habilita a Megaram
LD    A,0         ;seleciona página lógica 0
LD    (04000H),A  ;posiciona pág. lógica 0 no end. 4000H
LD    A,1         ;seleciona página lógica 1
LD    (06000H),A  ;posiciona pág. lógica 1 no end. 6000H
```

Executando essas instruções, as páginas lógicas 0 e 1 da Megaram estarão ocupando a página física 1 do micro, e está pronta para ser lida, mas não para ser escrita. Para que possamos escrever dados na Megaram, devemos executar a instrução "IN A,(08EH)". Veja abaixo as instruções que colocam as páginas 0 e 1 da Megaram na página física 1 do micro e as habilita para leitura e escrita.

```
OUT   (08EH),A
XOR   A
LD    (04000H),A
INC   A
LD    (06000H),A
IN    A,(08EH)
```

Ao ser executada, essa rotina posiciona as páginas lógicas 0 e 1 da Megaram na página física 1 do micro e as habilita para serem lidas e escritas.

2.1 - MEGARAM x MEMÓRIA MAPEADA

Uma dúvida que pode surgir aos programadores é sobre qual expansão de memória usar: a Megaram ou a Memória Mapeada. Como já descrito, a Memória Mapeada é a expansão padrão do MSX. Entretanto, a Megaram é muito popular no Brasil.

Uma solução razoável a essa questão é que os programas desenvolvidos reconheçam as duas expansões. Primeiramente, o programa deve pesquisar a Memória Mapeada, pois suas portas de I/O são padronizadas pela Microsoft. Caso esta não seja encontrada, faz-se a procura pela Megaram. Deve-se procurar primeiro a Memória Mapeada pois a porta de I/O utilizada pela Megaram não é reconhecida oficialmente pela Microsoft, o que poderá causar problemas de incompatibilidade no futuro.

3 - MAPEAMENTO DA RAM

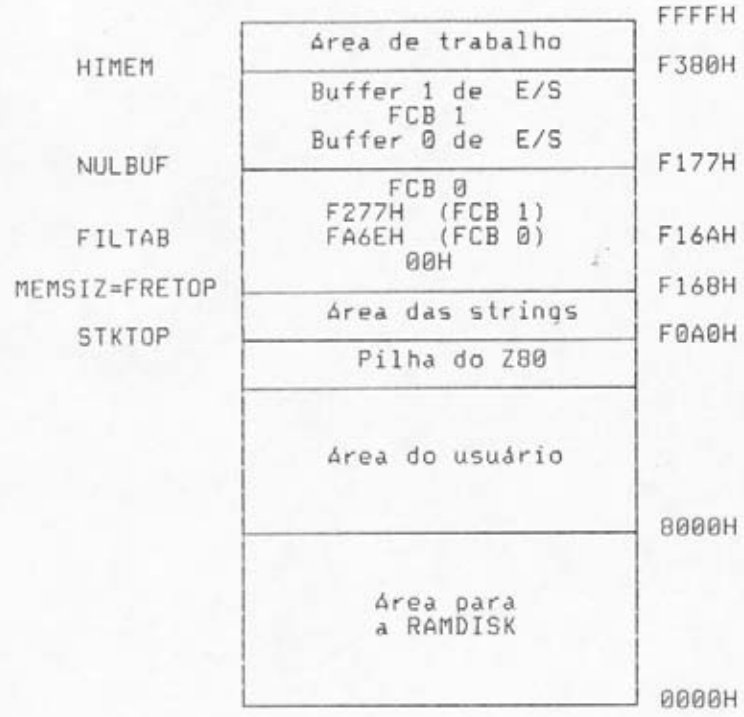
Independente de slots e páginas, há um mapeamento específico para a RAM.

Esse mapeamento situa-se na página física 3, ocupando os endereços mais altos da memória. Embora os endereços inferiores também sejam mapeados, não há problemas de troca entre as páginas 0, 1 e 2, desde que tomados os devidos cuidados, como, por exemplo, não desligar a página onde o programa está sendo executado.

Já a página física 3 jamais deve ser desligada, pois contém a área de trabalho do sistema.

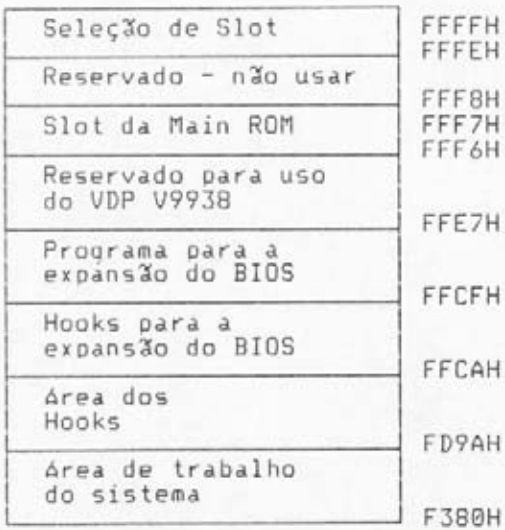
Ao entrar no BASIC, logo após um reset, a RAM é mapea-

da como se segue:



Esse é o mapeamento standard do MSX2, sem unidade de disco. Ao ser instalada, a unidade de disco altera alguns desses endereços. Veja o capítulo sobre o Sistema de Disco.

4 - A ÁREA DE TRABALHO DO SISTEMA



A área de Trabalho do Sistema se situa na página 3 e vai do endereço F380H até FFFFH. O uso dessa área pelo programador deve ser bem controlado, sob pena de alterações indesejáveis nas funções básicas do micro. A área de trabalho é subdividida e mapeada como mostrado na ilustração da página anterior.

A seguir, estão listados todos os valores da área de trabalho. A notação é a seguinte:

LABEL (ENDEREÇO.COMPRIMENTO)

Onde LABEL é o nome da variável de sistema; ENDEREÇO é o endereço onde ela se situa na RAM e COMPRIMENTO é o tamanho da variável em bytes.

4.1 - SUBROTINAS PARA ESCRITA, LEITURA E CHAMADAS INTER-SLOT

RDPRIM (F380H.5)
Função: lê em um slot primário.

WRPRIM (F385H.5)
Função: escreve em um slot primário.

CLPRIM (F38CH.14)
Função: chama um endereço em um slot primário.

4.2 - ENDEREÇOS PARA A FUNÇÃO USR E MODOS DE TEXTO

USRTAB (F39AH.20)
Valor inicial: FCERR
Conteúdo: São dez variáveis de sistema de dois bytes cada; apontam para o endereço de partida de um programa assembly a ser chamado pela função USR do BASIC (0 a 9). O valor inicial aponta para a rotina do gerador de erro.

LINL40 (F3AEH.1)
Valor inicial: 39
Conteúdo: Largura da tela no modo texto SCREEN 0.

LINL32 (F3AFH.1)
Valor inicial: 29
Conteúdo: Largura da tela no modo texto SCREEN 1.

LINLEN (F3B0H.1)
Valor inicial: 39
Conteúdo: Largura atual da tela de texto.

CRTCNT (F3B1H.1)
Valor inicial: 24
Conteúdo: Número de linhas nos modos de texto.

CLMSLT (F3B2H.1)
Valor inicial: 14
Conteúdo: Localização horizontal no caso de itens divididos por virgula no comando PRINT.

4.3 - VALORES DE INICIALIZAÇÃO DOS MODOS DE TELA.

SCREEN 0:

TXTNAM (F3B3H,2)
  Valor inicial: 0000H
  Conteúdo: Endereço da tabela de nomes dos padrões.

TXTCOL (F3B5H,2) - Sem significado.

TXTCGP (F3B7H,2)
  Valor inicial: 0800H
  Conteúdo: Valor inicial da tabela geradora de padrões.
  Observação: Nesta variável reside o único bug, ou erro, encontrado nos micros MSX2. Quando na SCREEN 0 for dado o comando WIDTH até 40, o valor estará correto. Porém, se o comando WIDTH for de 41 até 80, o valor correto será de 1000H, mas esta variável continuará marcando 0800H. Neste caso, quando for trabalhar com um programa ASSEMBLY a partir do BASIC, use uma instrução tipo ADD HL,HL, por exemplo, para corrigir o valor. Nos micros MSX2+ e MSX turbo R, o valor inicial dessa variável é de 0000H, de modo que a instrução mostrada não afeta a compatibilidade, a despeito desse bug não existir nesses micros.

TXTATR (F3B9H,2) - Sem significado.

TXTPAT (F3BBH,2) - Sem significado.

SCREEN 1:

T32NAM (F3BDH,2)
  Valor inicial: 1800H
  Conteúdo: Endereço da tabela de nomes dos padrões.

T32COL (F3BFH,2)
  Valor inicial: 2000H
  Conteúdo: Endereço da tabela de cores.

T32CGP (F3C1H,2)
  Valor inicial: 0000H
  Conteúdo: Endereço da tabela geradora de padrões.

T32ATR (F3C3H,2)
  Valor inicial: 1B00H
  Conteúdo: Endereço da tabela de atributos dos sprites.

T32PAT (F3C5H,2)
  Valor inicial: 3800H
  Conteúdo: Endereço da tabela geradora dos sprites.

SCREEN 2:

GRPNAM (F3C7H,2)
  Valor inicial: 1800H
  Conteúdo: Endereço da tabela de nomes dos padrões.

GRPCOL (F3C9H,2)
  Valor inicial: 2000H
  Conteúdo: Endereço da tabela de cores.

GRPCGP (F3CBH,2)
  Valor inicial: 0000H
  Conteúdo: Endereço da tabela geradora de padrões.

GRPATR (F3CDH,2)
  Valor inicial: 1B00H
  Conteúdo: Endereço da tabela de atributos dos sprites.

GRPPAT (F3CFH,2)
  Valor inicial: 3800H
  Conteúdo: Endereço da tabela geradora dos sprites.

SCREEN 3:

MLTNAM (F3D1H,2)
  Valor inicial: 0800H
  Conteúdo: Endereço da tabela de nomes dos padrões.

MLTCOL (F3D3H,2) - Sem significado.

MLTCGP (F3D5H,2)
  Valor inicial: 0000H
  Conteúdo: Endereço da tabela geradora de padrões.

MLTATR (F3D7H,2)
  Valor inicial: 1B00H
  Conteúdo: Endereço da tabela de atributos dos sprites.

MLTPAT (F3D9H,2)
  Valor inicial: 3800H
  Conteúdo: Endereço da tabela geradora dos sprites.

4.4 - OUTROS VALORES PARA TELA

CLIKSW (F3DBH,1)
  Valor inicial: 1
  Conteúdo: Liga/desliga click das teclas (0=desliga; outro valor, liga). Pode ser modificada pelo comando SCREEN.

CSRY (F3DCH,1)
  Valor inicial: 1
  Conteúdo: Coordenada Y do cursor.

CSRX (F3DDH,1)
  Valor inicial: 1
  Conteúdo: Coordenada X do cursor.

CNSDFG (F3DEH,1)
  Valor inicial: 0
  Conteúdo: Liga/desliga apresentação das teclas de função (0=liga; outro valor, desliga). Pode ser modificada pelos comandos KEY ON/OFF.

4.5 - ÁREA DOS REGISTRADORES DO VDP

RG0SAV (F3DFH,1)
  Valor inicial: 00H

RG1SAV (F3E0H,1)
  Valor inicial: E0H

RG2SAV (F3E1H,1)
  Valor inicial: 00H

RG3SAV (F3E2H,1)
  Valor inicial: 00H

RG4SAV (F3E3H,1)
  Valor inicial: 00H

RG5SAV (F3E4H,1)
  Valor inicial: 00H

RG6SAV (F3E5H,1)
  Valor inicial: 00H

RG7SAV (F3E6H,1)
  Valor inicial: 00H

STATFL (F3E7H,1)
  Valor inicial: 00H
  Conteúdo: Armazena o registrador de status do VDP. No caso de MSX2 ou superior, é o conteúdo do registrador de status S#0.

TRGFLG (F3E8H,1)
  Valor inicial: FFH
  Conteúdo: Armazena o estado dos botões dos joysticks.

FORCLR (F3E9H,1)
  Valor inicial: 15
  Conteúdo: Cor de frente e dos caracteres. Pode ser alterada pelo comando COLOR.

BAKCLR (F3EAH,1)
  Valor inicial: 4
  Conteúdo: Cor de fundo. Pode ser alterada pelo comando COLOR.

BDRCLR (F3EBH,1)
  Valor inicial: 7
  Conteúdo: Cor da borda. Pode ser alterada pelo comando COLOR.

MAXUPD (F3ECH,3)
  Valor inicial: JP 0000H (C3H, 00H, 00H)
  Conteúdo: Usada internamente pelo comando CIRCLE.

MINUPD (F3EFH,3)
  Valor inicial: JP 0000H (C3H, 00H, 00H)
  Conteúdo: Usada internamente pelo comando CIRCLE.

ATRBYT (F3F2H,1)
  Valor inicial: 15
  Conteúdo: Código de cor usado para gráficos.

4.6 - ÁREA USADA PELO COMANDO PLAY

QUEUES (F3F3H,2)
  Valor inicial: QUETAB (F959H)
  Conteúdo: Apontador para a fila de execução do comando PLAY.

FRCNEW (F3F5H,1)
  Valor inicial: 255
  Conteúdo: Usada internamente pelo interpretador BASIC.

4.7 - ÁREA USADA PARA O TECLADO

SCNCNT (F3F6H,1)
  Valor inicial: 1
  Conteúdo: Intervalo para a varredura das teclas.

REPCNT (F3F7H,1)
  Valor inicial: 50
  Conteúdo: Tempo de atrazo para o início da função de auto-repetição das teclas.

PUTPNT (F3F8H,2)
  Valor inicial: KEYBUF (FBF0H)
  Conteúdo: Aponta para o endereço de escrita do buffer de teclado.

GETPNT (F3FAH,2)
  Valor inicial: KEYBUF (FBF0H)
  Conteúdo: Aponta para o endereço de leitura do buffer de teclado.

4.8 - ÁREA USADA PELO COMANDO CIRCLE

ASPCT1 (F40BH,2)
  Conteúdo: 256/relação de aspecto. Pode ser alterada pelo comando SCREEN para uso do comando CIRCLE.

ASPCT2 (F40DH,2)
  Conteúdo: 256*relação de aspecto. Pode ser alterada pelo comando SCREEN para uso do comando CIRCLE.

4.9 - ÁREA USADA INTERNAMENTE PELO BASIC

ENDPRG (F40FH,5)
  Valor inicial: ":"; 00H; 00H; 00H; 00H.
  Conteúdo: Falso fim de linha de programa para os comandos RESUME e NEXT.

ERRFLG (F414H,1)
  Conteúdo: Área para salvar o número de erro.

LPTPOS (F415H,1)
  Valor inicial: 00H
  Conteúdo: Armazena posição atual da cabeça da impressora.

PRTFLG (F416H,1)
  Conteúdo: Flag para selecionar saida para tela ou impressora (0=tela; outro valor, impressora).

NTMSXP (F417H,1)
  Conteúdo: Flag para selecionar o tipo de impressora (0= impressora padrão MSX; outro valor, impressora não MSX). Pode ser alterada pelo comando SCREEN.

RAWPRT (F418H,1)
  Conteúdo: Flag para determinar se os caracteres gráficos de controle serão modificados ao serem enviados para a impressora (0=modifica; outro valor, não modifica).

VLZADR (F419H,2)
  Conteúdo: Endereço do caractere para a função VAL.

VLZDAT (F41BH,1)
  Conteúdo: Caractere que deve ser substituido por 0 pela função VAL.

CURLIN (F41CH,2)
  Conteúdo: Número de linha atual do interpretador BASIC. O valor FFFFH indica modo direto.

KBFMIN (F41EH,1)
  Valor inicial: ":"
  Conteúdo: Esse byte é um prefixo fictício para o texto tokenizado contido em KBUF.

KBUF (F41FH,318)
  Conteúdo: Esse buffer guarda a linha BASIC tokenizada coletada pelo interpretador.

BUFMIN (F55DH,1)
  Valor inicial: ","
  Conteúdo: Usada pelo comando INPUT.

BUF (F55EH,258)
  Conteúdo: Esse buffer guarda, no formato ASCII, os caracteres coletados diretamente pelo teclado.

ENDBUF (F660H,1)
  Conteúdo: Byte para prevenir overflow em BUF (F55EH).

TTYPOS (F661H,1)
  Conteúdo: Usada pelo comando PRINT para guardar a posição virtual do cursor.

DIMFLG (F662H,1)
  Conteúdo: Usada internamente pelo comando DIM.

VALTYP (F663H,1)
  Conteúdo: Guarda o tipo de variável contida em DAC (F3F6H) 2=inteira; 3=string; 4=precisão simples; 8=precisão dupla.

DORES (F664H,1)
  Conteúdo: Usada internamente pelo comando DATA para manter o texto no formato ASCII.

DONUM (F665H,1)
  Conteúdo: Flag usada internamente pelo BASIC.

CONTXT (F666H,2)
  Conteúdo: Armazena o endereço do texto usado pela rotina CHRGTR.

CONSAV (F668H,1)
  Conteúdo: Armazena a token de uma constante numérica; usada pela rotina GHRGTR.

CONTYP (F669H,1)
  Conteúdo: Armazena o tipo de uma constante numérica encontrada no texto de programa BASIC. é usada pela rotina CHRGTR.

CONLO (F66AH,8)
  Conteúdo: Armazena uma constante numérica usada pela rotina CHRGTR.

MEMSIZ (F672H,2)
  Conteúdo: Endereço mais alto de memória que pode ser usado pelo BASIC.

STKTOP (F674H,2)
  Conteúdo: Endereço do topo da pilha do Z80. Usada internamente pelo BASIC.

TXTTAB (F676H,2)
  Valor inicial: 8000H
  Conteúdo: Endereço inicial da área de texto BASIC.

TEMPPT (F678H,2)
  Valor inicial: TEMPST (F67AH)
  Conteúdo: Armazena o endereço da próximna posição livre em TEMPST.

TEMPST (F67AH,30)
  Conteúdo: Buffer utilizado para armazenar os descritores de strings.

DSCTMP (F698H,3)
  Conteúdo: Armazena o descritor de uma string durante o processamento.

FRETOP (F69BH,2)
  Conteúdo: Armazena o endereço da próxima posição livre na área de strings.

TEMP3 (F69DH,2)
  Conteúdo: Usada internamente pela função USR.

TEMP8 (F69FH,2)
  Conteúdo: Usada internamente pelo interpretador.

ENDFOR (F6A1H,2)
  Conteúdo: Armazena endereço para o comando FOR.

DATLIN (F6A3H,2)
  Conteúdo: Número de linha do comando DATA para uso do comando READ.

SUBFLG (F6A5H,1)
  Conteúdo: Flag para o array da função USR.

FLGINP (F6A6H,1)
  Conteúdo: Flag usada pelos comandos INPUT e READ (0= INPUT; outro valor, READ).

TEMP (F67AH,2)
  Conteúdo: Usada internamente pelo interpretador.

PTRFLG (F6A9H,1)
  Conteúdo: Usada internamente pelo interpretador para conversão dos números de linha em apontadores (0=operando não convertido; outro valor, operando convertido).

AUTFLG (F6AAH,1)
  Conteúdo: Flag para o comando AUTO (0=comando AUTO inativo; outro valor, comando AUTO ativo).

AUTLIN (F6ABH,2)
  Conteúdo: Número da última linha BASIC entrada.

AUTINC (F6ADH,2)
  Valor inicial: 10
  Conteúdo: Valor de incremento para a função AUTO.

SAVTXT (F6AFH,2)
  Conteúdo: Armazena o endereço atual do texto BASIC durante a execução.

SAVSTK (F6B1H,2)
  Conteúdo: Armazena o endereço atual da pilha do Z80. Usada pelo manipulador de erro e pela instrução RESUME.

ERRLIN (F6B3H,2)
  Conteúdo: Número de linha BASIC onde ocorreu algum erro.

DOT (F6B5H,2)
  Conteúdo: último número de linha durante o processamento. Usada internamente pelo interpretador e pelo manipulador de erro.

ERRTXT (F6B7H,2)
  Conteúdo: Endereço do texto BASIC onde ocorreu algum erro. Usada pelo comando RESUME.

ONELIN (F6B9H,2)
  Conteúdo: Endereço da linha de programa que deve ser executada ao ocorrer algum erro. Setada pelo comando ON ERROR GOTO.

ONEFLG (F6BBH,1)
  Conteúdo: Flag para indicar execução de rotina de erro (0=não executando; outro valor, rotina em execução).

TEMP2 (F6BCH,2)
  Conteúdo: Usada internamente pelo interpretador.

OLDLIN (F6BEH,2)
  Conteúdo: Armazena a última linha executada pelo programa. é atualizada pelos comandos END e STOP para ser usada pelo comando CONT.

OLDTXT (F6C0H,2)
  Conteúdo: Armazena o endereço da última instrução do texto BASIC.

VARTAB (F6C2H,2)
  Conteúdo: Endereço do primeiro byte da área de armazenamento das variáveis do BASIC.

ARYTAB (F6C4H,2)
  Conteúdo: Endereço do primeiro byte da área de armazenamento das matrizes do BASIC.

STREND (F6C6H,2)
  Conteúdo: Endereço do primeiro byte após a área de arzenanamento das matrizes, variáveis ou texto BASIC.

DATPTR (F6C8H,2)
  Conteúdo: Endereço do comando DATA atual para uso do comando READ.

DEFTBL (F6CAH,26)
  Conteúdo: Área de armazenamento do tipo de variável por nomes em ordem alfabética. Podem ser alteradas pelo grupo de comandos "DEF xxx".

4.10 - ÁREA PARA AS FUNÇÕES DO USUÁRIO

PRMSTK (F6E4H,2)
  Conteúdo: Definição prévia do bloco na pilha do Z80.

PRMLEN (F6E6H,2)
  Conteúdo: Comprimento do bloco de parâmetro "FN" atual em PARM1.

PARM1 (F6E8H,100)
  Conteúdo: Buffer para armazenamento das variáveis da função "FN" que está sendo avaliada.

PRMPRV (F74CH,2)
  Valor inicial: PRMSTK (F6E4H)
  Conteúdo: Endereço do bloco de parâmetro "FN" anterior.

PRMLN2 (F74EH,2)
  Conteúdo: Comprimento do bloco de parâmetros "FN" que está sendo montado em PARM2.

PARM2 (F750H,100)
  Conteúdo: Buffer usado para as variáveis da função "FN" atual.

PRMFLG (F7B4H,1)
  Conteúdo: Flag para indicar quando PARM1 está sendo procurada.

ARYTA2 (F7B5H,2)
  Conteúdo: último endereço para procura de variável.

NOFUNS (F7B7H,1)
  Conteúdo: Flag para indicar à função "FN" a existência de variáveis locais (0=não há variáveis; outro valor, há variáveis).

TEMP9 (F7B8H,2)
  Conteúdo: Usada internamente pelo interpretador.

FUNACT (F7BAH,2)
  Conteúdo: Número de funções "FN" atualmente ativas.

SWPTMP (F7BCH,8)
  Conteúdo: Buffer utilizado para conter o primeiro operando de um comando SWAP.

TRCFLG (F7C4H,1)
  Conteúdo: Flag para o comando TRACE (0=TRACE OFF, outro valor, TRACE ON).

4.11 - ÁREA PARA O MATH-PACK

FBUFFR (F7C5H,43)
  Conteúdo: Usado internamente pelo MATH-PACK.

DECTMP (F7F0H,2)
  Conteúdo: Usado para transformar um número inteiro em um número de ponto flutuante.

DECTM2 (F7F2H,2)
  Conteúdo: Usada internamente pela rotina de divisão.

DECCNT (F7F4H,1)
  Conteúdo: Usada internamente pela rotina de divisão.

DAC (F7F6H,16)
  Conteúdo: Acumulador primário que contém um número durante uma operação matemática.

HOLD8 (F806H,48)
  Conteúdo: Área de armazenamento para a multiplicação decimal.

HOLD2 (F836H,8)
  Conteúdo: Usada internamente pelo MATH-PACK.

HOLD (F83EH,8)
  Conteúdo: Usada internamente pelo MATH-PACK.

ARG (F847H,16)
Conteúdo: Acumulador secundário que contém o número a ser calculado com DAC (F7F6H).

RNDX (F857H,8)
Conteúdo: Armazena o último número aleatório de dupla precisão. Usada pela função RND.

4.12 - ÁREA DE DADOS DO INTERPRETADOR BASIC

MAXFIL (F85FH,1)
Conteúdo: Número de buffers de I/O existentes. Pode ser alterada pela instrução MAXFILES.

FILTAB (F860H,2)
Conteúdo: Endereço inicial da área de dados dos arquivos.

NULBUF (F862H,2)
Conteúdo: Aponta para o buffer usado pelos comandos SAVE e LOAD.

PTRFIL (F864H,2)
Conteúdo: Endereço dos dados do arquivo atualmente ativo.

RUNFLG (F866H,0)
Conteúdo: Não-zero, se algum programa foi carregado e executado. Usada pelo operando ",R" do comando LOAD.

FILNAM (F866H,11)
Conteúdo: Área para armazenamento de um nome de arquivo.

FILNM2 (F871H,11)
Conteúdo: Área para armazenamento de um nome de arquivo para ser comparado com FILNAM.

NLONLY (F87CH,1)
Conteúdo: Flag para indicar se um programa está sendo carregado ou não (0=programa não está sendo carregado; outro valor, programa está sendo carregado).

SAVEND (F87DH,2)
Conteúdo: Usada pelo comando BSAVE para conter o endereço final do programa assembly que deve ser salvo.

FNKSTR (F87FH,160)
Conteúdo: Área reservada para armazenar o conteúdo das teclas de função (16 caracteres x 10 posições).

CGPNT (F91FH,3)
Conteúdo: Endereço da fonte de caracteres. O primeiro byte é o ID do slot e os outros dois o endereço.

NAMBAS (F922H,2)
Conteúdo: Endereço da tabela de nomes no modo texto atual.

CGPBAS (F924H,2)
Conteúdo: Endereço da tabela geradora de padrões no modo texto atual.

PATBAS (F926H,2)
Conteúdo: Endereço atual da tabela geradora de sprites.

ATRBAS (F928H,2)
Conteúdo: Endereço atual da tabela de atributos dos sprites.

CLOC (F92AH,2)
Conteúdo: Usada internamente pelas rotinas gráficas.

CMASK (F92CH,1)
Conteúdo: Usada internamente pelas rotinas gráficas.

MINDEL (F92DH,2)
Conteúdo: Usada internamente pelo comando LINE.

MAXDEL (F92FH,2)
Conteúdo: Usada internamente pelo comando LINE.

4.13 - AREA DE DADOS PARA O COMANDO CIRCLE

ASPECT (F391H,2)
Conteúdo: Armazena a relação de aspecto usada pelo comando CIRCLE.

CENCNT (F933H,2)
Conteúdo: Usada internamente pelo comando CIRCLE.

CLINEF (F935H,1)
Conteúdo: Flag usada para indicar o desenho de uma linha a partir do centro da circunferência.

CNPNTS (F936H,2)
Conteúdo: Número de pontos dentro de um segmento de 45 graus da circunferência.

CPLOTF (F938H,1)
Conteúdo: Usada internamente pelo comando CIRCLE.

CPCNT (F939H,2)
Conteúdo: Coordenada Y dentro do segmento atual de 45 graus da circunferência.

CPCNT8 (F93BH,2)
Conteúdo: Usada internamente pelo comando CIRCLE.

CPCSUM (F93DH,2)
Conteúdo: Contador da computação de pontos do comando CIRCLE.

CSTCNT (F93FH,2)
Conteúdo: Contém a contagem de pontos inicial do comando CIRCLE.

CSCLXY (F941H,1)
  Conteúdo: Flag para indicar em qual direção a compressão elíptica deve ser feita.

CSAVEA (F942H,2)
  Conteúdo: Área reservada para ADVGRP.

CSAVEM (F944H,1)
  Conteúdo: Área reservada para ADVGRP.

CXOFF (F945H,2)
  Conteúdo: Coordenada X a partir do centro da circunferência.

CYOFF (F947H,2)
  Conteúdo: Coordenada Y a partir do centro da circunferência.

4.14 - ÁREA USADA PELO COMANDO PAINT

LOHMSK (F949H,1)
  Conteúdo: Usada internamente pelo comando PAINT.

LOHDIR (F94AH,1)
  Conteúdo: Usada internamente pelo comando PAINT.

LOHADR (F94BH,2)
  Conteúdo: Usada internamente pelo comando PAINT.

LOHCNT (F94DH,2)
  Conteúdo: Usada internamente pelo comando PAINT.

SKPCNT (F94FH,2)
  Conteúdo: Contador de salto.

MOVCNT (F951H,2)
  Conteúdo: Contador de movimento.

PDIREC (F953H,1)
  Conteúdo: Direção de pintura: 40H, para baixo; C0H, para cima; 00H, terminar.

LFPROG (F954H,1)
  Conteúdo: Flag usada pelo comando PAINT para indicar progresso à esquerda (0=não houve progresso; outro valor, houve progresso).

RTPROG (F955H,1)
  Conteúdo: Flag usada pelo comando PAINT para indicar progresso à direita (0=não houve progresso; outro valor, houve progresso).

4.15 - ÁREA USADA PELO COMANDO PLAY

MCLTAB (F956H,2)
  Conteúdo: Endereço da tabela de comando a ser usada pelos macro-comandos PLAY e DRAW.

MCLFLG (F958H,1)
  Conteúdo: Flag para indicar qual comando está sendo processado (0=DRAW; não zero, PLAY).

QUETAB (F959H,24)
  Conteúdo: Esta tabela contém os dados para as três filas musicais e para a fila RS232C, resevando seis bytes para cada uma.
+0: posição para colocar
+1: posição para pegar
+2: indicação de devolução
+3: tamanho do buffer na fila
+4: endereço do buffer na fila (high)
+5: endereço do buffer na fila (low)

QUEBAK (F971H,4)
  Conteúdo: Usada por BCKQ.

VOICAQ (F975H,128)
  Conteúdo: Fila da voz A.

VOICBQ (F975H,128)
  Conteúdo: Fila da voz B.

VOICCQ (FA75H,128)
  Conteúdo: Fila da voz C.

4.16 - ÁREA ADICIONADA PARA O MSX2

DPPAGE (FAF5H,1)
  Conteúdo: Página de vídeo que está atualmente sendo apresentada.

ACPAGE (FAF6H,1)
  Conteúdo: Página de vídeo ativa para receber comandos.

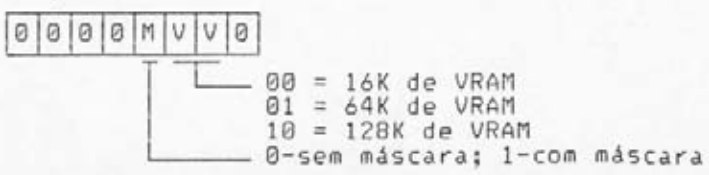
AVCSAV (FAF7H,1)
  Conteúdo: Reservada pela porta de controle AV.

EXBRSA (FAF8H,1)
  Conteúdo: Slot da SUBROM.

CHRCNT (FAF9H,1)
  Conteúdo: Contador de caracteres no buffer. Usada para transição Roman-Kana (0, 1 ou 2).

ROMA (FAFAH,2)
  Conteúdo: Área para armazenar o caractere do buffer para a transição Roman-Kana (somente na versão japonesa).

MODE (FAFCH,1)
  Conteúdo: Flag de modo e tamanho da VRAM:



NORUSE (FAFDH,1) - Sem significado.

XSAVE (FAFEH,2)
Conteúdo:

L	0	0	0	0	0	0	0	X	X	X	X	X	X	X	X

YSAVE (FB00H,2)
Conteúdo:

X	0	0	0	0	0	0	0	Y	Y	Y	Y	Y	Y	Y	Y

L=1, Requisição de interrupção da lightpen.
0000000 - Sem significado.
XXXXXXXXX = Coordenada X
YYYYYYYY = Coordenada Y

LOGOPR (FB02H,1)
Conteúdo: Código de operação lógica para o VDP.

4.17 - ÁREA USADA PELA RS232C

RSTMP (FB03H,50)
Conteúdo: Área de trabalho da RS232C ou disco.

TOCNT (FB03H,1)
Conteúdo: Usada internamente pela RS232C.

RSFCB (FB04H,2)
Conteúdo: Endereço da RS232C.

MEXBIH (FB07H,5)
Conteúdo: FB07H+0: RST 030H
+1: Byte de ID do slot
+2: Endereço (low)
+3: Endereço (high)
+4: RET

OLDSTT (FB0CH,5)
Conteúdo: FB0CH+0: RST 030H
+1: Byte de ID do slot
+2: Endereço (low)
+3: Endereço (high)
+4: RET

OLDINT (FB12H,5)
Conteúdo: FB12H+0: RST 030H
+1: Byte de ID do slot
+2: Endereço (low)
+3: Endereço (high)
+4: RET

DEVNUM (FB17H,1)
Conteúdo: Usada internamente pela RS232C

DATCNT (FB18H,3)
Conteúdo: FB18H+0: Byte de dados
+1: Apontador
+2: Apontador

ERRORS (FB1BH,1)
Conteúdo: Usada internamente pela RS232C.

FLAGS (FB1CH,1)
Conteúdo: Usada internamente pela RS232C.

ESTBLS (FB1DH,1)
Conteúdo: Usada internamente pela RS232C.

COMMSK (FB1EH,1)
Conteúdo: Usada internamente pela RS232C.

LSTCOM (FB1FH,1)
Conteúdo: Usada internamente pela RS232C.

LSTMOD (FB20H,1)
Conteúdo: Usada internamente pela RS232C.

4.18 - ÁREA USADA PELO DOS

FB21H A FB34H - Reservada.
Conteúdo: Usada internamente pelo DOS.

4.19 - ÁREA USADA PELO COMANDO PLAY

PRSCNT (FB35H,1)
Conteúdo: Usada internamente pelo comando PLAY para contar o número de operandos completados. O bit 7 será ligado após cada um dos três operandos serem analizados.

SAVSP (FB36H,2)
Conteúdo: Salva o valor do registrador SP antes da execução do comando PLAY.

VOICEN (FB38H,1)
Conteúdo: Número da voz que está sendo atualmente processada (0, 1 ou 2).

SAVVOL (FB39H,2)
Conteúdo: Salva o volume durante a geração de uma pausa.

MCLLEN (FB3BH,1)
Conteúdo: Usada internamente pelo comando PLAY.

MCLPTR (FB3CH,2)
Conteúdo: Endereço do operando que está sendo analisado.

QUEUEN (FB3EH,1)
Conteúdo: Utilizada pelo manipulador de interrupção para conter o número da fila musical que está sendo atualmente processada.

MUSICF (FB3FH,1)
Conteúdo: Flag para indicar quais filas musicais serão utilizadas.

PLYCNT (FB40H,1)
Conteúdo: Número de seqüências do comando PLAY armazenados na fila.

OFFSET PARA O BUFFER DE PARÂMETROS DO COMANDO PLAY

METREX (+00,2) - Contador de duração
VCXLEN (+02,1) - MCLLEN para a voz respectiva
VCXPTR (+03,2) - MCLPTR para a voz respectiva
VCXSTP (+05,2) - Endereço de dados na pilha
QLENGX (+07,1) - Número de bytes da fila respectiva
NTICSX (+08,2) - Segundo contador
TONPRX (+10,2) - Período do tom
AMPPRX (+12,1) - Volume e envelope
ENVPRX (+13,2) - Período do envelope
OCTAVX (+15,1) - Oitava
NOTELX (+16,1) - Comprimento do tom
TEMPOX (+17,1) - Tempo
VOLUMX (+18,1) - Volume
ENVLPX (+19,14) - Forma de onda do envelope
MCLSTX (+32,3) - Reservado para a pilha
MCLSEX (+35,1) - Inicialização da pilha
VCBSIZ (+36,1) - Tamanho do buffer de parâmetros

AREA DE DADOS PARA O BUFFER DE PARÂMETROS

VCBA (FB41H,37)
  Conteúdo: Parâmetros para a voz A.

VCBB (FB66H,37)
  Conteúdo: Parâmetros para a voz B.

VCBC (FB8BH,37)
  Conteúdo: Parâmetros para a voz C.

4.20 - ÁREA DE DADOS GERAIS

ENSTOP (FBB0H,1)
  Conteúdo: Flag para habilitar uma saida forçada para o interpretador ao detectar as teclas CTRL+SHIFT+GRAPH+CODE pressionadas juntas (0=desabilitada; outro valor, habilitada).

BASROM (FBB1H,1)
  Conteúdo: Indica a localização do texto BASIC (0=RAM; outro valor, em ROM).

LINTTB (FBB2H,24)
  Conteúdo: São 24 flags para indicar se cada uma das linhas de tela de texto avançou para a linha seseguinte (0=avançou; outro valor, não avançou).

FSTPOS (FBCAH,2)
  Conteúdo: Primeira localização do caracter coletado pela rotina INLIN (00B1H) do BIOS.

CODSAV (FBCCH,1)
  Conteúdo: Contém o caractere de tela substituido pelo cursor de texto.

FNKSW1 (FBCDH,1)
  Conteúdo: Flag para indicar quais teclas de função são mostradas quando habilitadas por KEY ON (1=F1 a F5; 0=F6 a F10).

FNKFLG (FBCEH,10)
  Conteúdo: Flags para habilitar, inibir ou paralizar a execução de uma linha definida pelo comando ON KEY GOSUB. São modificadas por KEY(n) ON/OFF/STOP (0=KEY(n) OFF/STOP; 1=KEY(n) ON).

ONGSBF (FBD8H,1)
  Conteúdo: Flag para indicar se algum dispositivo requereu uma interrupção de programa (0=normal; outro valor indica interrupção ativa).

CLIKFL (FBD9H,1)
  Conteúdo: Flag de click das teclas. Usada pelo manipulador de interrupção.

OLDKEY (FBDAH,11)
  Conteúdo: Estado anterior da matriz do teclado.

NEWKEY (FBE5H,11)
  Conteúdo: Novo estado da matriz do teclado.

KEYBUF (FBF0H,40)
  Conteúdo: Buffer circular que contém os caracteres do teclado decodificados.

LINWRK (FC18H,40)
  Conteúdo: Buffer usado pelo BIOS para conter uma linha completa de caracteres da tela.

PATWRK (FC40H,8)
  Conteúdo: Buffer usado pelo BIOS para conter um padrão de caractere 8x8.

BOTTOM (FC48H,2)
  Conteúdo: Endereço mais baixo usado pelo interpretador, normalmente 8000H.

HIMEM (FC4AH,2)
  Conteúdo: Endereço mais alto de RAM disponivel. Pode ser modificado pelo comando CLEAR.

TRPTBL (FC4CH,78)
  Conteúdo: Esta tabela contém o estado atual dos dispositivos de interrupção. Cada dispositivo aloca três bytes na tabela. O primeiro byte contém o estado do dispositivo (bit 0=ligado; bit 1=parado; bit 2=ativo). Os outros dois bytes contêm o endereço da linha de programa a ser executada caso ocorra uma interrupção.
  FC4CH/FC69H (3 x 10 bytes) = ON KEY GOSUB
  FC6AH/FC6CH (3 x 1 byte) = ON STOP GOSUB
  FC6DH/FC6FH (3 x 1 byte) = ON SPRITE GOSUB
  FC70H/FC7EH (3 x 5 bytes) = ON STRIG GOSUB
  FC7FH/FC81H (3 x 1 byte) = ON INTERVAL GOSUB
  FC82H/FC99H - Reservado para expansão

RTYCNT (FC9AH,1)
  Conteúdo: Usada internamente pelo BASIC.

INTFLG (FC9BH,1)
  Conteúdo: Se CTRL+STOP são pressionadas, esta variável é colocada em 03H e o processamento interrompido; se STOP for pressionada, o valor é 04H.

PADY (FC9CH,1)
  Conteúdo: Coordenada Y do paddle.

PADX (FC9DH,1)
  Conteúdo: Coordenada X do paddle.

JIFFY (FC9EH,2)
  Conteúdo: Esta variável é continuamente incrementada pelo manipulador de interrupção. Seu valor pode ser lido ou atribuído pela função TIME. Também é utilizada internamente pelo comando PLAY.

INTVAL (FCA0H,2)
  Conteúdo: Duração do intervalo usado por ON INTERVAL GOSUB.

INTCNT (FCA2H,2)
  Conteúdo: Contador para a instrução ON INTERVAL GOSUB.

GRPHED (FCA6H,1)
  Conteúdo: Flag para o envio de um caractere gráfico (0=normal; 1=caractere gráfico).
  Conteúdo: Área de contagem dos códigos de escape.

INSFLG (FCA8H,1)
  Conteúdo: Flag para indicar o modo de inserção (0=normal; outro valor, modo de inserção)

CSRSW (FCA9H,1)
  Conteúdo: Flag para indicar se o cursor será mostrado (0=não; outro valor, sim). Pode ser modificada pelo comando LOCATE.

CSTYLE (FCAAH,1)
  Conteúdo: Forma do cursor (0=bloco; outro valor, sub-alinhado).

CAPST (FCABH,1)
  Conteúdo: Estado da tecla CAPS LOCK (0=desligada; outro valor, ligada).

KANAST (FCACH,1)
  Conteúdo: Estado da tecla KANA (0=desligada; outro valor,ligada).

KANAMD (FCADH,1)
  Conteúdo: Flag usada apenas em máquinas japonesas.

FLBMEM (FCAEH,1)
  Conteúdo: Flag para indicar carregamento de programa em BASIC (0=está carregando; outro valor, não).

SCRMOD (FCAFH.1)
  Conteúdo: Número do modo de tela atual.

OLDSCR (FCB0H.1)
  Conteúdo: Modo de tela do último modo texto.

BDRATR (FCB2H.1)
  Conteúdo: Código de cor da borda. Usado por PAINT.

GXPOS (FCB3H.2)
  Conteúdo: Coordenada X gráfica.

GYPOS (FCB5H.2)
  Conteúdo: Coordenada Y gráfica.

GRPACX (FCB7H.2)
  Conteúdo: Acumulador gráfico para a coordenada X.

GRPACY (FCB9H.2)
  Conteúdo: Acumulador gráfico para a coordenada Y.

DRWFLG (FCBBH.1)
  Conteúdo: Flag usada pelo comando DRAW.

DRWSCL (FCBCH.1)
  Conteúdo: Fator de escala para o comando DRAW. O valor 0 indica que não será usada a escala.

DRWANG (FCBDH.1)
  Conteúdo: Ângulo para o comando DRAW.

RUNBNF (FCBEH.1)
  Conteúdo: Flag para indicar se o comando BLOAD ou BSAVE está em execução.

SAVENT (FCBFH.2)
  Conteúdo: Endereço inicial para os comandos BSAVE e BLOAD.

EXPTBL (FCC1H.4)
  Conteúdo: Tabela de flags para indicar se os slots primários estão expandidos.

SLTTBL (FCC5H.4)
  Conteúdo: Estes quatro bytes contêm o estado possível dos quatro registradores de slot primário, no caso do slot estar expandido.

SLTATR (FCC9H.64)
  Conteúdo: Tabela de atributos para cada slot.

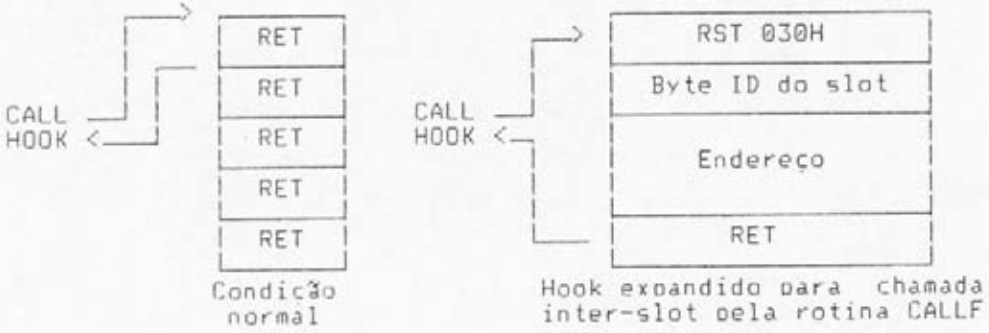
SLTWRK (FD09H.128)
  Conteúdo: Esta tabela aloca dois bytes como área de trabalho para cada página de cada slot.

PROCNM (FD89H.16)
  Conteúdo: Armazena o nome de uma instrução expandida (comando CALL) ou expansão de dispositivo (comando OPEN). Um byte 0 indica o fim do nome.

DEVICE (FD99H.1)
  Conteúdo: Usada para identificar um dispositivo em cartucho.

5 - OS HOOKS

A área de trabalho compreendida entre FD9AH E FFC9H é a área que contém os HOOKS ou GANCHOS. Cada hook é composto por cinco bytes que normalmente são preenchidos com o valor 0C9H (instrução RET). Os hooks são chamados a partir de posições estratégicas do BIOS de modo que as operações do BIOS/Interpretador possam ser modificadas ou ampliadas. Cada hook tem espaço suficiente para uma chamada distante para qualquer slot. Como são preenchidos inicialmente com intruções RET, isso apenas causa um retorno para o BIOS. Entretanto, ele pode ser modificado para chamar uma rotina em qualquer slot.



5.1 - DESCRIÇÃO DOS HOOKS

HKEYI (FD9AH)
  Chamada: Início do manipulador de interrupção.
  Objetivo: Adicionar operações que requeiram interrupção. como a RS232C.

HTIMI (FD9FH)
  Chamada: Início da rotina de processamento de interrupção.
  Objetivo: Adicionar rotinas de manipulação de interrupção.

HCHPU (FDA4H)
  Chamada: Início da rotina CHPUT (saída de caractere).
  Objetivo: Conectar outros dispositivos de console.

HDSPC (FDA9H)
  Chamada: Início do MSXIO DSPCSR (mostra cursor).
  Objetivo: Conectar outros dispositivos de console.

HERAC (FDAEH)
  Chamada: Início do MSXIO ERACSR (apaga cursor).
  Objetivo: Conectar outros dispositivos de console.

HDSPF (FDB3H)
  Chamada: Início da rotina DSPFNK (apresenta teclas de função).
  Objetivo: Conectar outros dispositivos de console.

HERAF (FDB8H)
  Chamada: Início da rotina ERAFNK (apaga teclas de função).
  Objetivo: Conectar outros dispositivos de console.

HTOTE (FDBDH)
  Chamada: Início da rotina TOTEXT (força a tela para modo texto).
  Objetivo: Conectar outros dispositivos de console.

HCHGE (FDC2H)
  Chamada: Início da rotina CHGET (pega um caractere).
  Objetivo: Conectar outros dispositivos de console.

HINIP (FDC7H)
  Chamada: Início do MSXIO INIPAT (inicialização dos padrões de caracteres).
  Objetivo: Usar outra tabela de caracteres.

HKEYC (FDCCH)
  Chamada: Início do MSXIO KEYCOD (decodificador de caracteres do teclado).
  Objetivo: Mudar a configuração do teclado.

HKEYA (FDD1H)
  Chamada: Início da rotina MSXIO NMI (key easy).
  Objetivo: Mudar a configuração do teclado.

HNMI (FDD6H)
  Chamada: Início do MSXIO NMI (interrupções não mascaráveis).
  Objetivo: Ganchos NMI.

HPINL (FDDBH)
  Chamada: Início da rotina PINLIN (pega uma linha).
  Objetivo: Usar outros dispositivos de entrada de console ou outros métodos de entrada.

HQINL (FDE0H)
  Chamada: Início da rotina QINLIN (pega uma linha apresentando "?").
  Objetivo: Usar outros dispositivos de entrada de console ou outros métodos de entrada.

HINLI (FDE5H)
  Chamada: Início da rotina INLIN (pega uma linha).
  Objetivo: Usar outros dispositivos de entrada de console ou outros métodos de entrada.

HONGO (FDEAH)
  Chamada: Início do comando ON GOTO.
  Objetivo: Usar outros dispositivos de manipulação de interrupção).

HDSKO (FDEFH)
  Chamada: Início do comando BASIC "DSKO$".
  Objetivo: Conectar dispositivos de disco.

HSETS (FDF4H)
  Chamada: Início do comando BASIC "SET".
  Objetivo: Conectar dispositivos de disco.

HNAME (FDF9H)
  Chamada: Início do comando BASIC "NAME".
  Objetivo: Conectar dispositivos de disco.

HKILL (FDFEH)
  Chamada: Início do comando BASIC "KILL".
  Objetivo: Conectar dispositivos de disco.

HIPL (FE03H)
  Chamada: Início do comando BASIC "IPL" (Initial Program Loading).
  Objetivo: Conectar dispositivos de disco.

HCOPY (FE08H)
  Chamada: Início do comando BASIC "COPY".
  Objetivo: conectar dispositivos de disco.

HCMD (FE0DH)
  Chamada: Início do comando BASIC "CMD" (Comandos expandidos).
  Objetivo: conectar dispositivos de disco.

HDSKF (FE12H)
  Chamada: Início do comando BASIC "DSKF".
  Objetivo: Conectar dispositivos de disco.

HDSKI (FE17H)
  Chamada: Início do comando BASIC "DSKI$".
  Objetivo: Conectar dispositivos de disco.

HATTR (FE1CH)
  Chamada: Início do manipulador do comando BASIC "ATTR$".
  Objetivo: Conectar dispositivos de disco.

HLSET (FE21H)
  Chamada: Início do manipulador do comando BASIC "LSET".
  Objetivo: conectar dispositivos de disco.

HRSET (FE26H)
  Chamada: Início do manipulador do comando BASIC "RSET".
  Objetivo: Conectar dispositivos de disco.

HFIEL (FE2BH)
  Chamada: Início do manipulador do comando BASIC "FIELD".
  Objetivo: Conectar dispositivos de disco.

HMKI$ (FE30H)
  Chamada: Início do manipulador do comando BASIC "MKI$".
  Objetivo: Conectar dispositivos de disco.

HMKS$ (FE35H)
  Chamada: Início do manipulador do comando BASIC "MKS$".
  Objetivo: Conectar dispositivos de disco.

HMKD$ (FE3AH)
  Chamada: Início do manipulador do comando BASIC "MKD$".
  Objetivo: Conectar dispositivos de disco.

HCVI (FE3FH)
  Chamada: Início do manipulador do comando BASIC "CVI".
  Objetivo: Conectar dispositivos de disco.

HCVS (FE44H)
Chamada: Início do manipulador do comando BASIC "CVS".
Objetivo: Conectar dispositivos de disco.

HCVD (FE49H)
Chamada: Início do manipulador do comando BASIC "CVD".
Objetivo: Conectar dispositivos de disco.

HGETP (FE4EH)
Chamada: Localizar FCB (pegar apontador de arquivo).
Objetivo: Conectar dispositivos de disco.

HSETP (FE53H)
Chamada: Localizar FCB (setar apontador de arquivo).
Objetivo: Conectar dispositivos de disco.

HNOFO (FE58H)
Chamada: Manipulador do comando BASIC "OPEN" (OPEN sem FOR).
Objetivo: Conectar dispositivos de disco.

HNULO (FE5DH)
Chamada: Manipulador do comando BASIC "OPEN" (arquivo aberto não usado).
Objetivo: Conectar dispositivos de disco.

HNTFL (FE62H)
Chamada: Fecha buffer 0 de E/S.
Objetivo: Conectar dispositivos de disco.

HMERG (FE67H)
Chamada: Início do manipulador do comando BASIC "MERGE/LOAD".
Objetivo: Conectar dispositivos de disco.

HSAVE (FE6CH)
Chamada: Início do manipulador do comando BASIC "SAVE".
Objetivo: Conectar dispositivos de disco.

HBINS (FE71H)
Chamada: Início do manipulador do comando BASIC "SAVE" (em binário).
Objetivo: Conectar dispositivos de disco.

HBINL (FE76H)
Chamada: Início do manipulador do comando BASIC "LOAD" (em binário).
Objetivo: Conectar dispositivos de disco.

HFILE (FE7BH)
Chamada: Início do manipulador do comando BASIC "FILES".
Objetivo: Conectar dispositivos de disco.

HDGET (FE80H)
Chamada: Início do manipulador do comando BASIC "GET/PUT".
Objetivo: Conectar dispositivos de disco.

HFILO (FE85H)
Chamada: Manipulador de saída seqüencial.
Objetivo: Conectar dispositivos de disco.

HINDS (FE8AH)
  Chamada: Manipulador de entrada seqüencial.
  Objetivo: Conectar dispositivos de disco.

HRSLF (FE8FH)
  Chamada: Manipulador de seleção prévia de drive.
  Objetivo: Conectar dispositivos de disco.

HSAVD (FE94H)
  Chamada: Reservar disco corrente.
  Objetivo: Conectar dispositivos de disco.

HLOC (FE99H)
  Chamada: Início do manipulador da função BASIC "LOC".
  Objetivo: Conectar dispositivos de disco.

HLOF (FE9EH)
  Chamada: Início do manipulador da função BASIC "LOF".
  Objetivo: Conectar dispositivos de disco.

HEOF (FEA3H)
  Chamada: Início do manipulador da função BASIC "EOF" (end
           of file).
  Objetivo: Conectar dispositivos de disco.

HFPOS (FEA8H)
  Chamada: Início do manipulador da função BASIC "FPOS" (file
           location).
  Objetivo: Conectar dispositivos de disco.

HBAKU (FEADH)
  Chamada: SPCDSK BAKUPT (backup).
  Objetivo: Conectar dispositivos de disco.

HPARD (FEB2H)
  Chamada: Pegar nome de periférico.
  Objetivo: Expandir os nomes lógicos de dispositivos.

HNODE (FEB7H)
  Chamada: Dispositivo sem nome.
  Objetivo: Setar nome default para outro dispositivo.

HPOSD (FEBCH)
  Chamada: SPCDEV POSDSK
  Objetivo: Conectar dispositivos de disco.

HDEVN (FEC1H)
  Chamada: Processar nome do dispositivo.
  Objetivo: Expandir nome lógico do dispositivo.

HGEND (FEC6H)
  Chamada: Despachar função I/O (assign device)
  Objetivo: Expandir nome lógico do dispositivo.

HRUNC (FECBH)
  Chamada: Limpar para comando "RUN".

HCLEA (FED0H)
  Chamada: Limpar para comando "CLEAR".

HLOPD (FED5H)
  Chamada: Seta loop e valor default.

HSTKE (FEDAH)
  Chamada: Reoor pilha.

HISFL (FEDFH)
  Chamada: Início da rotina ISFLIO (I/O de arquivo).

HOUTD (FEE4H)
  Chamada: Início da rotina OUTDO.

HCRDO (FEE9H)
  Chamada: Executar CR+LF (retorno de carro mais avanço de
           linha) para a rotina OUTDO.

HDSKC (FEEEH)
  Chamada: Entrada de atributo de disco.

HDOGR (FEF3H)
  Chamada: Executar operação gráfica.

HPRGE (FEF8H)
  Chamada: Fim de programa.

HERRP (FEFDH)
  Chamada: Manipuladoor de erro (mostra mensagem de erro).

HERRF (FF02H)
  Chamada: Manipulador de arquivo.

HREAD (FF07H)
  Chamada: "Ok" do looo principal.

HMAIN (FF0CH)
  Chamada: Início do looo principal.

HDIRD (FF11H)
  Chamada: Executar comando direto (declaração direta do loop
           principal).

HFINI (FF16H)
  Chamada: Término do looo principal.

HFINE (FF1BH)
  Chamada: Término do loop principal.

HCRUN (FF20H)
  Chamada: Simbolizar.

HCRUS (FF25H)
  Chamada: Simbolizar.

HISRE (FF2AH)
  Chamada: Simbolizar.

HNTFN (FF2FH)
  Chamada: Simbolizar.

HNOTR (FF34H)
Chamada: Simbolizar.

HSNGF (FF39H)
Chamada: Manipulador do comando BASIC "FOR".

HNEWS (FF3EH)
Chamada: Nova declaração do loop principal.

HGONE (FF43H)
Chamada: Executar loop de processamento.

HCHRG (FF48H)
Chamada: Início da rotina CHRGTR.

HRETU (FF4DH)
Chamada: Início do manipulador do comando BASIC "RETURN".

HPTRF (FF52H)
Chamada: Início do manipulador do comando BASIC "PRINT".

HCOMP (FF57H)
Chamada: Manipulador do comando BASIC "PRINT".

HFINP (FF5CH)
Chamada: Manipulador do comando BASIC "PRINT".

HTRMN (FF61H)
Chamada: Erro nos comandos "READ/INPUT".

HFRME (FF66H)
Chamada: Avaliador de expressão.

HNTPL (FF6BH)
Chamada: Avaliador de expressão.

HEVAL (FF70H)
Chamada: Avaliador de fatores.

HOKNO (FF75H)
Chamada: Avaliador de fatores.

HFING (FF7AH)
Chamada: Avaliador de fatores.

HISMI (FF7FH)
Chamada: Executar loop de processamento.

HWIDT (FF84H)
Chamada: Início do manipulador do comando BASIC "WIDTH".

HLIST (FF89H)
Chamada: Início do manipulador do comando BASIC "LIST".

HBUFL (FF8EH)
Chamada: BINTRP BUFLIN (linha de buffer).

HFRQI (FF93H)
Chamada: Converter para inteiro.

HSCNE (FF98H)
  Chamada: Número de linha para apontador.

HFRET (FF9DH)
  Chamada: Liberar descritor.

HPTRG (FFA2H)
  Chamada: Procura apontador de variável.
  Objetivo: Usar outro valor default para as variáveis.

HPHYD (FFA7H)
  Chamada: Início da rotina PHYDIO (physical disk input-output).
  Objetivo: Conectar dispositivos de disco.

HFORM (FFACH)
  Chamada: Início da rotina FORMAT (format disk).
  Objetivo: Conectar dispositivos de disco.

HERRO (FFB1H)
  Chamada: Início do manipulador de erro.
  Objetivo: Manipulação de erros para programas BASIC.

HLPTO (FFB6H)
  Chamada: Início da rotina LPTOUT.
  Objetivo: Usar outros modelos de impressoras.

HLPTS (FFBBH)
  Chamada: Início da rotina LPTSTT.
  Objetivo: Usar outros modelos de impressoras.

HSCRE (FFC0H)
  Chamada: Início do manipulador do comando BASIC "SCREEN".
  Objetivo: Expandir o comando "SCREEN".

HPLAY (FFC5H)
  Chamada: Início do manipulador do comando BASIC "PLAY".
  Objetivo: Expandir o comando "PLAY".

5.2 - HOOKS PARA EXPANSÃO DO BIOS

CALL (FFCAH)
  Chamada: Usado pelo BIOS expandido.

DISINT (FFCFH)
  Chamada: Usado pelo DOS.

ENAINT (FFD4H)
  Chamada: Usado pelo DOS.

A área compreendida entre FFD9H e FFE6H é reservada e não deve ser utilizada pelo programador.

6.3 - ÁREA USADA PELOS VDPs V9938 e V9958

(FFE7H,1) - Cópia do registrador #8 do VDP.

(FFE8H,1) - Cópia do registrador #9 do VDP.

(FFE9H,1) - Cópia do registrador #10 do VDP.

(FFEAH.1) - Cópia do registrador #11 do VDP.

(FFEBH.1) - Cópia do registrador #12 do VDP.

(FFECH.1) - Cópia do registrador #13 do VDP.

(FFEDH.1) - Cópia do registrador #14 do VDP.

(FFEEH.1) - Cópia do registrador #15 do VDP.

(FFEFH.1) - Cópia do registrador #16 do VDP.

(FFF0H.1) - Cópia do registrador #17 do VDP.

(FFF1H.1) - Cópia do registrador #18 do VDP.

(FFF2H.1) - Cópia do registrador #19 do VDP.

(FFF3H.1) - Cópia do registrador #20 do VDP.

(FFF4H.1) - Cópia do registrador #21 do VDP.

(FFF5H.1) - Cópia do registrador #22 do VDP.

(FFF6H.1) - Cópia do registrador #23 do VDP.

(FFF7H.1) - Slot da Main ROM.

FFF8H e FFF9H - Reservados.

(FFFAH.1) - Cópia do registrador #25 do VDP (V9958 somente).

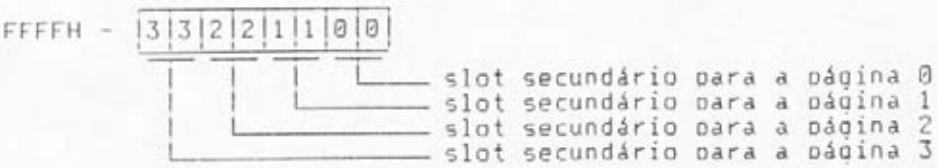
(FFFBH.1) - Cópia do registrador #26 do VDP (V9958 somente).

(FFFCH.1) - Cópia do registrador #27 do VDP (V9958 somente).

FFFDH e FFFEH - Reservados.

6 - REGISTRADOR DE SLOT SECUNDÁRIO

O endereço FFFFH é o *registrador de slot secundário*. apresentando o seguinte conteúdo:



Capítulo 4

O VÍDEO E O VDP

As máquinas MSX, com sua constante evolução, necessitaram de cada vez mais capacidade gráfica. Assim, no MSX1 é usado o VDP TMS9918A, que com apenas 16 cores e poucos recursos representou um fator muito limitante ao processamento gráfico dessas máquinas. Apenas dois anos depois de lançado, surgiu o MSX2 em 1.985, com o novo VDP V9938, totalmente compatível com o TMS9918A. O V9938 adicionou radicais mudanças no vídeo do MSX, possuindo as seguintes características principais:

- Paleta de 512 cores;
- Resolução máxima de 512 x 424 pontos e 16 cores;
- Máximo de 256 cores apresentadas simultaneamente;
- Modos gráficos bit-mapped de fácil manipulação;
- Modo texto de 80 caracteres por linha com recurso de "blink";
- Linha, procura e movimentação de áreas executáveis por hardware;
- Apresenta até 8 sprites na mesma linha horizontal;
- Cada linha de cada sprite pode ter uma cor diferente;
- Endereços podem ser especificados por coordenadas;
- Funções de operação lógica;
- Scroll vertical por hardware;
- Capacidade interna de digitalização;
- Capacidade interna de "superimpose".

Mais tarde, em 1.988, foi lançado o MSX2+, com o novo VDP V9958, compatível com o V9938 e com o TMS9918A. Este novo VDP acrescentou novas características ao vídeo, dentre as quais:

- Máximo de 19.268 cores apresentadas simultâneamente;
- Capacidade de sincronização externa;
- Possibilidade de múltiplas configurações MSX-VIDEO;
- Paletas de cores externas podem ser adicionadas usando a saída color-bus;
- Scroll vertical e horizontal por hardware.

O VDP V9958 também é usado nos modelos MSX turbo R, lançados em 1.990 e 1.991.

1 - OS REGISTRADORES DO VDP

O VDP V9938 tem 49 registradores e o V9958 tem 52 registradores internos para controlar as operações de vídeo. Esses registradores são divididos em três grupos. O grupo de controle e o grupo de status podem ser acessados pelo BASIC com a função VDP (n). O terceiro grupo, o de paletas, não pode ser acessado pelo BASIC.

O primeiro é o grupo de registros de controle. Eles vão de R#0 a R#23 e R#32 a R#46 para o V9938 (MSX2) e existem mais três, R#25 a R#27, para o V9958. São registradores de 8 bits apenas de escrita. O subgrupo que vai de R#0 a R#27 (Obs.: R#24 não existe) são registradores que controlam todos os modos de tela. O outro subgrupo, de R#32 a R#46 executam os comandos de hardware do VDP. Esses comandos serão descritos com detalhes mais adiante. A tabela seguinte descreve resumidamente as funções de cada re-

gistrador desse grupo.

R#0	VDP(0)	Registrador de modo #0
R#1	VDP(1)	Registrador de modo #1
R#2	VDP(2)	Nome da tabela de padrões
R#3	VDP(3)	Tabela de cores (low)
R#4	VDP(4)	Tabela geradora de padrões
R#5	VDP(5)	Tabela de atributos do sprites (low)
R#6	VDP(6)	Tabela geradora de padrões dos sprites
R#7	VDP(7)	Cor da borda e dos caracteres no modo texto
R#8	VDP(9)	Registrador de modo #2
R#9	VDP(10)	Registrador de modo #3
R#10	VDP(11)	Tabela de cores (high)
R#11	VDP(12)	Tabela de atributos dos sprites (high)
R#12	VDP(13)	Cor dos caracteres para a função "blink"
R#13	VDP(14)	Período de "blinking"
R#14	VDP(15)	Endereço de acesso à VRAM (high)
R#15	VDP(16)	Especificação indireta para S#n
R#16	VDP(17)	Especificação indireta para P#n
R#17	VDP(18)	Especificação indireta para R#n
R#18	VDP(19)	Ajuste de tela
R#19	VDP(20)	Examina nº de linha ao ocorrer interrupção
R#20	VDP(21)	Burst de cor para fase 0 (preset 00000000B)
R#21	VDP(22)	Burst de cor para fase 1/3 (preset 00111011B)
R#22	VDP(23)	Burst de cor para fase 2/3 (preset 00000101B)
R#23	VDP(24)	Scroll vertical
R#24	Este registrador não existe	
R#25	VDP(26)	Registrador de modo #4 (VDP V9958)
R#26	VDP(27)	Scroll horizontal (VDP V9958)
R#27	VDP(28)	Scroll horizontal fino (VDP V9958)
R#32	VDP(33)	SX: coordenada X a ser transferida (low)
R#33	VDP(34)	SX: coordenada X a ser transferida (high)
R#34	VDP(35)	SY: coordenada Y a ser transferida (low)
R#35	VDP(36)	SY: coordenada Y a ser transferida (high)
R#36	VDP(37)	DX: coordenada X de destino (low)
R#37	VDP(38)	DX: coordenada X de destino (high)
R#38	VDP(39)	DY: coordenada Y de destino (low)
R#39	VDP(40)	DY: coordenada Y de destino (high)
R#40	VDP(41)	NX: número de pontos a transferir na direção X (low)
R#41	VDP(42)	NX: número de pontos a transferir na direção X (high)
R#42	VDP(43)	NY: número de pontos a transferir na direção Y (low)
R#43	VDP(44)	NY: número de pontos a transferir na direção Y (high)
R#44	VDP(45)	CLR: transferência de dados para a CPU
R#45	VDP(46)	ARGT: registrador de argumento
R#46	VDP(47)	CMR: envia um comando ao VDP

O grupo seguinte é o grupo de registradores de estado. São registradores de 8 bits somente para leitura designados por S#0 a S#9. A listagem abaixo descreve suas funções:

S#0	VDP(8)	Informação de interrupção
S#1	VDP(-1)	Informação de interrupção
S#2	VDP(-2)	Registro de informação e controle
S#3	VDP(-3)	Coordenada X detectada (low)
S#4	VDP(-4)	Coordenada X detectada (high)

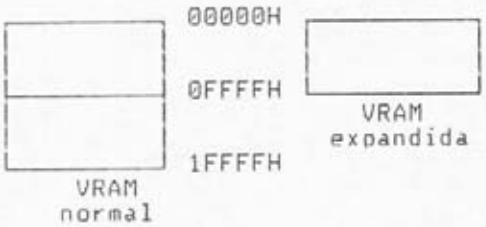
S#5	VDP(-5)	Coordenada Y detectada (low)
S#6	VDP(-6)	Coordenada Y detectada (high)
S#7	VDP(-7)	Dado obtido por um comando do VDP
S#8	VDP(-8)	Coordenada X obtida por comando de procura (low)
S#9	VDP(-9)	Coordenada X obtida por comando de procura (high)

Finalmente, o terceiro é o grupo de paleta de cores. São 16 registradores de 9 bits designados por P#0 a P#15. Cada registrador representa uma cor da paleta de 512 cores, reservando três bits para cada cor primária (verde, vermelho e azul), e possibilitando o uso de até 16 cores simultâneas.

1.1 - A VRAM E PORTAS DE ACESSO AO VDP

O VDP V9938 pode ser conectado a 64 ou 128 Kbuyes de VRAM, com um barramento de endereços de 17 bits. Já o V9958 deve ser conectado a 128 Kbytes de VRAM. Note que esta memória é controlada pelo VDP e não pode ser diretamente acessada pela CPU.

Opcionalmente, pode ser conectado mais um banco de 64 Kbytes de expansão. Entretanto, não há instruções específicas para essa expansão, o que pode ocasionar problemas de incompatibilidade entre máquinas diferentes para programas que usem a expansão.



PORTAS DE ACESSO AO VDP

Os VDPs V9938 e V9958 têm quatro portas de I/O para comunicação com a CPU. As funções dessas portas estão listadas na tabela abaixo. Na tabela, as portas são expressadas por **r** e **w** e seus valores estão armazenados respectivamente nos endereços 0006H e 0007H na Main-ROM.

r = (0006H) = porta de leitura (RDVDP)
w = (0007H) = porta de escrita (WRVDP)

É recomendável que sempre se use o BIOS para as funções de I/O para garantir a compatibilidade. Entretanto, quando as operações de tela requererem alta velocidade, essas portas podem ser usadas para acessar o VDP diretamente.

Porta #0 (leitura)	r	Lê dados da VRAM
Porta #0 (escrita)	w	Escreve dados na VRAM
Porta #1 (leitura)	r+1	Lê registrador de estado
Porta #1 (escrita)	w+1	Escreve no registrador de controle
Porta #2 (escrita)	w+2	Escreve nos registradores de paleta
Porta #3 (escrita)	w+3	Escreve no registrador especificado indiretamente.

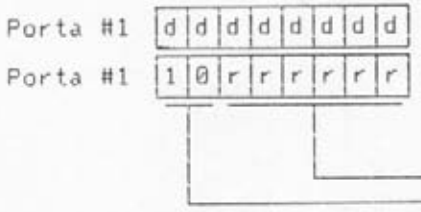
2 - ACESSO À VRAM E AO VDP

Escrevendo dados nos registros de controle

Os registros de controle são registradores apenas de escrita. Entretanto, o conteúdo do primeiro sub-grupo de registradores de controle (R#0 a R#27) pode ser obtido pelo comando VDP(n) do BASIC. Isto porque seus valores também são escritos na área de trabalho do sistema (endereços F3DFH a F3E6H / FFE7H a FFF6H / FFFAH a FFFCH).

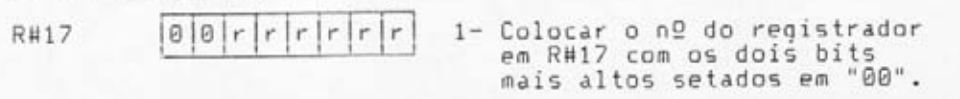
Existem três meios para escrever dados nos registradores de controle, descritos abaixo.

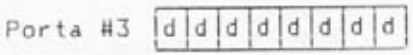
Acesso direto. O primeiro meio é especificar o dado e escrevê-lo diretamente. O dado é escrito primeiro, na porta #1, seguido do número do registrador de destino, conforme a ilustração abaixo:



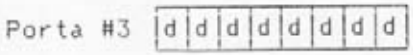
Acesso indireto. O segundo meio é escrever o dado no registrador especificado por R#17. Para isso, use o método direto para colocar o número do registrador em R#17, com os dois bits mais altos setados em "10" (como no método direto). Depois, pode-se enviar dados continuamente pela porta #3 para o mesmo registrador. Esse meio é útil para executar comandos do VDP.

Acesso direto com autoincremento. Esse meio é similar ao anterior, com a diferença que cada vez que um dado for escrito pela porta #3, R#17 é incrementado em um e o próximo dado enviado é escrito no registrador seguinte. Para usá-lo, coloque através do método direto o número do primeiro registrador em R#17, com os dois bits mais altos setados em "00". Depois, é só enviar os dados pela porta #3.





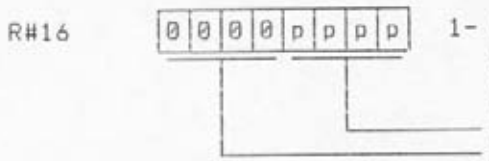
2- Envie o dado pela porta #3. O dado será colocado em R#n, sendo n o registrador especificado em R#17.



3- O dado seguinte enviado pela porta #3 será colocado em R#(n+1) e assim por diante.

2.1 - SETANDO A PALETA

Para colocar dados nos registradores de paleta (P#0 a P#15), especifique o número do registrador de paleta nos quatro bits mais baixos de R#16 e envie os dados pela porta #2. Como cada registrador de paleta tem 9 bits, os dados devem ser enviados por dois bytes consecutivos. Depois que os dois bytes forem enviados. R#16 é automaticamente incrementado. Esta característica torna a paleta fácil de ser inicializada. Veja a figura abaixo para entender melhor.



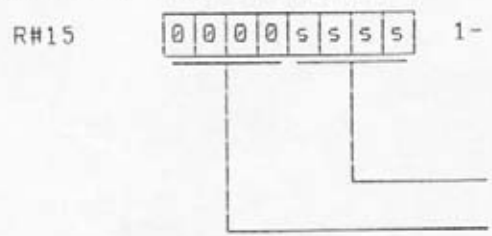
1- Coloque o nº do registrador de paleta em R#16 com os quatro bits mais altos setados em "0000".
Nº do registrador de paleta.
Sempre "0000".

2- Envie o nível de vermelho e azul pela porta #2.
Azul (0 a 7).
Vermelho (0 a 7).

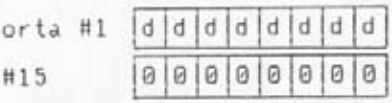
3- Depois envie o nível de verde também pela porta #2. Assim que o código de verde for enviado, R#16 é incrementado e aponta para o próximo registrador de paleta.
Verde (0 a 7).

2.2 - LENDO OS REGISTRADORES DE STATUS

Os registradores de status são registradores apenas de leitura. O conteúdo deles pode ser lido pela porta #1, colocando em R#15 o número do registrador de status que se deseja ler. As interrupções devem ser desativadas (DI) durante o processo de leitura de algum registrador de status. Depois de lido o registrador de status, R#15 deve ser setado em 0 antes das interrupções serem reabilitadas.



1- Coloque o nº do registrador de status em R#15, com os quatro bits mais altos setados em "0000". Importante: as interrupções devem estar desabilitadas (DI).
Número do registrador de status (0 a 9).
Sempre "0000".



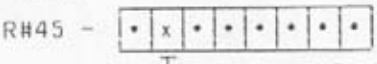
2- Leia o dado pela porta #1.

3- Sete R#15 em 0 antes de reabilitar as interrupções

.

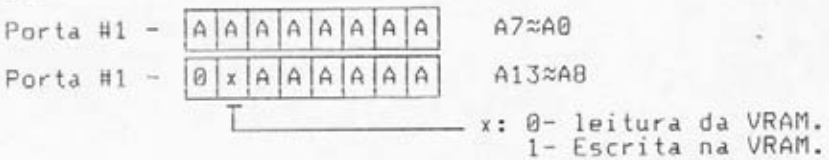
2.3 - ACESSO À VRAM PELA CPU

Usando a RAM expandida. Os primeiros 64 Kbytes de VRAM e a RAM expandida ocupam o mesmo espaço de endereçamento do VDP. Se o micro não possuir RAM expandida, sempre será selecionada a VRAM principal. Os dois bancos de 64K são controlados pelo bit 6 de R#45.



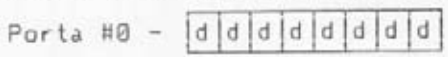
Setando a página da VRAM (três bits mais altos). O bus de 17 bits de endereços do VDP para acessar 128K de VRAM é designado por A16 a A0. R#14 contém os três bits mais altos, de A16 a A14, podendo selecionar 8 páginas de 16 Kbytes de VRAM.

Selecionando o endereço da VRAM. Os 14 bits mais baixos de endereçamento da VRAM (A0 a A13) devem ser enviados pela porta #1 em dois bytes consecutivos. Note que o bit 7 do segundo byte deve ser sempre 0 e o bit 6 deve ser 0 se quisermos fazer uma leitura ou 1 se quisermos fazer uma escrita.



Lendo e escevendo na VRAM. Depois de setar o endereço da VRAM, leia ou escreva o dado através da porta #0. A flag de leitura/escrita deve ser setada juntamente com A13 a A8, da forma descrita acima.

O contador de endereços é automaticamente incrementado em um cada vez que um byte for lido ou escrito pela porta #0. Essa característica facilita o acesso contínuo à VRAM.



O acesso à VRAM é feito pela porta #0. O contador de endereço é automaticamente incrementado.

3 - OS MODOS DE TELA DOS VDPs V9938 e V9958

O MSX2 tem vários modos de tela a mais que o MSX1. Foram acrescentados seis novos modos de tela. Já para o MSX2+ e o turbo R existem mais dois modos de tela. Na tabela da página seguinte, os modos marcados com um "*" (texto 2 e gráficos 3 a 7) foram introduzidos para o MSX2 (V9938) e os modos marcados com

doze modos de tela possiveis na tabela, juntamente com uma curta descrição de cada uma.

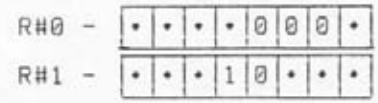
MODO	SCREEN	DESCRIÇÃO RESUMIDA
TEXTO 1	SCREEN 0 WIDTH = 40	40 caracteres por linha de texto; uma cor para todos os caracteres.
TEXTO 2 *	SCREEN 0 WIDTH = 80	80 caracteres por linha de texto; função de "blink" inclusa.
MULTICOR	SCREEN 3	Pseudo-gráfico; um caractere é dividido em quatro blocos.
GRAPHIC 1	SCREEN 1	32 caracteres por linha de texto; caracteres de várias cores são disponiveis.
GRAPHIC 2	SCREEN 2	256 x 192 pontos; 16 cores podem ser especificadas para cada 8 pontos horizontais.
GRAPHIC 3 *	SCREEN 4	Igual a GRAPHIC 2, mas usa sprites modo 2.
GRAPHIC 4 *	SCREEN 5	256 x 212 pontos; 16 cores de 512 são disponíveis para cada ponto.
GRAPHIC 5 *	SCREEN 6	512 x 212 pontos; 4 cores de 512 são disponíveis para cada ponto.
GRAPHIC 6 *	SCREEN 7	512 x 212 pontos; 16 cores de 512 são disponíveis para cada ponto.
GRAPHIC 7 *	SCREEN 8	256 x 212 pontos; até 256 cores são disponíveis para cada ponto.
GRAPHIC 8 **	SCREEN 10 SCREEN 11	256 x 212 pontos; 65536 cores disponíveis para cada quatro pontos horizontais ou 16 cores de 512 disponíveis para cada ponto. Máximo de 12499 cores simultâneas.
GRAPHIC 9 **	SCREEN 12	256 x 212 pontos; 131072 cores disponíveis para cada quatro pontos horizontais; máximo de 19268 cores simultâneas na tela.

3.1 - MODO TEXTO 1

-24 linhas de até 40 caracteres cada;
-Uma cor de fundo e uma cor para os caracteres, selecionadas de 16 (MSX1) ou 512 (MSX2 ou superior);
-256 caracteres disponíveis com resolução de 6 pontos horizontais por 8 verticais para cada caractere;
-Requer 2048 bytes para a fonte (8 bytes x 256 caracteres) e e 960 bytes para a tela (40 caracteres x 24 linhas);
-Compativel com SCREEN 0 (WIDTH 40)

O modo TEXTO 1 é selecionado usando 5 bits de R#0 e

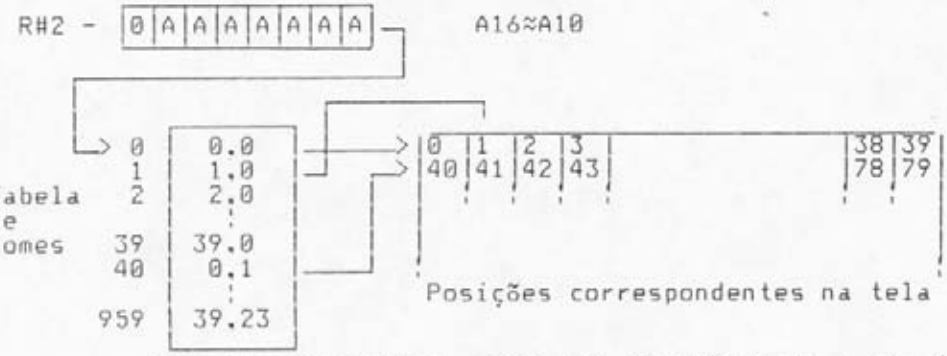
R#1. Os três bits em R#0 devem ser "000" e os dois bits em R#1 devem ser "10". Veja a figura abaixo:



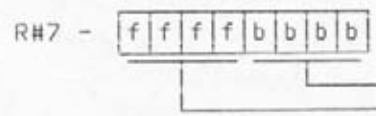
A área onde a fonte de caracteres é armazenada chama-se *tabela geradora de padrões*. Ela está localizada na VRAM e cada caractere é definido por 8 bytes, mas os dois bits mais baixos de cada byte não são mostrados. Portanto, o tamanho máximo de cada caracter é de 6 x 8 pontos. A fonte contém 256 caracteres diferentes, numerados de 0 a 255.

A localização da tabela geradora de padrões está especificada em R#4. Note que os 6 bits mais altos de endereçamento (A16 a A11) estão especificados e que os 11 bits mais baixos (A10 a A0) são sempre 0. Por isso, a tabela geradora de padrões sempre começa em um múltiplo de 2 Kbytes a partir de 00000H. Esse endereço pode ser obtido pela variável de sistema BASE(2) do BASIC.

A *tabela de nomes dos padrões* armazena a posição de cada caractere que deve ser apresentado na tela. Um byte é usado para cada caractere. O endereço da tabela de nomes está especificado em R#2. Note que os 7 bits mais altos de endereço (A16 a A10) estão especificados e que os 10 bits mais baixos são sempre 0. Por isso, a tabela de nomes sempre começa em um múltiplo de 1 Kbyte a partir de 00000H. Esse endereço pode ser obtido usando a variável de sistema BASE(0) do BASIC.



Para especificar as cores dos caracteres e a cor de fundo, é usado o registrador R#7. Os quatro bits mais altos de R#7 especificam a cor dos caracteres (cor de frente) e os quatro bits mais baixos a cor de fundo e da borda.

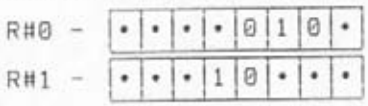


3.2 - MODO TEXTO 2

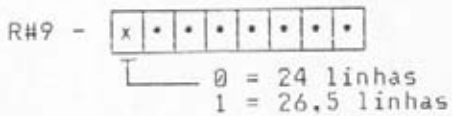
-24 ou 26,5 linhas de até 80 caracteres cada:

de 512 cores:
-256 caracteres disponíveis com resolução de 6 pontos horizontais por 8 verticais para cada caractere:
-Função de "blink" (piscar) independente para cada caractere:
-Requer 2048 bytes para a fonte (256 caracteres x 8 bytes):
-Para 24 linhas requer 1920 bytes para a tela (80 caract. x 24 linhas) e 240 bytes (1920 bits) para os atributos de blinking:
-Para 26,5 linhas, requer 2160 bytes para tela (80 caract. x 27 linhas) e 270 bytes (2160 bits) para os atributos de blinking:
-Compatível com SCREEN 0 (WIDTH 80).

O modo texto 2 é selecionado por cinco bits de R#0 e R#1, conforme ilustrado abaixo.

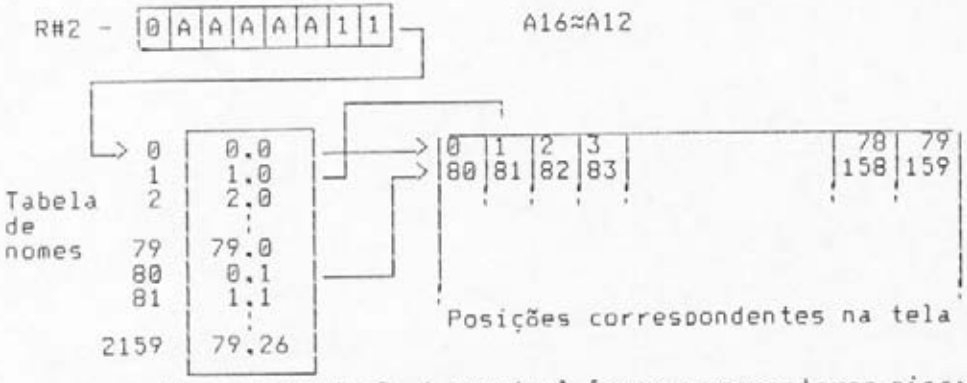


O modo texto 2 pode apresentar 24 linhas ou 26,5 linhas, dependendo do valor do bit 7 de R#9. Note que no modo 26,5 linhas, na última linha apenas a metade superior dos caracteres é apresentada. Esse modo não é suportado pelo BASIC.



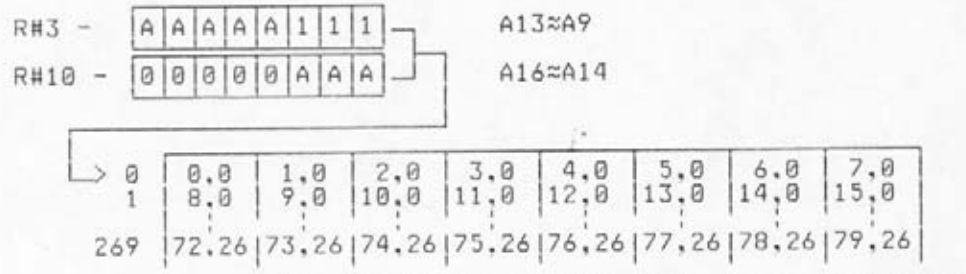
A tabela geradora de padrões do modo texto 2 tem a mesma estrutura e função que a do modo texto 1.

Como o número de caracteres que podem ser mostrados nesse modo foi aumentado para um máximo de 2160 (80 x 27), o valor máximo ocupado pela tabela de nomes é de 2160 bytes. O endereço inicial da tabela de nomes deve ser especificado em R#2. Os cinco bits mais altos (A16 a A12) especificam o endereço e os 12 bits mais baixos (A11 a A0) são sempre 0. Por isso, o endereço inicial da tabela de nomes é sempre um múltiplo de 4 Kbytes a partir de 00000H.

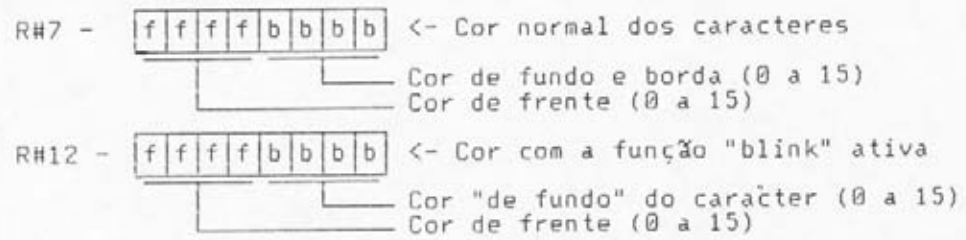


No modo texto 2, é possível fazer os caracteres piscarem. Esse recurso é chamado de "blink". A tabela de "blink" armazena a posição de cada caractere na tela; Um bit na tabela cor-

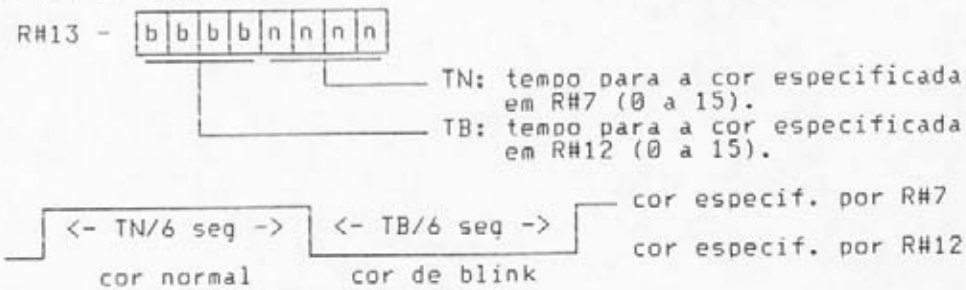
responde a um caractere. Quando esse bit for 1, a função de "blink" é ativada para o respectivo caractere e quando for 0 a função é desativada. O endereço da tabela de "blink" é armazenado em R#3 e R#10. Os 8 bits mais altos (A16 a A9) especificam o endereço e os 9 bits mais baixos são sempre 0. Por isso, o endereço na tabela de "blink" é sempre um múltiplo de 512 bytes.



As cores dos caracteres no modo texto 2 são especificadas em R#7 e R#12. Os quatro bits mais altos de R#7 especificam a cor dos caracteres (cor de frente) e os quatro bits mais baixos representam a cor de fundo e da borda. Quando a função de "blink" estiver ativa, a cor "de fundo" do caractere será especificada pelos quatro bits mais altos de R#12 e a cor do caractere pelos quatro bits mais baixos de R#12.



O tempo de "blinking", ou seja, o tempo em que o caractere assume as cores de "blink" e depois volta às cores normais é especificado em R#13. Os quatro bits mais altos de R#13 definem o tempo em que o caractere fica com a cor original e os quatro bits mais baixos definem o tempo em que o caractere fica na cor de "blink". O período de tempo é especificado em unidades de 1/6 de segundo.



3.3 - MODO MULTICOR

-64 (horizontal) por 48 (vertical) blocos;
-Até 16 cores podem ser apresentadas simultameamente;

-Cada bloco tem 4 x 4 pontos e uma cor;
-Requer 2048 bytes para a tabela de cores e 768 bytes para especificar a localização na tela;
-Sprites modo 1;
-Compativel com SCREEN 3.

O modo MULTICOR é selecionado por R#0 e R#1, conforme ilustrado abaixo:

R#0 -	•	•	•	•	0	0	0	•
R#1 -	•	•	•	0	1	•	•	•

A organização desse modo é muito complexa e ele praticamente não tem utilização, por sua resolução extremamente baixa (apenas 64 x 48 pontos), que o tornou totalmente obsoleto. Por estes motivos, seu funcionamento não será descrito aqui.

3.4 - MODO GRÁFICO 1

-32 (horizontal) x 24 (vertical) padrões;
-Até 16 cores podem ser apresentadas simultameamente (escolhidas de 512 para MSX2 ou superior);
-256 tipos de padrões são disponiveis;
-Cada padrão tem 8 x 8 pontos e pode ser definido com qualquer figura;
-Cores diferentes para cada 8 padrões podem ser usadas;
-Requer 2048 bytes para a fonte de padrões, 768 bytes para a tabela de nomes e 32 bytes para a tabela de cores;
-Sprites modo 1;
-Compativel com SCREEN 1.

O modo gráfico 1 é selecionado por R#0 e R#1 conforme a ilustração abaixo:

R#0 -	•	•	•	•	0	0	0	•
R#1 -	•	•	•	0	0	•	•	•

Neste modo, 256 tipos de padrões, ou caracteres, correspondentes aos códigos 0 a 255, podem ser apresentados na tela. A fonte de cada padrão é definida na tabela geradora de padrões. O endereço inicial da tabela geradora de padrões é especificado em R#4. Note que somente os 6 bits mais altos de endereço (A16 a A11) são especificados.

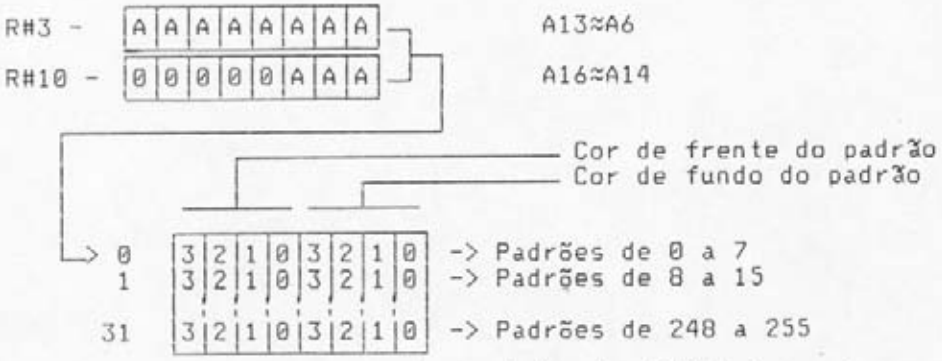
R#4 -	0	0	A	A	A	A	A	A

A16≈A11

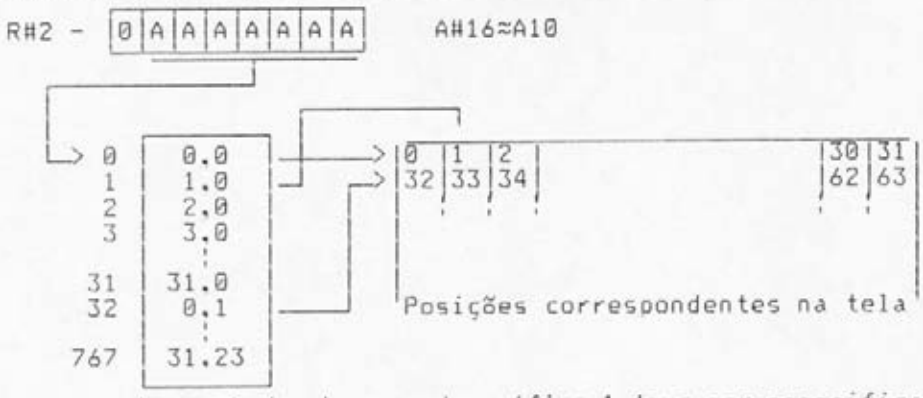
Endereço inicial da tabela geradora de padrões.

A tabela de cores especifica uma cor para cada 8 padrões ou caracteres na tabela geradora de padrões. O endereço inicial da tabela de cores é especificado em R#3 e R#10. Note que somente os 11 bits mais altos de endereço são especificados (A16 a A6).

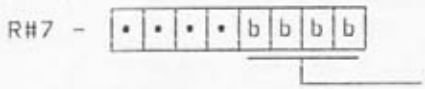
Para maior esclarecimento, veja na página seguinte a ilustração de como funciona esse endereçamento.



A tabela de nomes dos padrões tem 768 bytes e é a responsável pela apresentação dos padrões ou caracteres na tela. O endereço inicial dessa tabela é especificado em R#2. Observe que apenas os 7 bits mais altos são especificados (A16 a A10).



A cor da borda no modo gráfico 1 deve ser especificada ros 4 bits mais baixos de R#7.



3.5 - MODOS GRÁFICOS 2 E 3

-32 (horizontal) por 24 (vertical) padrões;
-Até 16 cores podem ser apresentadas simultaneamente;
-768 padrões diferentes são disponíveis;
-Cada padrão tem 8x8 pontos;
-Qualquer figura pode ser definida para cada padrão;
-Apenas duas cores podem ser definidas para cada 8 pontos horizontais;
-Requer 6144 bytes para a fonte de padrões e mais 6144 bytes para a tabela de cores;
-Sprites modo 1 para Graphic 2 e modo 2 para Graphic 3;
-Graphic 2 compatível com SCREEN 2 e Graphic 3 com SCREEN 4.

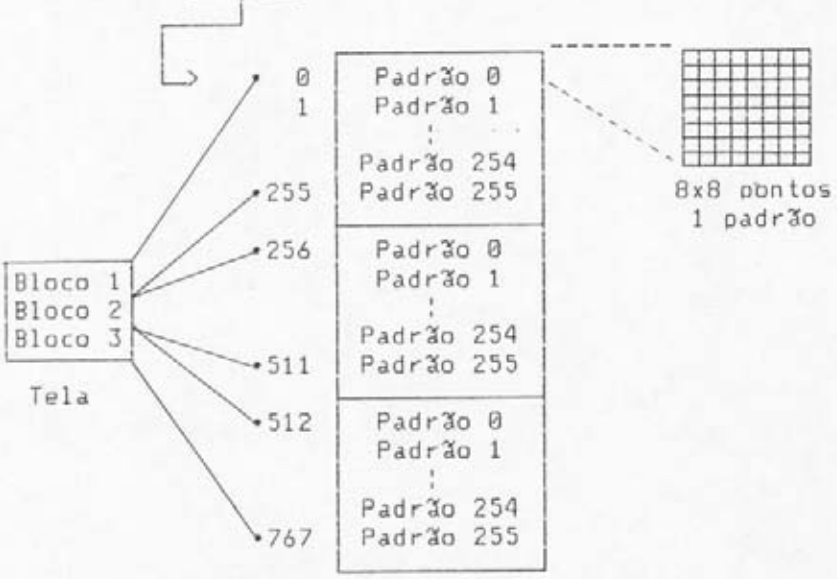
Os modos gráficos 2 e 3 são selecionados por R#0 e R#1 conforme ilustração na página seguinte.

GRAPHIC 2	R#0 -	•	•	•	•	0	0	1	•
	R#1 -	•	•	•	0	0	•	•	•
GRAPHIC 3	R#0 -	•	•	•	•	0	1	0	•
		•	•	•	0	0	•	•	•

Nesses dois modos, a tabela geradora de padrões é compatível com o modo gráfico 1, onde 768 padrões diferentes podem ser mostrados. Como cada padrão tem 8 x 8 pontos e pode ter um desenho diferente, há uma simulação de apresentação de 256 x 192 pontos na tela. O endereço inicial da tabela geradora de padrões é especificado em R#4. Apenas os 4 bits mais altos do endereço são válidos (A16 a A13); por isso, o endereço inicial será sempre um múltiplo de 8 Kbytes a partir de 00000H. Nesse modo, a tela é dividida em três blocos de 256 padrões cada um, perfazendo um total de 768 padrões.

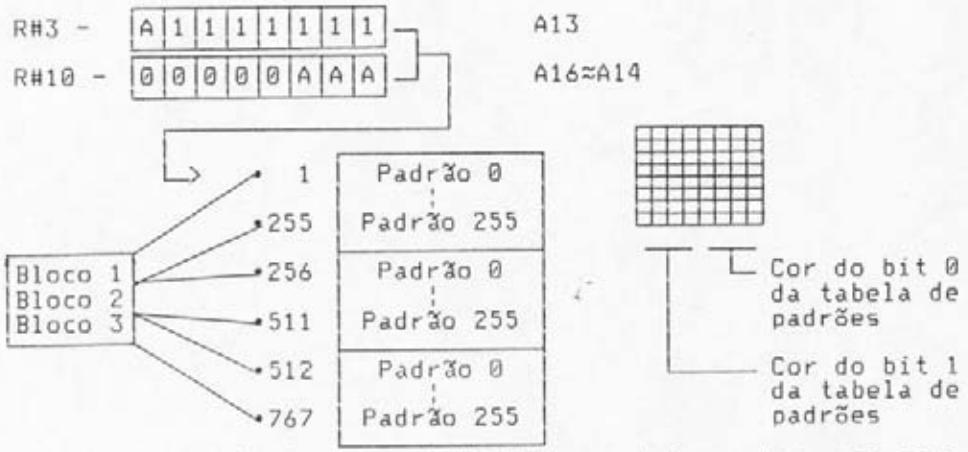
R#4 -	0	0	A	A	A	A	1	1

A16≈A13

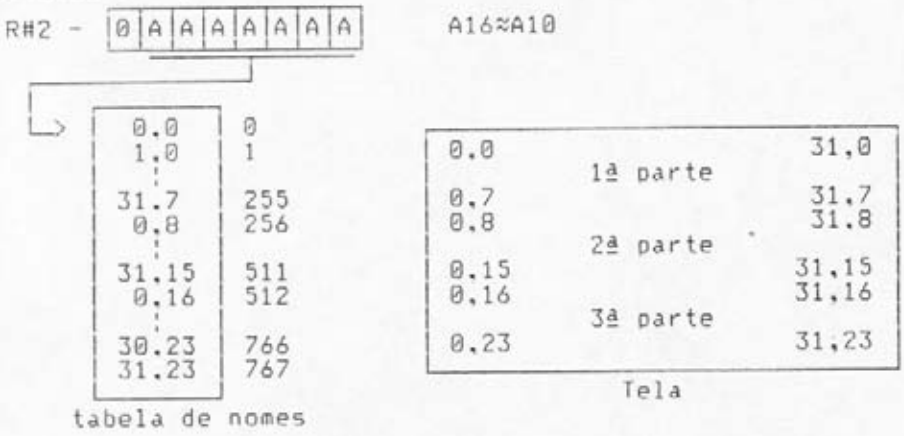


O tamanho da tabela de cores é o mesmo da tabela geradora de padrões e as cores podem ser especificadas para cada bit 0 ou 1 de cada linha horizontal de cada padrão. O endereço inicial da tabela de cores é especificado por R#3 e R#10.

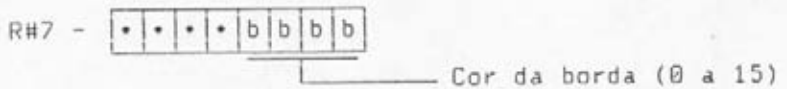
Note que apenas os quatro bits mais altos são especificados. Veja a ilustração de como é organizada esta tabela na página seguinte.



A tabela de nomes é dividida em três partes, uma para cada bloco da tela. Cada parte tem 256 bytes e é reponsável pela apresentação de 256 padrões na tela. O endereço inicial da tabela de nomes é especificado em R#2.



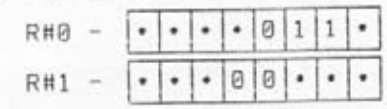
A cor da borda nos modos gráficos 2 e 3 é especificada nos 4 bits mais baixos de R#7.



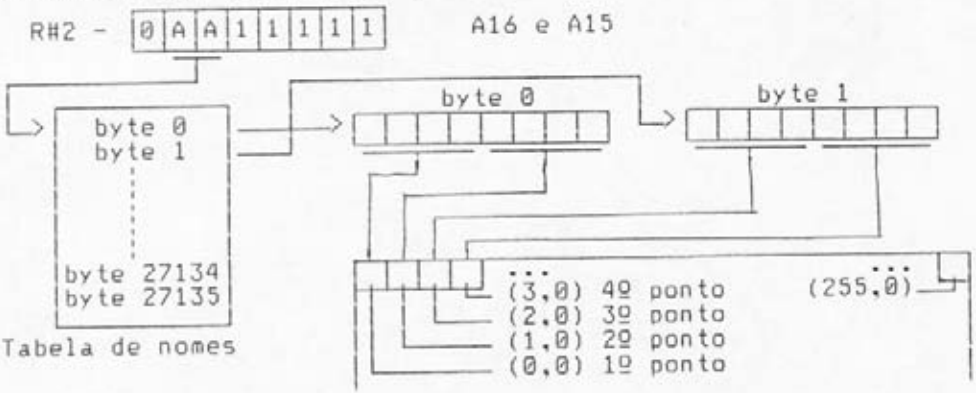
3.6 - MODO GRÁFICO 4

-256 (horizontal) por 212 (vertical) pontos;
-Apresenta até 16 cores simultâneas escolhidas de 512;
-Possui comandos gráficos de alta velocidade;
-Sprites modo 2;
-Requer 24 Kbytes (4bits x 256 x 192) ou 26,5 Kbytes (4bits x 256 x 212) de memória;
-Gráficos bit-mapped de fácil manipulação;
-Compatível com SCREEN 5.

O modo gráfico 4 é selecionado por R#0 e R#1, conforme a ilustração abaixo:



No modo gráfico 4, um byte na tabela geradora de padrões corresponde a dois pontos na tela. Cada ponto é representado por 4 bits e portanto podem ser especificadas 16 cores para cada ponto. O endereço inicial da tabela geradora de padrões é especificado em R#2. Apenas os dois bits mais altos de endereço podem ser especificados (A16 e A15). Por isso, a tabela sempre começa em 00000H, 08000H, 10000H ou 18000H.

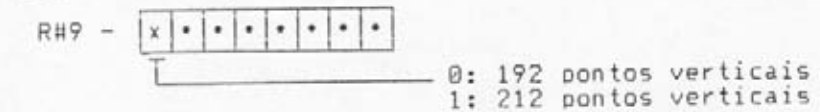


Localização dos pontos na tela

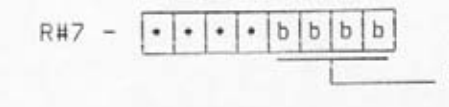
Observe que cada byte corresponde a dois pontos em seqüência horizontal na tela e como a resolução horizontal é de 256 pontos, 128 bytes serão necessários para cada linha de tela. Observe também que nesse modo não há necessidade da tabela de nomes. O endereço de cada ponto pode ser calculado pela seguinte expressão:

ENDEREÇO = X/2 + Y*128 + ENDEREÇO INICIAL

Onde X é a coordenada horizontal e Y é a coordenada vertical do ponto. O ponto é especificado pelos 4 bits mais altos do endereço se X for par e pelos 4 bits mais baixos se X for impar. O número de pontos verticais deve ser especificado em R#9. veja a ilustração abaixo.



A cor da borda deve ser especificada nos 4 bits mais baixos de R#7. Veja a ilustração:



3.7 - MODO GRÁFICO 5

-512 (horizontal) por 212 (vertical) pontos;
-Apresenta até 4 cores simultâneas escolhidas de 512;
-Comandos de hardware são disponíveis;
-Sprites modo 2;
-Requer 24 Kbytes (2bits x 512 x 192) ou 26,5 Kbytes (2bits x 512 x 212) de memória;
-Gráficos bit-mapped de fácil manipulação;
-Compatível com SCREEN 6.

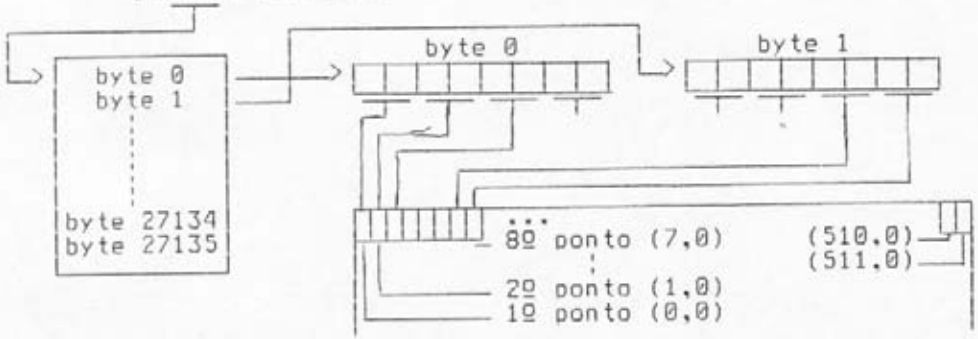
O modo gráfico é selecionado por R#0 e R#1, conforme a ilustração abaixo.

R#0 -	•	•	•	•	1	0	0	•
R#1 -	•	•	•	0	0	•	•	•

No modo gráfico 5, um byte na tabela de padrões corresponde a quatro pontos na tela. Cada ponto é representado por 2 bits e portanto apenas quatro cores podem ser especificadas. O endereço inicial da tabela geradora de padrões é especificado em R#2, sendo que apenas os dois bits mais altos são válidos. Por isso, a tabela sempre começa nos endereços 00000H, 08000H, 10000H ou 18000H. Veja o esquema abaixo.

R#2 -	0	A	A	1	1	1	1	1

A16 e A15



Localização dos pontos na tela

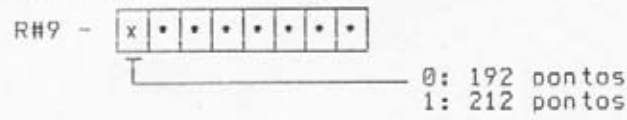
Observe que cada byte representa quatro pontos em seqüência horizontal na tela e como temos 512 pontos de resolução horizontal, serão necessários 128 bytes para representar cada linha de tela. O endereço de cada ponto na tabela pode ser calculado pela seguinte expressão:

ENDEREÇO = X/4 + Y*128 + ENDEREÇO INICIAL

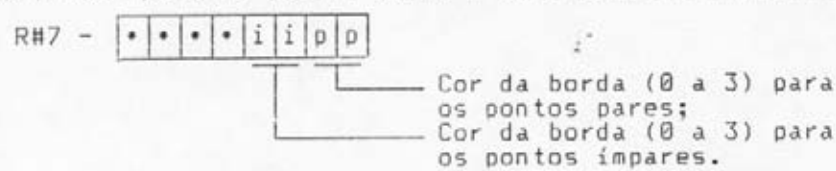
Onde X é a coordenada horizontal e Y a coordenada vertical. Mas como cada byte representa quatro pontos é necessária mais uma operação para saber qual par de bits representa o ponto.

Se X mod 4 = 0, o ponto é representado pelos bits 7 e 6;
Se X mod 4 = 1, o ponto é representado pelos bits 5 e 4;
Se X mod 4 = 2, o ponto é representado pelos bits 3 e 2;
Se X mod 4 = 3, o ponto é representado pelos bits 1 e 0.

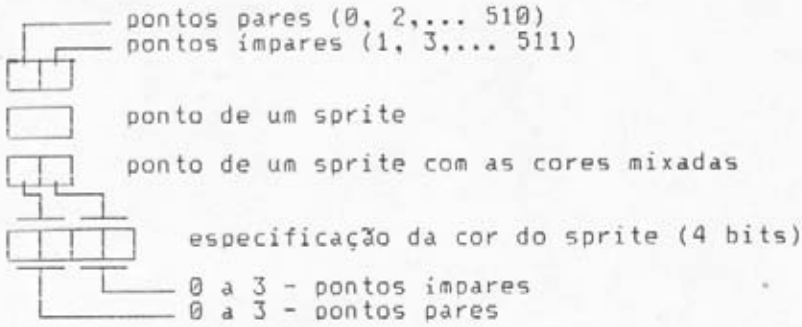
O número de pontos verticais deve ser especificado em R#9. Veja a figura abaixo.



No modo gráfico 5, há um tratamento especial para a cor da borda e dos sprites. A cor é especificada por quatro bits em R#7, dois para os pontos pares e dois para os ímpares.



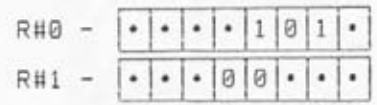
Para os sprites, as cores são especificadas como ilustrado abaixo.



3.8 - MODO GRÁFICO 6

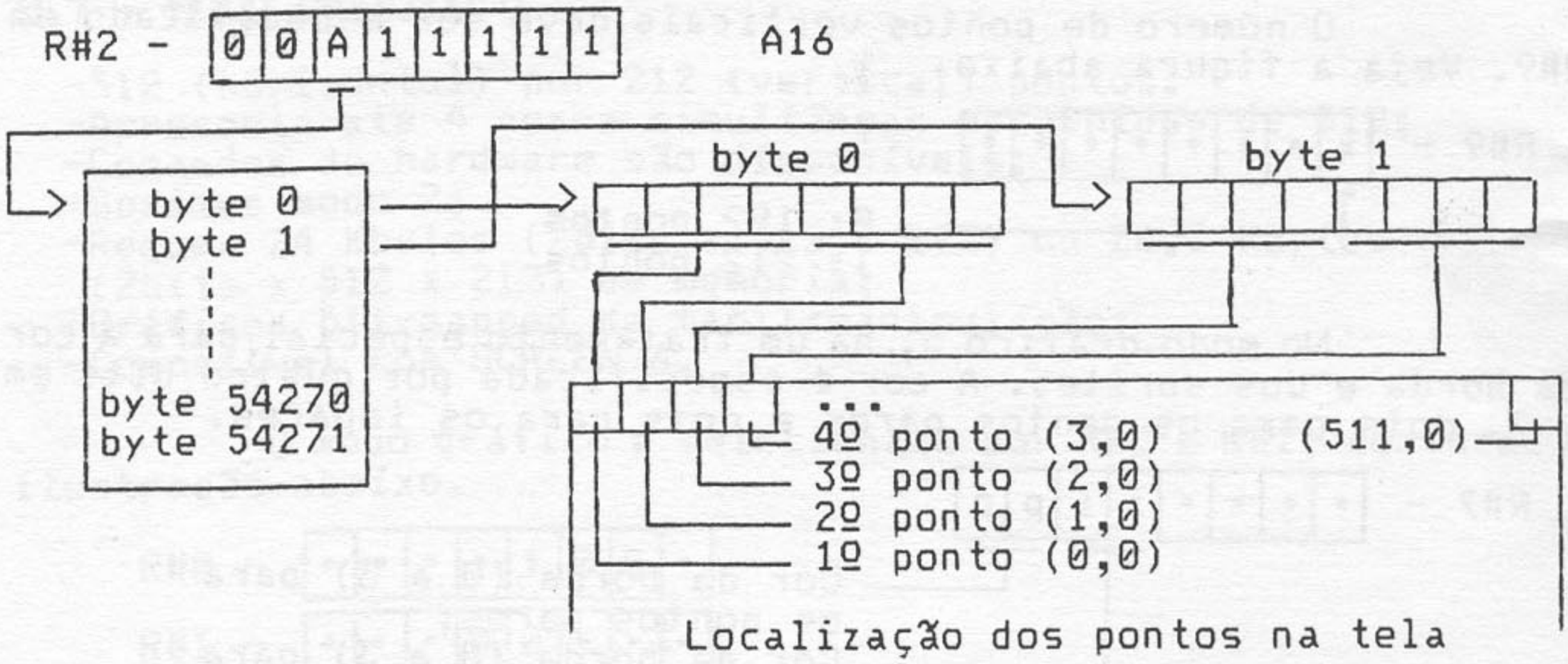
-512 (horizontal) por 212 (vertical) pontos;
-Apresenta até 16 cores simultâneas escolhidas de 512;
-Comandos de hardware estão disponíveis;
-Requer 48 Kbytes (4bits x 512 x 192) ou 53 Kbytes (4bits x 512 x 212) de memória;
-Sprites modo 2;
-Compatível com SCREEN 7.

O modo gráfico 6 é selecionado por R#0 e R#1 conforme a ilustração abaixo.



No modo gráfico 6, um byte na tabela de padrões corresponde a dois pontos na tela. Cada ponto é representado por 4 bits e 16 cores podem ser especificadas. O endereço de início da tabela de padrões é especificado por um único bit em R#2, e os dois endereços iniciais da tabela podem ser 00000H ou 10000H.

Veja a ilustração na página seguinte.

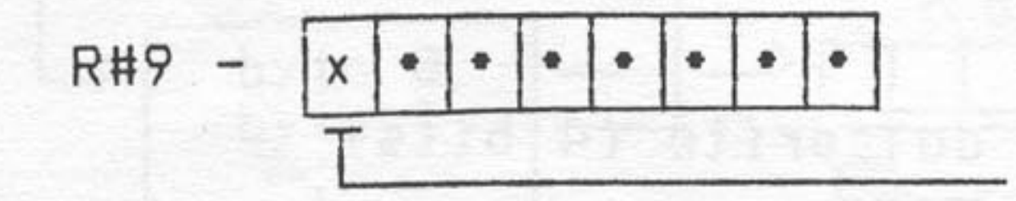


Localização dos pontos na tela

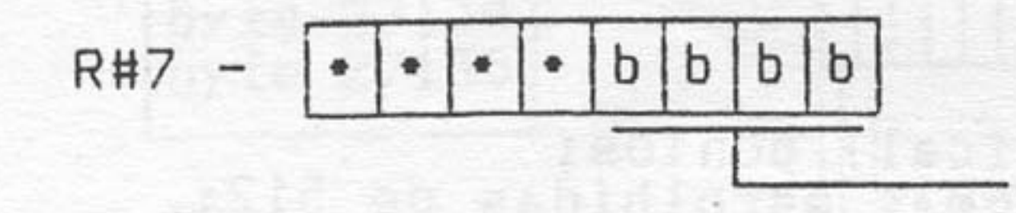
Como temos 512 pontos em cada linha horizontal e cada byte representa dois pontos, significa que são precisos 256 bytes para cada linha de tela. O endereço de cada ponto na tabela pode ser calculado pela seguinte expressão:

ENDEREÇO = X/2 + Y*256 + ENDEREÇO INICIAL

Onde X é a coorcenada horizontal do ponto e Y a coordenada vertical. A cor do ponto será representada pelos 4 bits mais altos se X for par ou pelos 4 bits mais baixos se X for ímpar. O número de pontos verticais deve ser especificado em R#9.



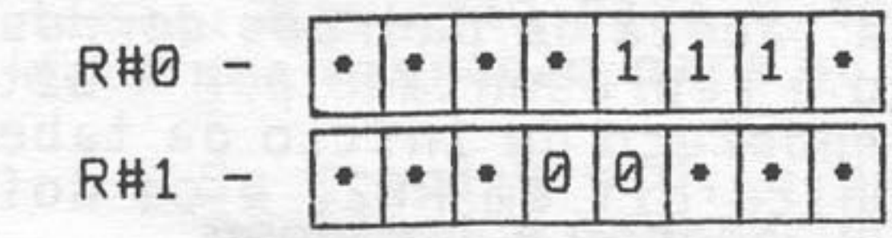
A cor da borda é especificada pelos 4 bits mais baixos de R#7.



3.9 - MODO GRÁFICO 7

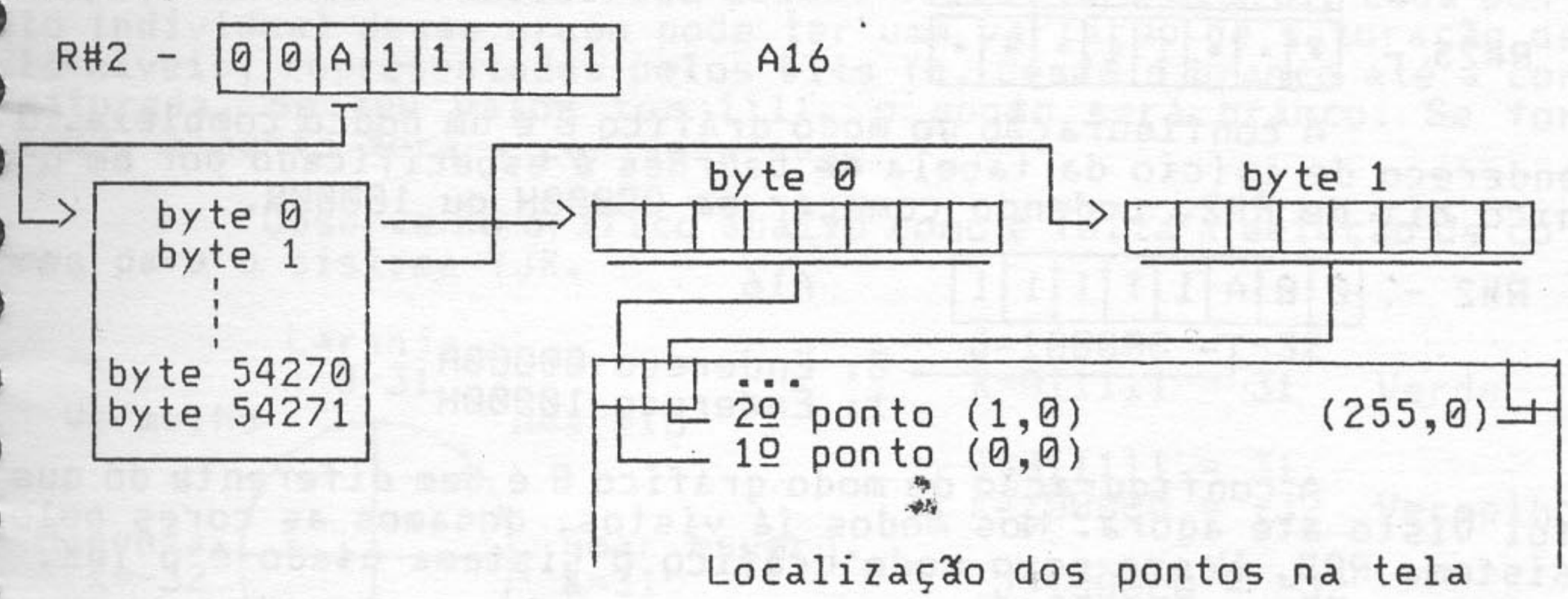
-256 (horizontal) por 212 (vertical) pontos;
-Máximo de 256 cores podem apresentadas simultaneamente;
-Comandos de hardware estão disponíveis;
-Requer 48 Kbytes (8bits x 256 x 192) ou 53 Kbytes (8bits x 256 x 212) de memória;
-Sprites modo 2;
-Compatível com SCREEN 8.

O modo gráfico 7 é selecionado por R#0 e R#1, conforme a ilustração abaixo:



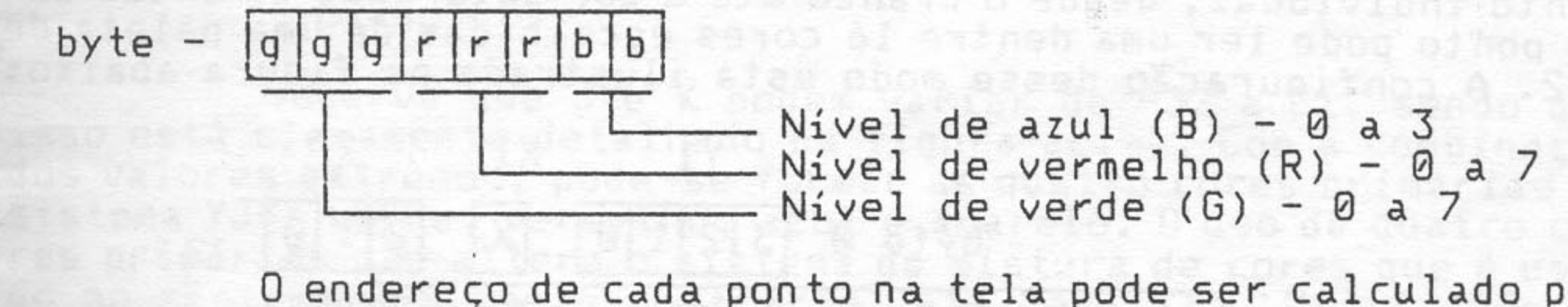
A configuração do modo gráfico 7 é a mais simples de todas; um ponto na tela corresponde a um byte na tabela de pa-

drões, podendo portanto apresentar até 256 cores simultâneas. O endereço inicial da tabela de padrões é especificada em R#2 por um único bit. Por isso, a tabela de padrões só pode começar nos endereços 00000H ou 10000H.



Localização dos pontos na tela

Neste modo, não é usada a paleta de cores, sendo que cada byte de dados reserva 3 bits de intensidade para o verde, 3 bits para o vermelho e dois bits para o azul.

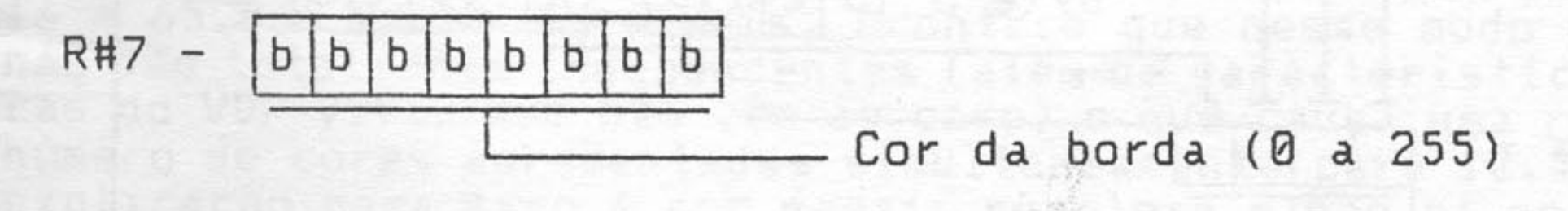


O endereço de cada ponto na tela pode ser calculado pela seguinte expressão:

ENDEREÇ0 = X + Y*256 + ENDEREÇO INICIAL

Onde X é a coordenada horizontal e Y a coordenada vertical. O número de pontos verticais deve ser especificado em R#9.

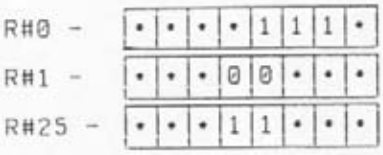
A cor da borda deve ser especificada em R#7, no mesmo formato dos bytes de dados da tela. Todos os bits de R#7 são válidos.



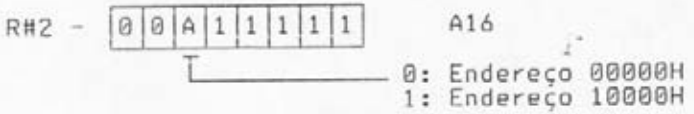
3.10 - MODO GRÁFICO 8

-256 (horizontal) por 212 (vertical) pontos;
-Até 12.499 cores podem ser apresentadas simultaneamente;
-Cores são especificadas para cada 4 pontos horizontais;
-Comandos de hardware estão disponíveis;
-Requer 48 Kbytes (192 pontos verticais) ou 53 Kbytes (212 pontos verticais) de memória;
-Sprites modo 2;
-Compatível com SCREEN 10 e SCREEN 11;
-Esse modo só é suportado pelo MSX2+ ou superior.

O modo gráfico 8 é selecionado por R#0, R#1 e R#25, conforme a ilustração da página seguinte.

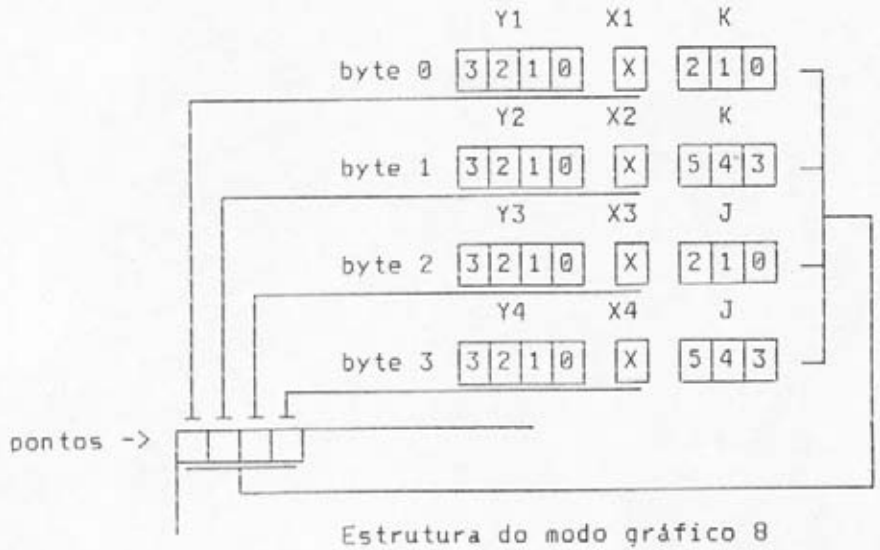


A configuração do modo gráfico 8 é um pouco complexa. O endereço de início da tabela de padrões é especificado por um único bit de R#2, podendo começar em 00000H ou 10000H.



A configuração do modo gráfico 8 é bem diferente do que foi visto até agora. Nos modos já vistos, dosamos as cores pelo sistema RGB. Nesse novo modo gráfico o sistema usado é o YJK.

Neste modo, os pontos estão organizados de quatro em quatro na horizontal. Cada grupo de 4 pontos pode ter uma única cor, escolhidas de 4096, com até 16 níveis de saturação para cada ponto individual, desde o branco até a cor saturada; ou então cada ponto pode ter uma dentre 16 cores escolhidas de uma paleta de 512. A configuração desse modo está ilustrada na figura abaixo:



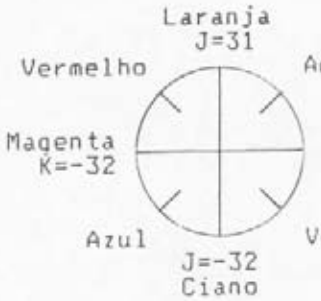
Estrutura do modo gráfico 8

Quando os bits Xn forem 1, a cor para cada ponto será escolhida da paleta de 512, com os quatro bits Yn variando de 0 a 15, tal qual as cores são escolhidas para o modo gráfico 4. Nesse caso, os bits J e K são ignorados. Observe que não é obrigatório que todos os bits X de um grupo de 4 pontos sejam iguais a 1, podendo haver mistura nos quatro pontos que compõem o grupo.

Quando os bits X forem 0, então será usado o sistema YJK para o respectivo ponto. A cor é escolhida pelos vetores J e K, sendo que J é representado por seis bits e K por outros seis,

conforme a figura da página anterior. Como há 12 bits para representar a cor, temos 2^12 = 4.096 cores, que é o número máximo de cores que podem ser definidas. Observe que cada grupo de 4 pontos só pode ter uma cor escolhida dessas 4.096. Entretanto, cada ponto individual desse grupo pode ter uma variação de saturação de 16 niveis, representadas pelos bits Yn, desde o branco até a cor saturada. Se seu valor for 1111, o ponto será branco. Se for 0000, o ponto terá a cor saturada.

Observe no gráfico abaixo como é feita a seleção de cores para o sistema YJK.



J=100000 = -32
K=011111 = 31 Verde

J=011111 = 31
K=100000 = -32 Vermelho

J=100000 = -32
K=100000 = -32 Azul

J=011111 = 31
K=011111 = 31 Amarelo

Observe que J e K podem variar de -32 a 31, sendo que isso está claramente detalhado na figura acima. Com a combinação dos valores extremos, pode-se formar as quatro cores primárias do sistema YJK: verde, vermelho, azul e amarelo. O uso de quatro cores primárias não altera o sistema de mistura de cores que é usado no sistema RGB, pelo contrário, até facilita. Utilizando os valores intermediários, podem ser geradas até 4.096 cores. A conversão do sistema YJK para RGB e vice-versa está mostrada nas fórmulas abaixo:

```
Y = R/4 + G/8 + B/2        R = Y + J
J = R - Y                  G = Y + K
K = G - Y                  B = 5/4 Y - 1/2 J - 1/4 K
```

Um detalhe importante é quanto ao número de cores. Como temos 4096 cores e 16 níveis de saturação, na verdade são 4.096 x 16 = 65.536 cores possiveis. Acontece que nesse modo os pontos não são totalmente independentes (além de caracteristicas técnicas do VDP V9958 que não vêm ao caso) o que causa uma redução no número de cores apresentadas simultaneamente para 12.499, mas a explicação para isso é por demais complexa e não há necessidade dela para se usar todos os recursos do modo gráfico 8.

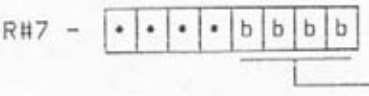
O endereço de cada ponto na tela no modo gráfico 8 pode ser calculado pela seguinte expressão:

```
ENDEREÇO = X + Y*256 + ENDEREÇO INICIAL
```

Onde X é a coordenada horizontal e Y a vertical.

O número de pontos verticais deve ser especificado em R#9, como já descrito.

A cor da borda deve ser especificada em R#7, obedecendo à paleta de cores, como no modo gráfico 4.

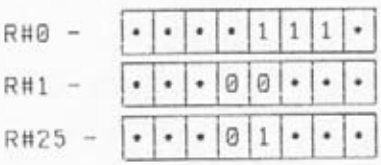


A diferença entre a SCREEN 10 e a SCREEN 11 está no tratamento dado a elas pela ROM. A SCREEN 10 é tratada como a SCREEN 5, enquanto a SCREEN 11 é tratada como a SCREEN 8, tanto pela ROM como pelo Interpretador BASIC.

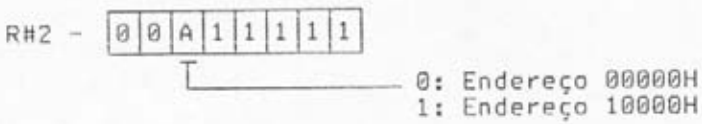
3.11 - MODO GRÁFICO 9

-256 (horizontal) por 212 (vertical) pontos;
-Até 19.268 cores podem ser apresentadas simultaneamente;
-Cores são especificadas para cada 4 pontos horizontais;
-Requer 48 Kbytes (192 pontos verticais) ou 53 Kbytes (212 pontos verticais) de memória;
-Comandos de hardware são disponíveis;
-Sprites modo 2;
-Este modo só é suportado pelo MSX2+ ou superior.

O modo gráfico 9 é selecionado por R#0, R#1 e R#25:



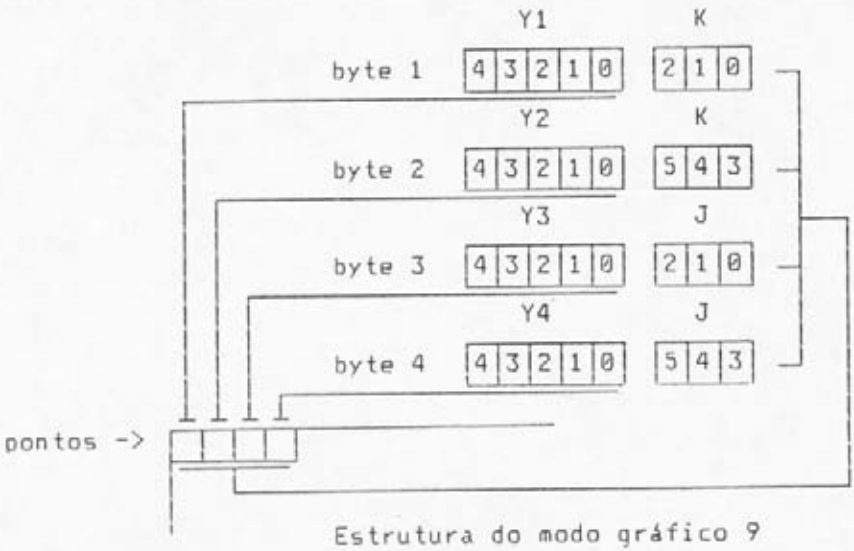
A organização do modo gráfico 9 é semelhante e mais simples que a do modo gráfico 8. O endereço de início da tabela de padrões é especificado por um único bit em R#2, podendo começar apenas em 00000H ou 10000H.



No modo gráfico 9 é usado o sistema YJK puro. Os pontos estão organizados de quatro em quatro na tela, sendo que cada grupo de quatro pontos podem ter uma única cor, escolhida de 4096 com até 32 niveis de saturação para cada ponto individual, do branco até a cor saturada.

A cor é escolhida pelos vetores J e K exatamente da mesma forma que no modo gráfico 8. Já o valor de Y, que é o valor de saturação, pode variar de 11111 (branco) até 00000 (cor saturada), ou seja, de 0 a 31. Como temos 4.096 cores, escolhidas por J e K, e 32 niveis de saturação para cada ponto, temos 4.096 x 32 = 131.072 cores possíveis, mas por motivos já explicados no modo gráfico 8, há uma redução para o número de cores que podem ser apresentadas simultaneamente para 19.268.

Veja a ilustração da organização do modo gráfico 9 na página seguinte.



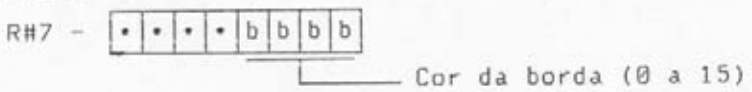
O endereço de cada ponto na tela para o modo gráfico 9 pode ser calculado pela seguinte expressão:

ENDEREÇO = X + Y*256 + ENDEREÇO INICIAL

Onde X é a coordenada horizontal e Y a vertical.

O número de pontos verticais deve ser especificado em R#9, como já descrito.

A cor da borda deve ser especificada em R#7, obedecendo à paleta de cores, tal qual no modo gráfico 8.



4 - MISCELANEA DE FUNÇÕES DE TELA

• LIGA/DESLIGA A TELA

A função de ligar e desligar a apresentação na tela é controlada pelo bit 6 de R#1. Quando estiver desligada, a tela inteira fica com a cor especificada pelos quatro bits mais baixos de R#7 (8 bits no modo gráfico 7). Os comandos de hardware do VDP ficam mais rápidos quando a tela está desligada.

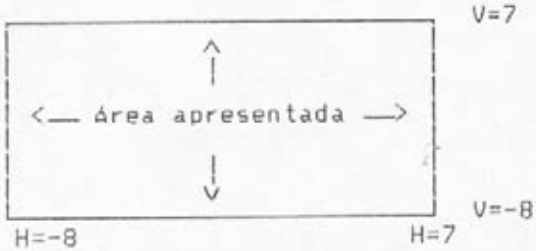
• AJUSTE DA LOCALIZAÇÃO DA TELA

R#18 é usado para ajustar a localização da tela. Corresponde à instrução SET ADJUST do BASIC.

Veja a ilustração na página seguinte.

R#18 - | ± | V | V | V | ± | H | H | H |

Ajuste horizontal (-8 a +7)
Ajuste vertical (-8 a +7)



- TROCANDO O NÚMERO DE PONTOS NA DIREÇÃO Y (VERTICAL)

O número de pontos na direção Y pode ser escolhido entre 192 ou 212, através do bit 7 de R#9. Esta função só é válida no modo texto 2 e nos modos gráficos 4 a 9.

R#9 - | x | • | • | • | • | • | • | • |

0 = 192 pontos
1 = 212 pontos

- FREQÜÊNCIA DE INTERRUPÇÃO

A freqüência de interrupção no MSX é controlada pelo VDP e pode ser de 50 Hz ou 60 Hz. A freqüência de 60 Hz é usada para o sistema NTSC no Japão e no sistema PAL-M brasileiro. A de 50 Hz é usada para o sistema PAL-N europeu.

R#9 - | • | • | • | • | • | • | x | • |

0 = 60 Hz
1 = 50 Hz

- TROCANDO AS PÁGINAS DE VÍDEO

Nos modos gráficos 4 a 9, as páginas em apresentação podem ser trocadas modificando o endereço de início da tabela de padrões.

Observe as ilustrações abaixo e na página seguinte.

Modos gráficos 4 e 5:

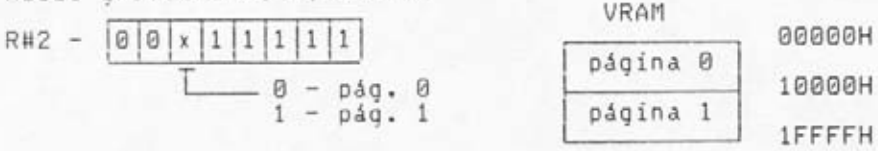
R#2 - | 0 | x | x | 1 | 1 | 1 | 1 | 1 |

00 - pág. 0
01 - pág. 1
10 - pág. 2
11 - pág. 3

VRAM

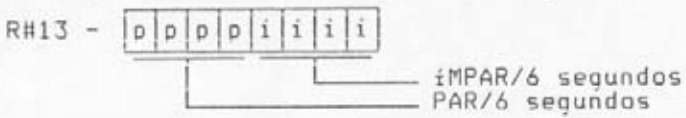
	00000H
página 0	
	08000H
página 1	
	10000H
página 2	
	18000H
página 3	
	1FFFFH

Modos gráficos 6, 7, 8 e 9.



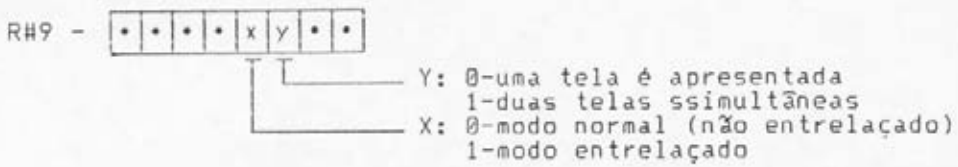
• FUNÇÃO DE TROCA AUTOMÁTICA DE TELA

Nos modos gráficos 4 a 9, duas páginas podem ser apresentadas alternadamente. As páginas 0 e 1 ou 2 e 3 podem usar este recurso. Para iniciar a troca automática de telas, selecione a página ímpar (1 ou 3) usando R#2 e regule o tempo de troca em R#13. Os quatro bits mais altos de R#13 representam o tempo para a página par e os quatro bits mais baixos para a página ímpar. O período de tempo usado é de 1/6 de segundo. Colocando o valor 0 para o período de tempo, apenas a página ímpar será mostrada.

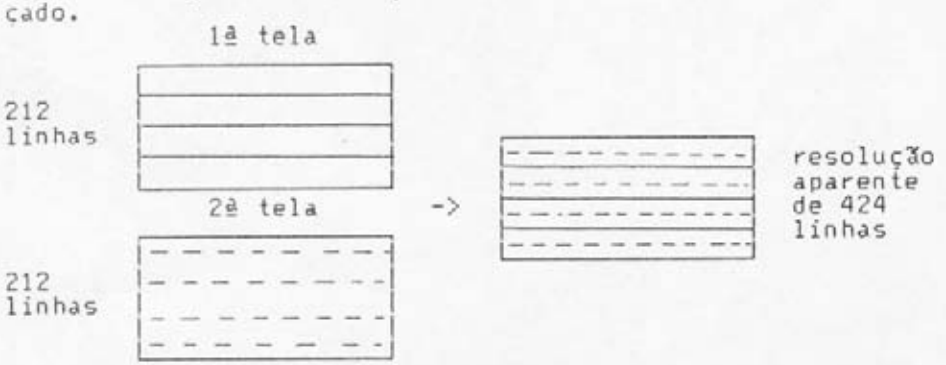


• USANDO O MODO ENTRELAÇADO

O modo entrelaçado pode ser usado para se ter uma resolução vertical aparente de 424 linhas. Isso é feito alternando em alta velocidade duas páginas de vídeo e mostrando apenas a metade da altura de cada linha dessas páginas. As duas páginas são trocadas 60 vezes por segundo. Quando uma página ímpar é selecionada nos modos gráficos 4 a 9, a troca de telas ou páginas é normalmente lenta, feitas em unidades de 1/6 de segundo. De qualquer forma, combinando essa função com o modo entrelaçado, o número aparente de pontos verticais será o dobro. O modo entrelaçado é selecionado por R#9.



Veja na ilustração abaixo como funciona o modo entrelaçado.



A primeira e a segunda telas são apresentadas alternadamente a intervalos de 1/60 de segundo a cada ciclo.

• SCROLL VERTICAL

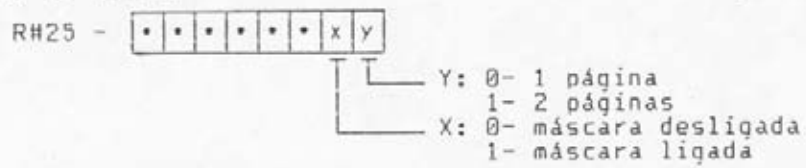
O registro R#23 é usado para indicar a linha inicial da tela. Trocando-se o valor deste registro, pode-se fazer um scroll vertical muito suave. Note que como o scroll é feito para 256 linhas, a tabela de sprites poderá aparecer e ser movida para outra página.

• SCROLL HORIZONTAL (VDP V9958 ou superior)

O scroll horizontal é suportado pelo MSX2+ ou superior. Ele é controlado por R#26 e R#27, sempre considerando que a tela tem 256 pontos em cada linha horizontal, mesmo nos modos gráficos 5 e 6.

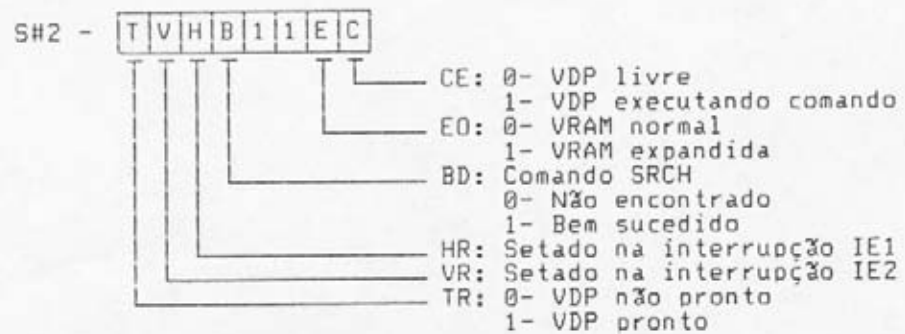
O registrador R#26 pode variar de 1 a 32 (00000001 a 00100000) e cada incremento corresponde a um deslocamento de 8 pontos na tela. Já R#27 pode variar de 7 a 0 (00000111 a 00000000) sendo que cada decremento corresponde ao deslocamento de um ponto na tela. O importante é que quando R#26 é incrementado, R#27 deve ser decrementado e vice-versa.

Já o bit 0 de R#25 determina se o scroll será feito com duas páginas consecutivas ou não. Se for 0, o scroll será feito apenas com uma página de video. Se for 1, o scroll será feito com duas páginas, sendo que a página que está sendo exibida deve ser ímpar. O bit 1 de R#25 determina a ligação de uma máscara que cobre as 8 colunas da esquerda da tela. Se for 0, a máscara estará desligada e se for 1 estará ligada. A cor da máscara será igual à cor da borda.



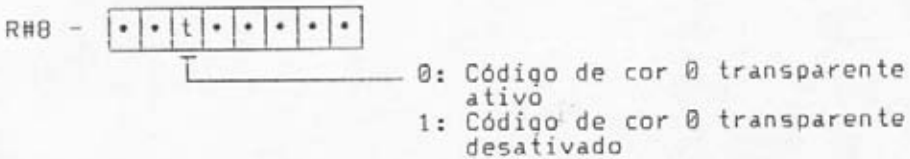
• O REGISTRADOR DE INFORMAÇÃO E CONTROLE

O registrador S#2 é o registrador de informações para controle dos comandos de hardware do VDP. Sua organização é a seguinte:



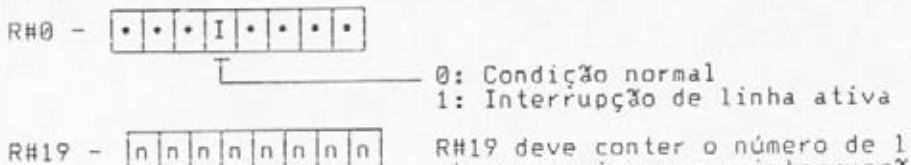
• ESPECIFICANDO O CÓDIGO DE COR 0

Das 16 cores da paleta, a cor 0 é transparente, ou seja, não pode ser definida uma cor para ela e qualquer objeto desenhado com ela não será visto. Entretando, setando o bit 5 de R#8, a função de transparente será desativada e a cor 0 poderá ser definida por P#0.

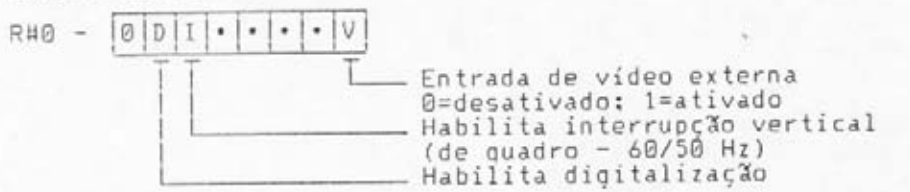


• GERANDO INTERRUPÇÃO POR VARREDURA DE LINHA

No MSX-VIDEO, uma interrupção pode ser gerada quando termina a varredura de uma linha específica da tela. Para isso, basta colocar em R#19 o número da linha que deverá gerar a interrupção e setar o bit 4 de R#0.



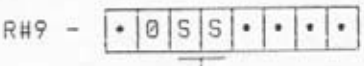
• O REGISTRADOR DE MODO 0



• O REGISTRADOR DE MODO 2

• MODOS DE SINCRONIZAÇÃO

Os modos de sincronização são selecionados pelo registrador de modo 3, conforme ilustração na página seguinte.



Modo de sincronização:
00 - interna
01 - mixada
10 - externa (digitalização)
11 - sem sincronização

5 - SPRITES

Sprites são padrões ou desenhos móveis de 8x8 ou 16x16 pontos na tela. Eles são usados principalmente em jogos.

Existem dois modos de sprites no MSX2. O modo 1 é compatível com o VDP TMS9918A do MSX1. O modo 2 inclui algumas funções novas que foram implementadas nos VDPs V9938 e V9958.

5.1 - FUNÇÃO DOS SPRITES

Até 32 sprites podem ser apresentados simultaneamente na tela. Eles têm dois tamanhos: 8 x 8 e 16 x 16 pontos. Apenas um tamanho pode ser apresentado na tela ao mesmo tempo. O tamanho de um ponto do sprite é normalmente do tamanho de um ponto da tela, mas nos modos gráficos 5 e 6 (que tem resolução de 512x212), o tamanho horizontal é de dois pontos da tela, de forma que o tamanho absoluto do sprite é sempre o mesmo em qualquer modo de tela.

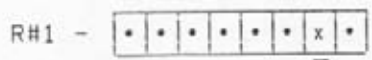
O modo do sprite é automaticamente selecionado de acordo com a screen em uso. Para Graphic 1, Graphic 2 e Multicor, o modo 1 é selecionado e para os modos gráficos 3 a 9 é selecionado o modo 2.

5.2 - SPRITES MODO 1

Os sprites modo 1 são exatamente iguais aos sprites do MSX1. Podem haver na tela até 32 sprites numerados de 0 a 31. Os sprites de números mais baixos têm prioridade de apresentação mais alta. Quando os sprites são colocados na mesma linha horizontal, até 4 sprites são apresentados de acordo com a prioridade, e a parte do 5º sprite e maiores coexistentes na mesma linha horizontal não são mostradas. Veja a ilustração:

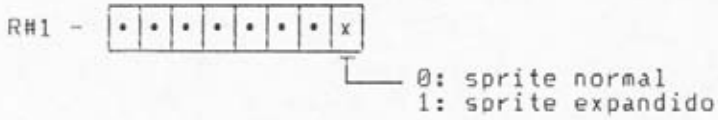
Sprites modo 1

O tamanho dos sprites, de 8x8 ou 16x16, é selecionado pelo bit 1 de R#1. O tamanho default é 8x8 pontos.

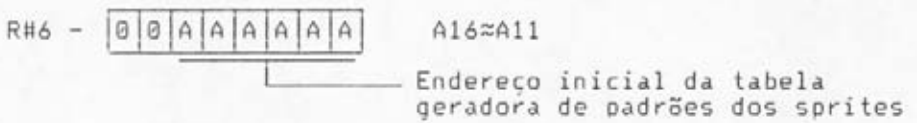


0: 8x8 pontos
1: 16x16 pontos

Os sprites também podem ser expandidos para o dobro do tamanho, na vertical e na horizontal, sendo que nesse caso um ponto do sprite corresponde a quatro pontos na tela. Esta função é controlada pelo bit 0 de R#1.



Os padrões dos sprites são definidos na VRAM. Até 256 sprites podem ser definidos se o tamanho for 8x8, e até 64 se o tamanho for 16x16. Os padrões são numerados de 0 a 255 e alocados na VRAM. Para formar um sprite 16x16 são usados 4 sprites 8x8. O endereço da tabela de padrões dos sprites é especificado em R#6 e tem 2048 bytes, reservando 8 bytes para cada padrão.



Cada sprite é apresentado por um dos 32 planos dos sprites e cada plano é representado por 4 bytes, na *Tabela de Atributos dos Sprites*. O endereço inicial desta tabela é especificado em R#15 e R#11. Os quatro bytes da tabela de atributos contêm as seguintes informações:

- •Coordenada Y: Especifica a coordenada vertical do sprite. Note que a linha de topo da tela não é 0, mas 255. Colocando este valor em 208 (D0H), todos os sprites após esse plano não são mostrados.
- •Coordenada X: Especifica a coordenada horizontal do sprite.
- •Nº do padrão: Especifica qual caractere da tabela geradora de padrões dos sprites será apresentado.
- •Código cor: Especifica a cor, de acordo com a paleta, dos bits setados em "1" na tabela geradora de padrões.
- •EC: Setando em "1" esse bit, os sprites são deslocados 32 pontos à esquerda da coordenada especificada.

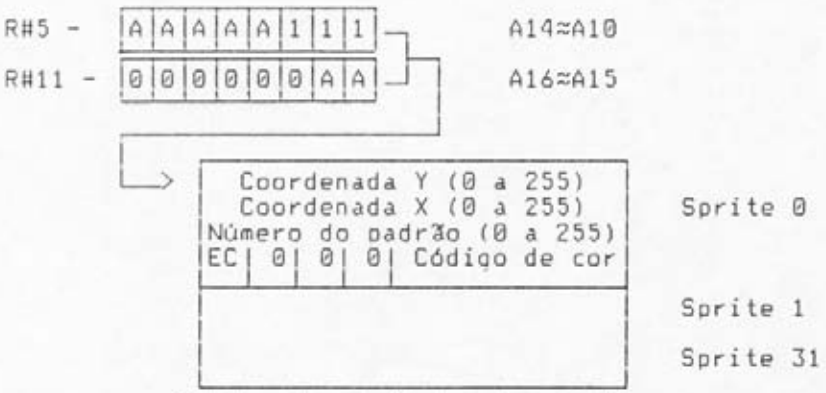
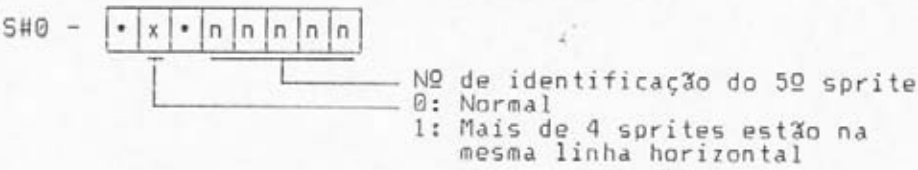


Tabela de atributos dos sprites

Quando dois sprites se sobrepõem na tela, o bit 5 de S#0 é setado informando a situação. A informação de sobreposição ou conflito somente acontece quando os bits "1" se encontram, ou seja, quando a parte "desenhada" dos sprites se sobrepõem.

Quando mais de quatro sprites são colocados na mesma linha horizontal, o bit 6 de S#0 é setado e o número do 5º sprite é colocado nos cinco bits mais baixos de S#0.



5.3 - SPRITES MODO 2

Os sprites modo 2 foram adicionados ao VDP V9938 trazendo novas características e maior flexibilidade que os sprites modo 1.

O número máximo de sprites que podem ser apresentados simultaneamente é de 32, e até 8 sprites podem ocupar a mesma linha horizontal. Os sprites de número menor têm prioridade maior, como no modo 1.

O tamanho do sprite (8x8 ou 16x16) e a expansão para o dobro do tamanho são setados da mesma forma que para os sprites modo 1.

Os sprites modo 2 dispõem de uma função de liga-desliga a apresentação na tela, controlada pelo bit 1 de R#8. Quando este bit for 0, os sprites aparecerão normalmente na tela, mas quando for 1, nenhum sprite aparecerá.

A tabela geradora de padrões é setada da mesma forma que para os sprites modo 1, mas a tabela de atributos sofreu mudanças.

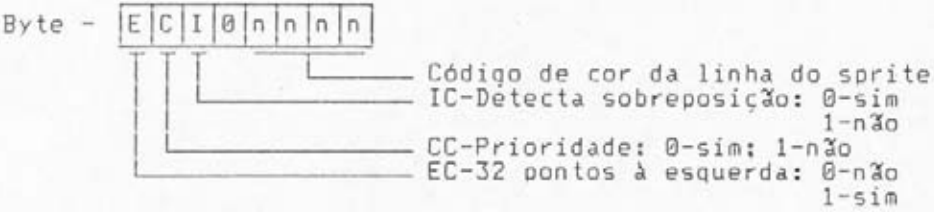
No sprite modo 2, uma cor diferente pode ser especificada para cada linha do sprite. Essa informação é armazenada na *Tabela de Cores dos Sprites*, que é independente da tabela de atributos. A tabela de atributos armazena o seguinte:

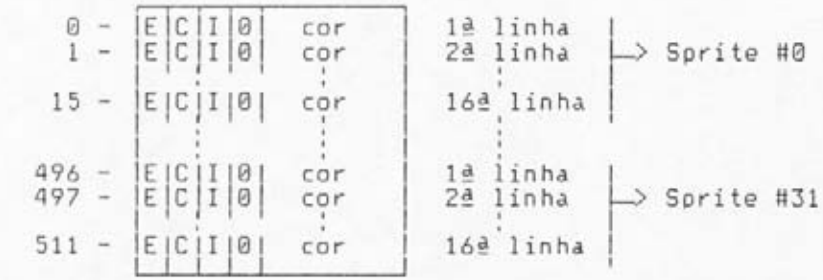
- Coordenada Y: Coordenada vertical do sprite. Colocando em 216 (D8H), os sprites de prioridade menor não serão mostrados. No restante, é igual aos sprites modo 1.
- Coordenada X: Mesmo que no sprite modo 1.
- Nº do padrão: Mesmo que no sprite modo 1.

Tabela de atributos dos sprites modo 2

A tabela de cores dos sprites é automaticamente setada em um endereço 512 bytes antes do endereço inicial da tabela de atributos. 16 bytes são alocados para cada plano dos sprites e cada linha de cada sprite contém as seguintes informações:

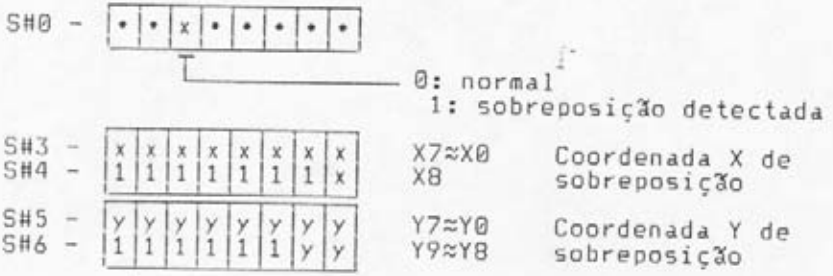
- •Código de cor: Uma cor pode ser especificada para cada linha do sprite.
- •EC: Mesma função do bit EC modo 1, mas apenas a linha especificada será deslocada 32 pontos à esquerda quando este bit for "1".
- •CC: Quando este bit for "1", este sprite terá a mesma prioridade que os sprites de prioridade maior. Quando os sprites de mesma prioridade se sobrepõem, é feita uma operação lógica OR entre as cores dos sprites para determinar a nova cor. Nesse caso a sobreposição não causa conflito e não é detectada.
- •IC: Quando este bit for "1", a linha respectiva do sprite não causará conflito quando ocorrer sobreposição com outros sprites.



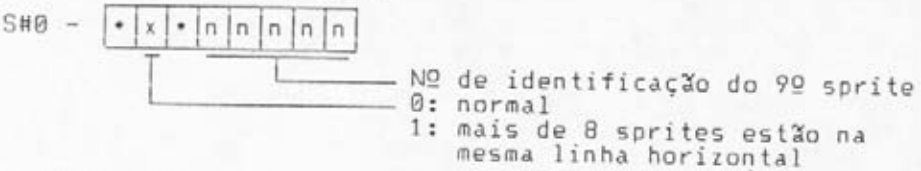


A sobreposição ou conflito de sprites modo 2 é detectada quando a cor do ponto do sprite não é transparente e os bits CC dos sprites forem 0. Quando a sobreposição é detectada, o bit 5 de S#0 é setado em "1" e a coordenada da sobreposição é colocada em S#3 a S#6, conforme a ilustração abaixo. Observe que a coordenada obtida por estes registros não é aquela onde o conflito atual ocorreu. Para obter as coordenadas exatas, use a seguinte expressão:

Coordenada X = (S#3 e S#4) - 12
Coordenada Y = (S#5 e S#6) - 8



Quando mais de 8 sprites são colocados na mesma linha horizontal, o bit 6 de S#0 é setado em "1" e o número do plano do sprite de menor prioridade é colocado nos 5 bits mais baixos de S#0, conforme a ilustração.



6 - COMANDOS DO VDP

O MSX-VIDEO pode executar operações gráficas básicas, chamadas de *Comandos do VDP*. São executados por hardware e estão disponíveis para os modos gráficos 4 a 9. Quando os comandos do VDP são executados, a localização dos pontos de início e destino são representados por coordenadas (X,Y) e não há divisão de páginas de vídeo, sendo os 128 Kbytes de VRAM tratados como um único bloco.

GRAPHIC 4 (SCREEN 5)			**ENDEREÇO**	GRAPHIC 5 (SCREEN 6)		
(0,0)		(255,0)	00000H	(0,0)		(511,0)
	Página 0				Página 0	
(0,255)		(255,255)	07FFFH	(0,255)		(511,255)
(0,256)		(255,256)		(0,256)		(511,256)
	Página 1				Página 1	
(0,511)		(255,511)	0FFFFH	(0,511)		(511,511)
(0,512)		(255,512)		(0,512)		(511,512)
	Página 2				Página 2	
(0,767)		(255,767)	17FFFH	(0,767)		(511,767)
(0,768)		(255,768)		(0,768)		(511,768)
	Página 3				Página 3	
(0,1023)		(255,1023)	1FFFFH	(0,1023)		(511,1023)

GRAPHIC 7≈9 (SCREEN 8≈12) GRAPHIC 6 (SCREEN 7)

6.1 - DESCRIÇÃO DOS COMANDOS DO VDP

Existem 12 tipos de comandos do VDP que podem ser executados pelo MSX-VIDEO. Veja a tabela:

NOME COMANDO	DESTINO	ORIGEM	UNIDADE	MEMÔNICO	R#46-4msb
MOVIMENTOS RAPIDOS	VRAM	CPU	bytes	HMMC	1 1 1 1
	VRAM	VRAM	bytes	YMMM	1 1 1 0
	VRAM	VRAM	bytes	HMMM	1 1 0 1
	VRAM	VDP	bytes	HMMV	1 1 0 0
MOVIMENTOS LÓGICOS	VRAM	CPU	pontos	LMMC	1 0 1 1
	CPU	VRAM	pontos	LMCM	1 0 1 0
	VRAM	VRAM	pontos	LMMM	1 0 0 1
	VRAM	VDP	pontos	LMMV	1 0 0 0
LINHA	VRAM	VDP	pontos	LINE	0 1 1 1
PROCURA	VRAM	VDP	pontos	SRCH	0 1 1 0
PSET	VRAM	VDP	pontos	PSET	0 1 0 1
POINT	VDP	VRAM	pontos	POINT	0 1 0 0
RESERVADO	-	-	-	-	0 0 1 1
	-	-	-	-	0 0 1 0
	-	-	-	-	0 0 0 1
PARADA	-	-	-	-	0 0 0 0

Quando um dado é escrito em R#46 (registrador de comando), o VDP começa a executar o comando e seta o bit 0 (CE / command execute) do registrador de status S#2. Os parâmetros necessários devem ser colocados em R#32 a R#45 antes do comando ser executado. Quando a execução do comando termina, o bit 0 de S#2 é resetado (0). Para interromper a execução de um comando, use o comando de parada. Os comandos do VDP só funcionam nos modos gráficos 4 a 9, mas nos modos 8 e 9 devem ser usados com cautela, pois a tela pode borrar, já que nestes modos os pontos estão organizados em blocos de quatro na horizontal.

6.2 - OPERAÇÕES LÓGICAS

Quando os comandos são executados, várias operações lógicas podem ser feitas entre a VRAM e um dado especificado. Essas operações são feitas de acordo com a tabela abaixo.

Na tabela da página seguinte, SC representa a cor de origem e DC a cor de destino. IMP, AND, OR, EOR e NOT escrevem o resultado de cada operação no destino. Nas operações com os nomes precedidos por "T", os pontos de origem que tiverem a cor 0 (SC = 0) não serão objeto de operações lógicas no destino (DC). Usando este recurso, somente as porções coloridas são sobrepostas. Este

recurso é especialmente efetivo para animações.

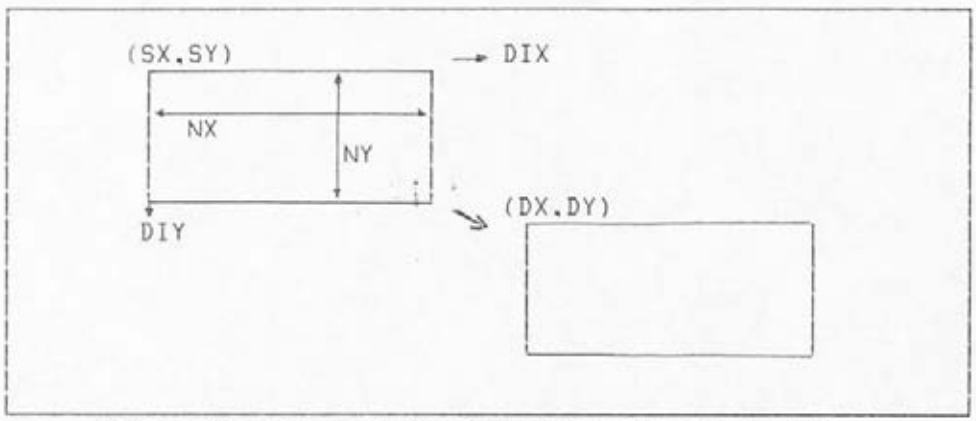
NOME	OPERAÇÃO	R#46-4lsb
IMP	DC = SC	0 0 0 0
AND	DC = SC * DC	0 0 0 1
OR	DC = SC + DC	0 0 1 0
EOR	DC = SC * DC + SC * DC	0 0 1 1
NOT	DC = SC	0 1 0 0
TIMP	Se SC=0, DC=DC senão DC = SC	1 0 0 0
TAND	Se SC=0, DC=DC senão DC = SC*SC	1 0 0 1
TOR	Se SC=0, DC=DC senão DC = SC+DC	1 0 1 0
TEOR	Se SC=0, DC=DC senão DC = SC*DC + SC*DC	1 0 1 1
TNOT	Se SC=0, DC=DC senão DC = SC	1 1 0 0

SC = Código da cor de origem
DC = Código da cor de destino
EOR = OR exclusivo

6.3 - ESPECIFICAÇÃO DE ÁREAS

Os comandos de movimentação de áreas transferem os dados do vídeo dentro de uma área especificada por um retângulo. A área a ser transferida é especificada em um vértice, a partir do qual são dados o tamanho dos lados do retângulo juntamente com a direção em que os dados serão transferidos e as coordenadas de destino.

SX e SY são as coordenadas de origem; NX e NY são o comprimento de cada lado do retângulo em pontos e DIX e DIY especificam a direção na qual os dados serão transferidos e dependem do tipo de comando. DX e DY especificam o vértice de destino.



Especificação de áreas para os comandos do VDP

6.4 - USANDO OS COMANDOS

Os comandos do VDP são classificados em três tipos: comandos de transferência rápida (high-speed transfer), comandos de transferência lógica (logical transfer) e comandos de desenho.

Os comandos devem ser acessados por via direta; por isso deve ser tomado um certo cuidado com a sincronização. O acesso às portas de I/O deve ser feito através de uma leitura dos endereços 0006H e 0007H, usando o registrador C, como demonstrado abaixo:

```
RDVDP:  EQU  00006H
WRVDP:  EQU  00007H
        LD   A,(RDVDP)
        LD   C,A          ;leitura da VRAM
        INC  C            ;leitura do reg. de estado
         :
        LD   A,(WRVDP)
        LD   C,A          ;escrita na VRAM
        INC  C            ;escrita no reg. de controle
        INC  C            ;escrita nos regs. de paleta
        INC  C            ;escrita no reg. especif. indireto
```

Para a espera do VDP, pode ser usada a seguinte rotina:

```
RDVDP:  EQU  00006H
WRVDP:  EQU  00007H
        :
        ;--- ESPERA VDP ---
        ;
WAIT:   LD   A,2
        CALL STATUS
        AND  1
        JR   NZ,WAIT
        XOR  A
        CALL STATUS
        RET
        :
STATUS: PUSH BC
        LD   BC,(WRVDP)
        INC  C
        OUT  (C),A
        LD   A,08FH
        OUT  (C),A
        LD   BC,(RDVDP)
        INC  C
        IN   A,(C)
        POP  BC
        RET
```

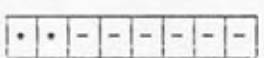
6.4.1 - HMMC (transferência rápida - CPU -> VRAM)

Neste comando, os dados são transferidos da CPU para uma área especificada na VRAM. Operações lógicas não são possíveis; os dados são transferidos em bytes em alta velocidade. Note que os bits baixos nos modos gráficos 4 a 6 não podem servir como referência; apenas os bits mais altos devem ser considerados. Veja a ilustração abaixo:

GRAPHIC 4:

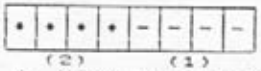
Como o byte da VRAM representa dois pontos, os 4 bits mais baixos da coordenada X não serão representados.

GRAPHIC 5:



Como o byte da VRAM representa quatro pontos, os 2 bits mais baixos da coordenada X não serão representados.

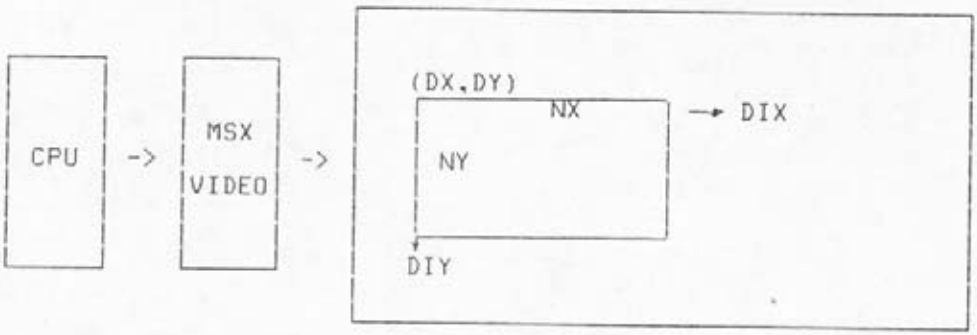
GRAPHIC 6:



Como o byte da VRAM representa dois pontos, os 4 bits mais baixos da coordenada X não serão representados.

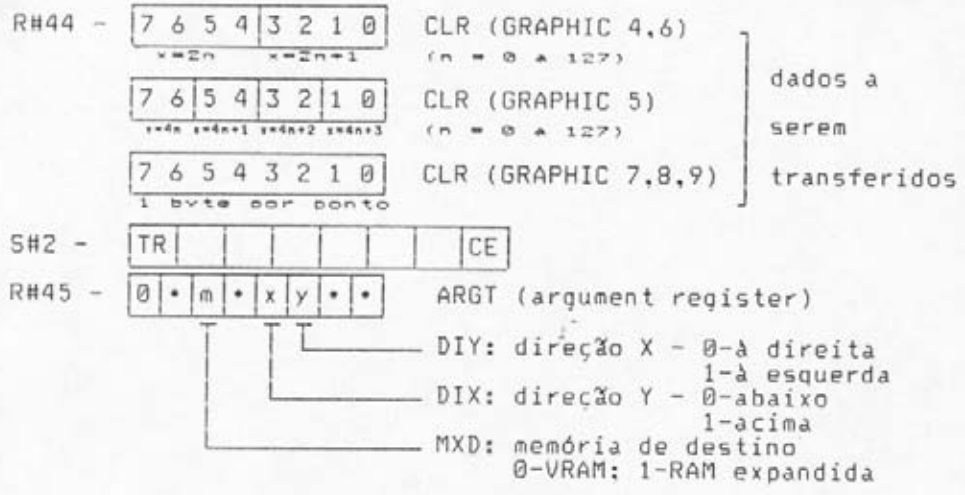
Coloque os parâmetros mostrados na tabela nos registros apropriados. Neste ponto, escreva o primeiro byte de dados a ser transferido para a CPU em R#44. Para executar, escreva o código de comando F0H em R#46 e o byte contido em R#44 será escrito na VRAM. Depois, o VDP espera o próximo dado da CPU.

A CPU escreve o segundo byte de dados em R#44, e assim por diante. Note que o dado só será transferido depois que o VDP recebê-lo (se o bit TR for "1"), referindo ao bit TR de S#2. Quando o bit CE de S#2 for 0, isso significa que todos os dados foram transferidos. Observe as ilustrações para entender melhor.

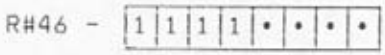


MXD: Seleciona a memória de destino. 0-VRAM; 1-RAM expandida
NX: Número de pontos a transferir na direção X (0 a 511)
NY: Número de pontos a transferir na direção Y (0 a 1023)
DIX: Direção de NX a partir da origem. 0-à direita
1-à esquerda
DIY: Direção de NY a partir da origem. 0-abaixo
1-acima
DX: Coordenada X de destino na tela (0 a 511)
DY: Coordenada Y de destino na tela (0 a 1023)
CLR (R#44): 1º byte de dados a ser transferido

R#36 -	x x x x x x x x	X7≈X0	DX	Coordenadas da tela para onde os dados começarão a serem transferidos
R#37 -	0 0 0 0 0 0 0 x	X8		
R#38 -	y y y y y y y y	Y7≈Y0	DY	
R#39 -	0 0 0 0 0 0 y y	Y9≈Y8		
R#40 -	x x x x x x x x	X7≈X0	NX	Nº de pontos a transferir na direção X
R#41 -	0 0 0 0 0 0 0 x	X8		
R#42 -	y y y y y y y y	Y7≈Y0	NY	Nº de pontos a transferir na direção Y
R#43 -	0 0 0 0 0 0 y y	Y9≈Y8		



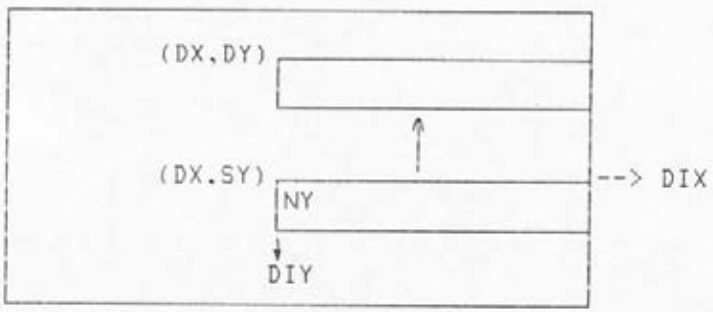
Para executar o comando HMMC. devemos setar R#46 da seguinte forma:



6.4.2 - YMMM (transferência rápida - VRAM na direção Y)

Neste comando, os dados de uma área especifica da VRAM são transferidos para outra área da VRAM. Observe que este comando transfere os dados apenas na direção Y (vertical).

Depois de colocar os dados nos registros apropriados, é só escrever o código do comando E0H em R#46 para executá-lo. Enquanto o bit CE de S#2 for 1, o comando estará sendo executado.



MXD: seleciona a memória de destino. 0-VRAM: 1-RAM expandida
SY: coordenada Y de origem (0 a 1023)
NY: Nº de pontos a transferir na direção Y (0 a 1023)
DIX: seleciona a direção de transferência. 0-para a direita até o limite da tela; 1-para a esquerda até o limite da tela
DIY: seleciona a direção vertical a partir da origem. 0-abaixo; 1-acima
DX: Coordenada X de inicio e de destino (0 a 511) *
DY: Coordenada Y inicial de destino (0 a 1023)

* Para os modos gráficos 4 a 6, o bit mais baixo em DX é ignorado e para o modo gráfico 5, são ignorados os dois bits mais baixos.

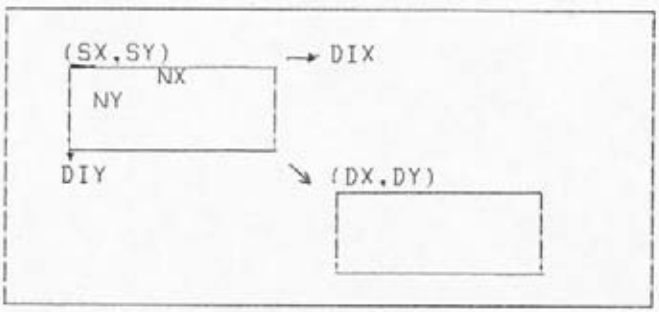
```
R#34 - | y | y | y | y | y | y | y | y |   Y7≈Y0   SY: coordenada Y de
R#35 - | 0 | 0 | 0 | 0 | 0 | 0 | y | y |   Y9≈Y8       origem

R#36 - | x | x | x | x | x | x | x | x |   X0≈X7   DX: coordenada X de
R#37 - | 0 | 0 | 0 | 0 | 0 | 0 | 0 | x |   X8          origem e de destino

R#38 - | y | y | y | y | y | y | y | y |   Y7≈Y0   DY: coordenada Y de
R#39 - | 0 | 0 | 0 | 0 | 0 | 0 | y | y |   Y9≈Y8       destino

R#42 - | y | y | y | y | y | y | y | y |   Y7≈Y0   NY: nº de pontos a trans-
R#43 - | 0 | 0 | 0 | 0 | 0 | 0 | y | y |   Y9≈Y8       ferir na direção Y

R#45 - | • | • | m | • | y | x | • | • |   ARGT (registrador de argumento)
                    |   |   |
                    |   |   +-- direção X (0-à direita; 1-à esq.)
                    |   +------ direção Y (0-abaixo; 1-acima)
                    +---------- seleciona memória de destino:
                                0-VRAM; 1-RAM expandida

S#2 - | TR |  |  |  |  |  |  | CE |
```

Para executar o comando YMMM, após carregar os registradores, basta escrever o valor E0H em R#46.

```
R#46 - | 1 | 1 | 1 | 0 | • | • | • | • |
```

6.4.3 - HMMM (transferência rápida - VRAM -> VRAM)

Neste comando, os dados são transferidos de uma área da VRAM para outra. Depois de colocar os parâmetros nos registradores, é só escrever o valor D0H em R#46 para executar o comando. Enquanto o comando estiver sendo executado, o bit CE de S#2 ficará setado em 1. Veja a ilustração abaixo.



MXS: seleciona memória de origem. 0-VRAM; 1-RAM expandida
MXD: seleciona memória de destino. 0-VRAM; 1-RAM expandida
SX: coordenada X de origem (0 a 511) *
SY: coordenada Y de origem (0 a 1023)
NX: nº de pontos a transferir na direção X (0 a 511) *
NY: nº de pontos a transferir na direção Y (0 a 1023)
DIX: direção de NX a partir da origem. 0-à direita; 1-à esq.
DIY: direção de NY a partir da origem. 0-abaixo; 1-acima

DX: coordenada X de destino na tela (0 a 511) *
DY: coordenada Y de destino na tela (0 a 1023)
* Para os modos gráficos 4 e 6, o bit mais baixo em DX, SX e NX é ignorado e para o modo gráfico 5 são ignorados os dois bits mais baixos.

```
R#32 - | x | x | x | x | x | x | x | x |   X7≈X0   SX ┐
R#33 - | 0 | 0 | 0 | 0 | 0 | 0 | 0 | x |   X8         │ Coordenadas
                                                      │ de origem
R#34 - | y | y | y | y | y | y | y | y |   Y7≈Y0   SY ┘
R#35 - | 0 | 0 | 0 | 0 | 0 | 0 | y | y |   Y9≈Y8

R#36 - | x | x | x | x | x | x | x | x |   X7≈X0   DX ┐
R#37 - | 0 | 0 | 0 | 0 | 0 | 0 | 0 | x |   X8         │ Coordenadas
                                                      │ de destino
R#38 - | y | y | y | y | y | y | y | y |   Y7≈Y0   DY ┘
R#39 - | 0 | 0 | 0 | 0 | 0 | 0 | y | y |   Y9≈Y8

R#40 - | x | x | x | x | x | x | x | x |   X7≈X0   NX   Nº de pontos a trans-
R#41 - | 0 | 0 | 0 | 0 | 0 | 0 | 0 | x |   X8           ferir na direção X

R#42 - | y | y | y | y | y | y | y | y |   Y7≈Y0   NY   Nº de pontos a trans-
R#43 - | 0 | 0 | 0 | 0 | 0 | 0 | y | y |   Y9≈Y8        ferir na direção Y

R#45 - | • | • | d | s | y | x | • | • |   ARGT (registrador de argumento)
```

DIX: direção X - 0-à direita
1-à esquerda
DIY: direção Y - 0-abaixo
1-acima
MXS: seleciona memória de origem
0-VRAM; 1-RAM expandida
MXD: seleciona memória de destino
0-VRAM; 1-RAM expandida

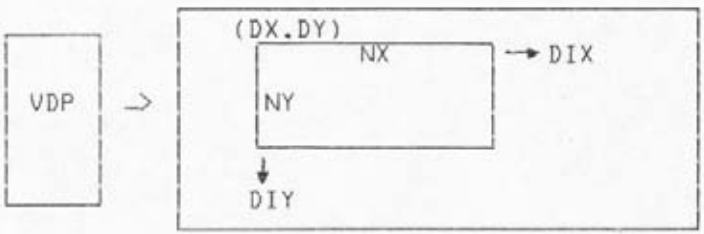
```
S#2 - | TR |   |   |   |   |   |   | CE |
```

Para executar o comando HMMM, devemos colocar o valor D0H em R#46.

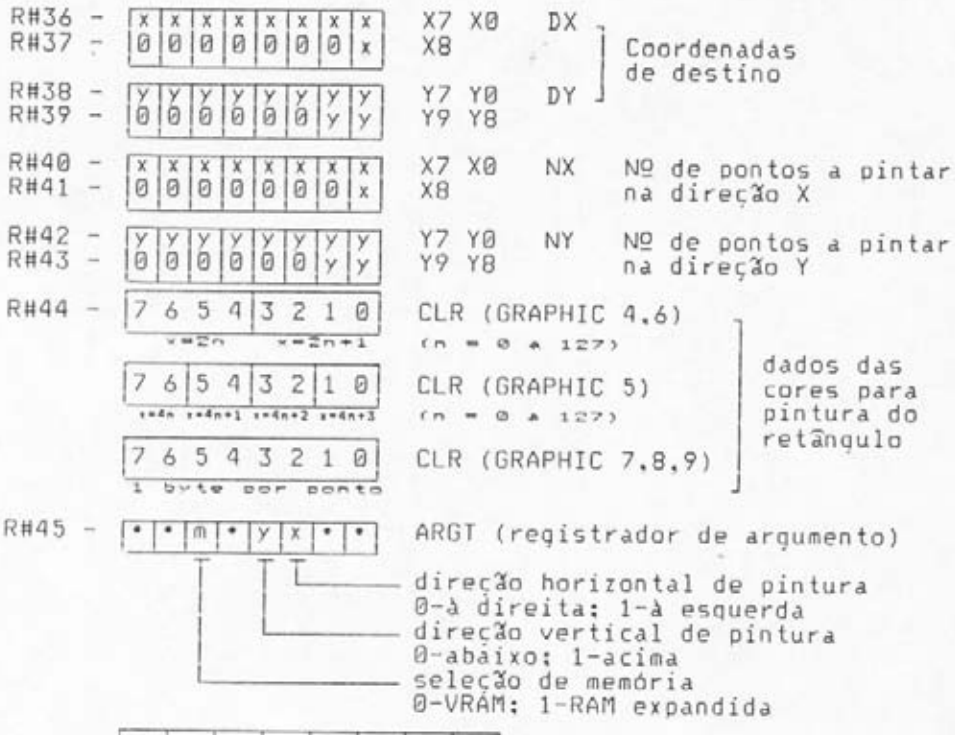
```
R#46 - | 1 | 1 | 0 | 1 | • | • | • | • |
```

6.4.4 - HMMV (desenha retângulo em alta velocidade)

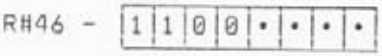
Neste comando, cada byte de dados especificado é desenhado na VRAM com o código de cor respectivo, depois de colocar os parâmetros para desenhar o retângulo nos registradores adequados. Para executar o comando, deve-se escrever o valor C0H em R#46. Enquanto o comando estiver sendo executado, o bit CE de S#2 fica setado em "1".



MXD: seleciona memória. 0-VRAM: 1-RAM expandida
NX: nº de pontos a pintar na direção X (0 a 511) *
NY: nº de pontos a pintar na direção Y (0 a 1023)
DIX: direção de NX a partir da origem. 0-à direita: 1-à esq.
DIY: direção de NY a partir da origem. 0-abaixo: 1-acima
DX: coordenada X de origem (0 a 511) *
DY: coordenada Y de origem (0 a 1023)
CLR (R#44): registrador de cor
* Para os modos gráficos 4 e 6, o bit mais baixo em DX e NX é ignorado e para o modo gráfico 5 são ignorados os dois bits mais baixos.



Para executar a pintura do retângulo, devemos colocar o valor de comando C0H em R#46.

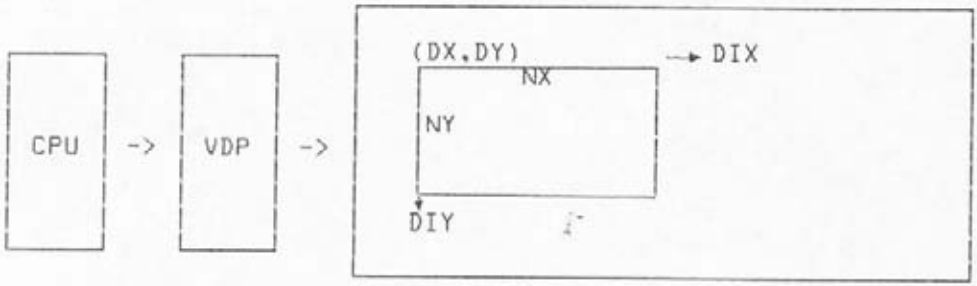


6.4.5 - LMMC (transferência lógica - CPU -> VRAM)

Neste comando, os dados são transferidos da CPU para uma área específica da VRAM em pontos. Operações lógicas durante a transferência são possíveis. Nos comandos de transferência lógica, como o LMMC, os dados são transferidos em pontos e um byte é requerido para cada ponto em todos os modos de tela.

Depois de carregar os registradores do VDP com os dados requeridos, execute o comando escrevendo o valor B0H em R#46. As operações lógicas podem ser especificadas usando os 4 bits mais

baixos do registrador de comando R#46. Os dados são transferidos tendo como referência os bits TR e CE de S#2, como nos comandos de transferência rápida.



MXD: seleciona memória de destino. 0-VRAM; 1-RAM expandida
NX: número de pontos a transferir na direção X (0 a 511)
NY: número de pontos a transferir na direção Y (0 a 1023)
DIX: direção de NX a partir da origem. 0-à direita; 1-à esq.
DIY: direção de NY a partir da origem. 0-abaixo; 1-acima
DX: coordenada X de destino (0 a 511)
DY: coordenada Y de destino (0 a 1023)
CLR (R#44): primeiro byte ou ponto a ser transferido

R#36 -	x	x	x	x	x	x	x	x	X7≈X0	DX
R#37 -	0	0	0	0	0	0	0	x	X8	
R#38 -	y	y	y	y	y	y	y	y	Y7≈Y0	DY
R#39 -	0	0	0	0	0	0	y	y	Y9≈Y9	
R#40 -	x	x	x	x	x	x	x	x	X7≈X0	NX
R#41 -	0	0	0	0	0	0	0	x	X8	
R#42 -	y	y	y	y	y	y	y	y	Y7≈Y0	NY
R#43 -	0	0	0	0	0	0	y	y	Y9≈Y8	

Coordenadas de destino

nº de pontos a transferir na direção X

nº de pontos a transferir na direção Y

R#44 -	•	•	•	•	3	2	1	0	CLR (GRAPHIC 4,6)
	•	•	•	•	•	•	1	0	CLR (GRAPHIC 5)
	7	6	5	4	3	2	1	0	CLR (GRAPHIC 7,8,9)

dados a transferir

R#45 -	•	•	m	•	y	x	•	•

ARGT (registrador de argumento)

- direção X (0-à direita: 1-à esq.)
- direção Y (0-abaixo; 1-acima)
- memória de destino: 0 - VRAM; 1 - RAM expandida

S#2 -	TR							CE

Para executar o comando LMMC. devemos escrever o valor B0H em R#46 (10110000B). Os quatro bits mais baixos de R#46 podem conter a operação lógica a ser executada no destino.

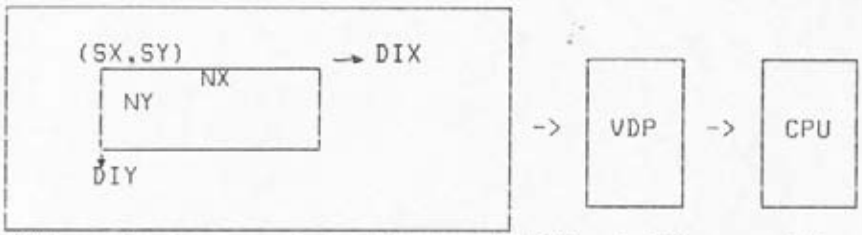
R#46 -	1	0	1	1	c	c	c	c

- código de operação lógica
- comando de execução LMMC

6.4.6 - LMCM (transferência lógica - VRAM -> CPU)

Neste comando, os dados são transferidos de uma área especificada na VRAM para a CPU em pontos.

Depois de colocar os dados nos registros apropriados, basta escrever o valor A0H em R#46 para o comando ser executado e os dados transferidos para a CPU. Para isso, a CPU deve verificar o bit TR de S#2 e se esse bit for 1, a CPU pode ler o dado contido em S#7.Quando o bit CE de S#2 for 0, os dados a serem transferidos terminaram.



MXS: seleciona memória fonte: 0-VRAM; 1-RAM expandida
SX: seleciona coordenada X de origem (0 a 511)
SY: seleciona coordenada Y de origem (0 a 1023)
NX: nº de pontos a transferir na direção X (0 a 511)
NY: nº de pontos a transferir na direção Y (0 a 1023)
DIX: direção de NX a partir da origem. 0-à direita; 1-à esq.
DIY: direção de NY a partir da origem. 0-abaixo; 1-acima

R#32 -	x	x	x	x	x	x	x	x	X7≈X0	SX
R#33 -	0	0	0	0	0	0	0	x	X8	
R#34 -	y	y	y	y	y	y	y	y	Y7≈Y0	NX
R#35 -	0	0	0	0	0	0	y	y	Y9≈Y8	

coordenadas de origem

R#40 -	x	x	x	x	x	x	x	x	X7≈X0	NX
R#41 -	0	0	0	0	0	0	0	x	X8	

nº de pontos a transferir na direção X

R#42 -	y	y	y	y	y	y	y	y	Y7≈Y0	NY
R#43 -	0	0	0	0	0	0	y	y	Y9≈Y8	

nº de pontos a transferir na direção Y

R#45 -	•	•	•	m	y	x	•	•

ARGT (registrador de argumento)

- x: direção X (0-à direita; 1-à esq.)
- y: direção Y (0-abaixo; 1-acima)
- m: MXS: seleciona memória fonte 0-VRAM; 1-RAM expandida

S#2 -	TR							CE

Para executar o comando LMCM, basta escrever o valor A0H em R#46.

R#46 -	1	0	1	0	•	•	•	•

CMR

S#7 -	•	•	•	•	3	2	1	0
	•	•	•	•	•	•	1	0
	7	6	5	4	3	2	1	0

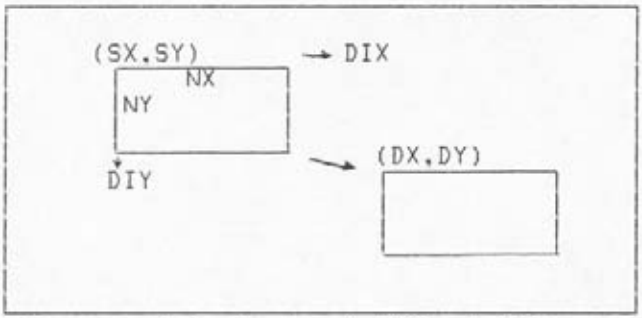
GRAPHIC 4,6
GRAPHIC 5
GRAPHIC 7,8,9

Nota 1: Ler o registro S#7 ao preencher os registros, para que o bit TR seja resetado antes da execução do comando.
Nota 2: Quando o último dado for escrito em S#7 e o bit TR for 1, o comando será terminado pelo VDP e o bit CE será resetado.

6.4.7 - LMMM (transferência lógica - VRAM -> VRAM)

Neste comando, os dados especificados em uma área da VRAM podem ser transferidos para outra área da VRAM em pontos.

Depois de colocar os parâmetros nos registradores adequados, escreva o código de comando 9xH (x é o código de operação lógica) em R#46 para executar. Enquanto o bit CE de S#2 for 1, o comando estará sendo executado.



MXS: seleciona memória fonte. 0-VRAM; 1-RAM expandida
MXD: seleciona memória destino. 0-VRAM; 1-RAM expandida
SX: coordenada X de origem (0 a 511)
SY: coordenada Y de origem (0 a 1023)
NX: nº de pontos a transferir na direção X (0 a 511)
NY: nº de pontos a transferir na direção Y (0 a 1023)
DIX: direção de NX a partir da origem. 0-à direita; 1-à esq.
DIY: direção de NY a partir da origem. 0-abaixo; 1-acima
DX: coordenada X de destino (0 a 511)
DY: coordenada Y de destino (0 a 1023)

R#32	|x|x|x|x|x|x|x|x|	X7≈X0	SX	coordenadas de origem
R#33	|0|0|0|0|0|0|0|x|	X8		
R#34	|y|y|y|y|y|y|y|y|	Y7≈Y0	SY	
R#35	|0|0|0|0|0|0|y|y|	Y9≈Y8		
R#36	|x|x|x|x|x|x|x|x|	X7≈X0	DX	coordenadas de destino
R#37	|0|0|0|0|0|0|0|x|	X8		
R#38	|y|y|y|y|y|y|y|y|	Y7≈Y0	DY	
R#39	|0|0|0|0|0|0|y|y|	Y9≈Y8		
R#40	|x|x|x|x|x|x|x|x|	X7≈X0	NX	nº de pontos a transferir na direção X
R#41	|0|0|0|0|0|0|0|x|	X8		
R#42	|y|y|y|y|y|y|y|y|	Y7≈Y0	NY	nº de pontos a transferir na direção Y
R#43	|0|0|0|0|0|0|y|y|	Y9≈Y8		

R#45 - | * | * | d | s | y | x | * | * | ARGT (registrador de argumento)

- direção X (0-à direita; 1-à esq.)
- direção Y (0-abaixo; 1-acima)
- MXS: seleciona memória fonte 0-VRAM; 1-RAM expandida
- MXD: seleciona memória destino 0-VRAM; 1-RAM expandida

S#2 - | TR | | | | | | | CE |

Para executar o comando LMMM, deve-se escrever em R#46 o valor 90H, sendo que os quatro bits mais baixos podem conter um código de operação lógica.

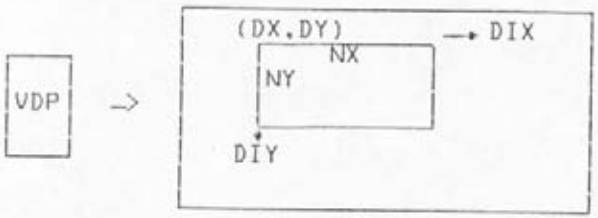
R#46 - | 1 | 0 | 0 | 1 | c | c | c | c | CMR

- código de operação lógica
- comando LMMM

6.4.8 - LMMV (pintura lógica da VRAM)

Uma área especifica da VRAM pode ser pintada em pontos, e operações lógicas entre a VRAM e o dado especificado são possiveis.

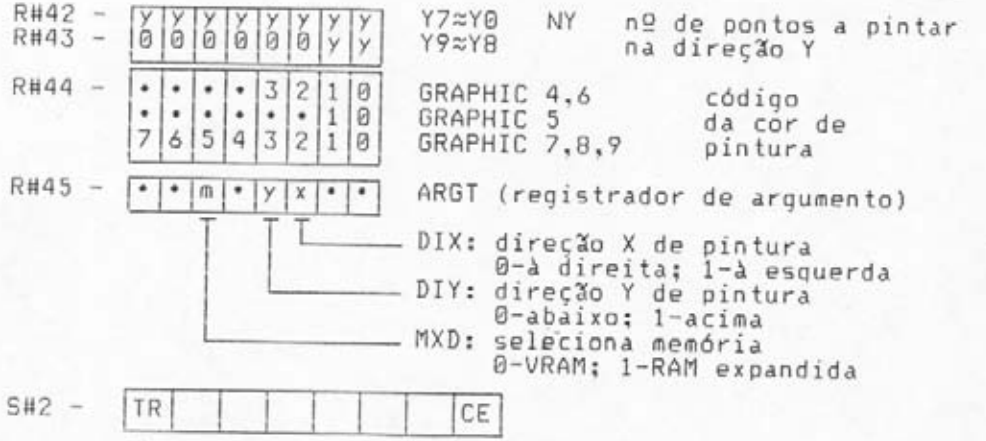
Depois de setar os parâmetros nos registradores adequados, basta escrever o valor 8xH (x é o código de operação lógica) em R#46 para executar o comando. Enquanto o bit CE de S#2 for 1, o comando estará sendo executado.



MXD: seleciona memória. 0-VRAM; 1-RAM expandida
NX: número de pontos a pintar na direção X (0 a 511)
NY: número de pontos a pintar na direção Y (0 a 1023)
DIX: direção de NX a partir da origem. 0-à direita; 1-à esq.
DIY: direção de NY a partir da origem. 0-abaixo; 1-acima
DX: coordenada X de início da pintura (0 a 511)
DY: coordenada Y de início da pintura (0 a 1023)
CLR (R#44) - dados para a pintura

R#36 - | x | x | x | x | x | x | x | x | X7≈X0 | DX
R#37 - | 0 | 0 | 0 | 0 | 0 | 0 | 0 | x | X8 | DX
coordenadas de início da pintura
R#38 - | y | y | y | y | y | y | y | y | Y7≈Y0 | DY
R#39 - | 0 | 0 | 0 | 0 | 0 | 0 | y | y | Y9≈Y8 | DY

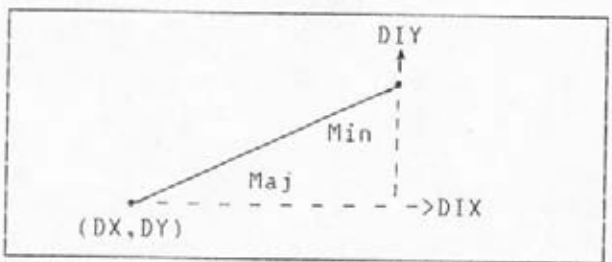
R#40 - | x | x | x | x | x | x | x | x | X7≈X0 | NX | nº de pontos a pintar na direção X
R#41 - | 0 | 0 | 0 | 0 | 0 | 0 | 0 | x | X8 | NX



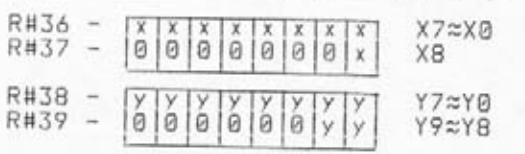
Para executar o comando LMMV, deve-se escrever o valor 8xH (x é o código de operação lógica) em R#46, sendo que os quatro bits mais baixos podem conter um código de operação lógica.

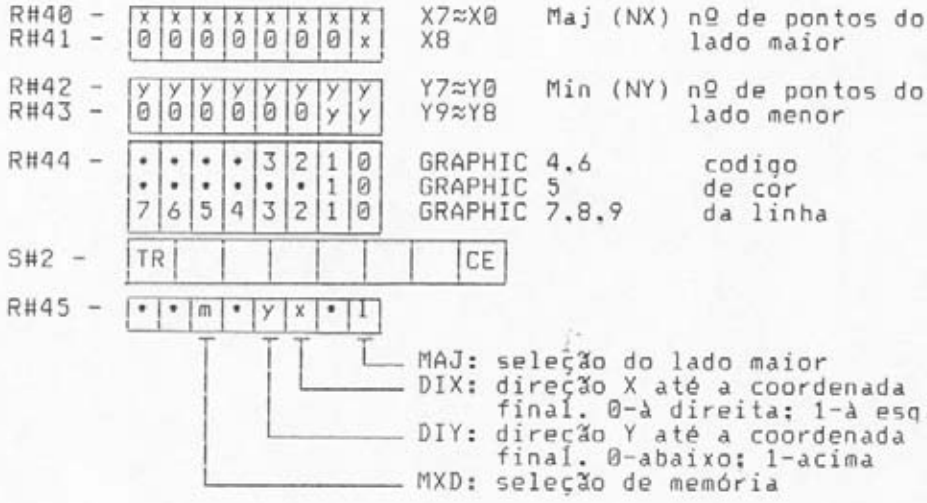
6.4.9 - LINE (desenhando uma linha)

As linhas podem ser desenhadas entre coordenadas na VRAM. Os parâmetros são especificados incluindo a coordenada (X,Y) de início e o comprimento de X e Y até o ponto final. Veja a ilustração abaixo.



MXD: seleciona memória. 0-VRAM; 1-RAM expandida
Maj: número de pontos do lado maior (0 a 1023)
Min: número de pontos do lado menor (0 a 511)
MAJ: 0 se o lado maior for paralelo ao eixo X
1 se o lado maior for paralelo ao eixo Y ou igual ao lado menor
DIX: direção X para desenho da reta. 0-à direita; 1-à esq.
DIY: direção Y para desenho da reta. 0-abaixo; 1-acima
DX: coordenada X de origem (0 a 511)
DY: coordenada Y de origem (0 a 1023)





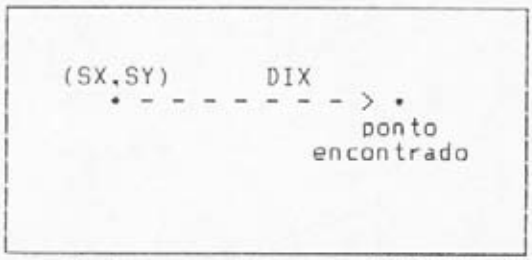
Para executar o comando LINE, deve-se escrever o valor 7xH em R#46, sendo que "x" pode representar um código de operação lógica.

6.4.10 - SRCH (procura código de cor)

Este comando procura a existência de uma cor especifica a partir de uma coordenada na VRAM, sempre na horizontal, para a direita ou para a esquerda. é um comando útil para as rotinas de pintura.

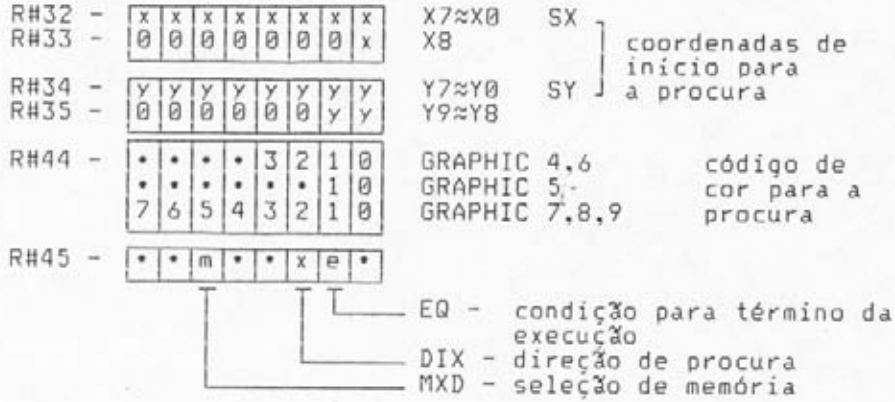
Depois de colocar os parâmetros nos registradores adequados, basta escrever o comando 60H em R#46 para executar o comando. O comando termina quando o ponto da cor especificada é encontrado ou quando a borda da tela é atingida. Enquanto o bit CE de S#2 for 1, o comando estará sendo executado.

Terminado o comando, se o ponto com a cor foi encontrado, o bit BD de S#2 será 1 e a coordenada do ponto estará armazenada em S#8 e S#9.

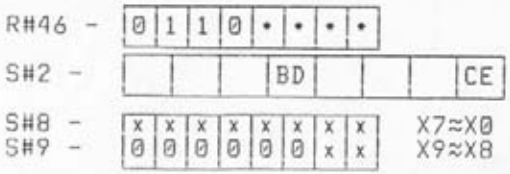


MXD: seleciona memória para procura. 0-VRAM; 1-RAM expandida
SX: coordenada X para início da procura (0 a 511)
SY: coordenada Y para início da procura (0 a 1023)

DIX: direção de procura. 0-à direita; 1-à esquerda
EQ: 0-termina a execução quando a borda é encontrada;
1-termina a execução quando um ponto com a cor especificada é encontrado.
CLR (R#44): código de cor



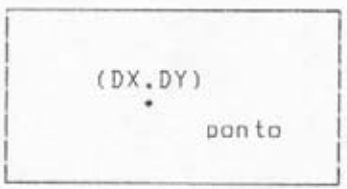
Para executar o comando SRCH, basta escrever o valor 60H no registrador R#46.



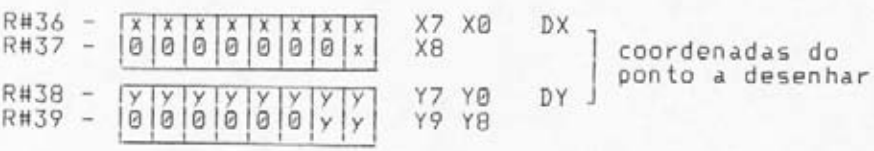
6.4.11 - PSET (desenhando um ponto)

Através desse comando, pode-se desenhar um ponto em qualquer coordenada da VRAM. Depois de colocar os parâmetros nos registradores adequados, basta escrever o valor 5xH em R#46 para executar o comando (x é o código de operação lógica).

Enquanto o bit CE de S#2 for 1, o comando estará sendo executado. Veja a ilustração abaixo.



MXD: seleciona memória. 0-VRAM; 1-RAM expandida
DX: coordenada X do ponto (0 a 511)
DY: coordenada Y do ponto (0 a 1023)



R#44 -	•	•	•	•	3	2	1	0	GRAPHIC 4,6	código
	•	•	•	•	•	•	1	0	GRAPHIC 5	de cor
	7	6	5	4	3	2	1	0	GRAPHIC 7,8,9	do ponto

R#45 -	•	•	m	•	•	•	•	•	ARGT (registrador de argumento)

m: seleção de memória
0-VRAM; 1-RAM expandida

Para executar o comando PSET, basta escrever o valor 5xH em R#46, onde "x" é o código de operação lógica.

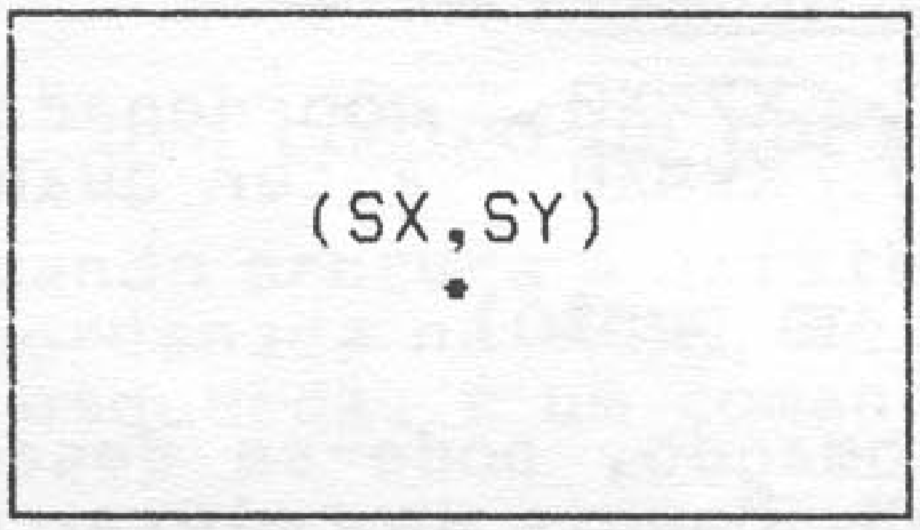
R#46 -	0	1	0	1	c	c	c	c	CMR

c c c c: código de operação lógica
0 1 0 1: comando PSET

6.4.12 - POINT (ler o código de cor de um ponto)

O comando POINT lê o código de cor de um ponto em qualquer coordenada da VRAM.

Depois de colocar os parâmetros nos registradores adequados, basta escrever o valor 40H em R#46 para executar o comando. Enquanto o bit CE de S#2 for 1, o comando estará sendo executado. O código de cor do ponto ficará armazenado em S#7.



MXS: seleção de memória. 0-VRAM; 1-RAM expandida
SX: coordenada X do ponto (0 a 511)
SY: coordenada Y do ponto (0 a 1023)

R#32 -	x	x	x	x	x	x	x	x	X7 X0	SX	coordenadas do ponto
R#33 -	0	0	0	0	0	0	0	x	X8		
R#34 -	y	y	y	y	y	y	y	y	Y7 Y0	SY	
R#35 -	0	0	0	0	0	0	y	y	Y9 Y8		

R#45 -	•	•	•	m	•	•	•	•	ARGT (registrador de argumento)

m: seleção de memória
0-VRAM; 1-RAM expandida

Para executar o comando POINT, basta escrever o valor 40H em R#46. O código de cor retornará em S#7.

R#46 -	0	1	0	0	•	•	•	•	CMR

S#7 -	•	•	•	•	3	2	1	0	GRAPHIC 4,6	código de cor
	•	•	•	•	•	•	1	0	GRAPHIC 5	do ponto
	7	6	5	4	3	2	1	0	GRAPHIC 7,8,9	especificado

6.5 - TORNANDO OS COMANDOS MAIS RÁPIDOS

A estrutura do VDP permite que várias outras tarefas seja executadas enquanto um comando estiver ativo. Às vezes, a execução de alguns desses comandos fica lenta devido a isso. Se essas funções forem destivadas, a execução do comando ficará mais rápida.

• INIBIÇÃO DA APRESENTAÇÃO DOS SPRITES

Este meio é muito prático e permite um sensível aumento da velocidade quando os sprites são removidos da tela. Para isso, basta setar o bit 1 de R#8 em "1".

R#8 -	•	•	•	•	•	•	1	•

• INIBIÇÃO DE APRESENTAÇÃO DA TELA

Este meio só deve ser usado no caso de inicialização da tela, uma vez que, quando inibida, a tela toda fica com uma só cor. Para tanto, basta setar o bit 6 de R#1 em "1".

R#1 -	•	1	•	•	•	•	•	•

Capítulo 5

GERADORES DE ÁUDIO

Os micros MSX têm várias opções para a geração de sons, incluindo desde geradores de AM simples até digitalizadores. Estas opções estão listadas abaixo:

1- PSG (standard MSX1)
2- 1-bit I/O port (standard MSX1)
3- FM-OPLL (opcional MSX2, standard MSX2+)
4- PCM (standard MSX turbo R)
5- MSX-AUDIO (opcional)

1 - O PSG

PSG significa "Programmable Sound Generator", ou seja, *Gerador de Sons Programável*. O PSG pode gerar três vozes em até 4096 escalas (equivalente a 8 oitavas) e 16 níveis de volume independente para cada voz. Adicionalmente, possui um gerador de ruído branco (chiado), mas que deve estar presente em uma das três vozes. O chip responsável é o AY-3-8910A.

O PSG tem 16 registradores de 8 bits cada para a geração de sons. Na tabela abaixo, estes estão descritos. Note que os registradores 14 e 15 são usados para operações de I/O e não para a especificação de sons.

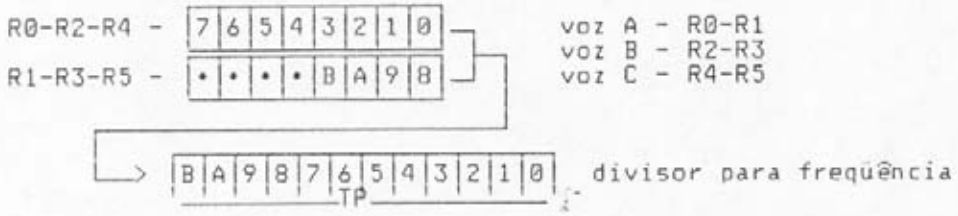
	7 6 5 4 3 2 1 0	
R0 -	a a a a a a a a	freqüência da
R1 -	• • • • a a a a	voz A
R2 -	b b b b b b b b	freqüência da
R3 -	• • • • b b b b	voz B
R4 -	c c c c c c c c	freqüência da
R5 -	• • • • c c c c	voz C
R6 -	• • • r r r r r	freqüência do ruído branco
R7 -	i o r r r t t t	habilita / desabilita
R8 -	• • • m v v v v	volume da voz A
R9 -	• • • m v v v v	volume da voz B
R10 -	• • • m v v v v	volume da voz C
R11 -	f f f f f f f f	freqüência
R12 -	f f f f f f f f	da envoltória
R13 -	• • • • e e e e	forma da envoltória
R14 -	i i i i i i i i	porta A de I/O
R15 -	o o o o o o o o	porta B de I/O

1.1 - DESCRIÇÃO DOS REGISTRADORES E FUNCIONAMENTO

• ESPECIFICAÇÃO DA FREQÜÊNCIA

A freqüência central usada pelo PSG para comandar o di-

visor de freqüências é de 111860,78 Hz. Assim, para obter a freqüência de saída do gerador de tons, basta dividir 111860,78 pelo valor TP, representado pelos pares de registradores R0-R1, R2-R3 e R4-R5. Veja a ilustração abaixo.

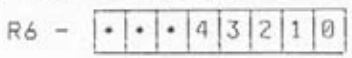


Os valores de cada registro TP para as 8 oitavas dos três geradores de tom estão listados na tabela abaixo.

Cifrado		1	2	3	4	5	6	7	8
Dó	C	D5D	6AF	357	1AC	0D6	06B	035	01B
	C#	C9C	64E	327	194	0CA	085	032	019
Ré	D	BE7	5F4	2FA	17D	0BE	05F	030	018
	D#	B3C	59E	2CF	168	0B4	05A	02D	016
Mi	E	A9B	54E	2A7	153	0AA	055	02A	015
Fá	F	A02	501	281	140	0A0	050	028	014
	F#	973	4BA	25D	12E	097	04C	026	013
Sol	G	8EB	476	23B	11D	08F	047	024	012
	G#	86B	436	21B	10D	087	043	022	011
Lá	A	7F2	3F9	1FD	0FE	07F	040	020	010
	A#	780	3C0	1E0	0F0	078	03C	01E	00F
Si	B	714	38A	1C5	0E3	071	039	01C	00E

• O GERADOR DE RUíDO BRANCO

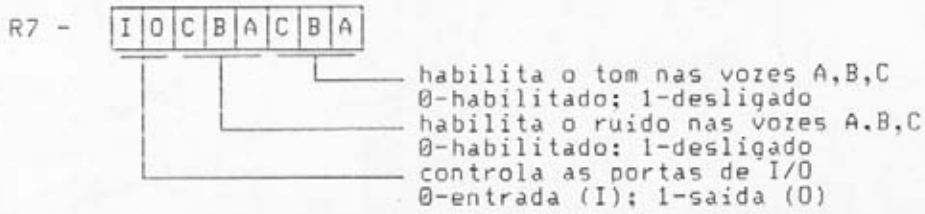
O gerador de chiado é útil para gerar sons de explosões e outros. O PSG gera o chiado através de uma das três vozes de tom e sua freqüência é especificada no registro R6. São usados apenas os cinco primeiros bits, que contêm um valor que serve de divisor de freqüência para o chiado.



A freqüência central usada pelo gerador de ruído também é de 111860.78 Hz. Como o valor de R6 pode variar de 1 a 31, a freqüência do ruído varia de 3.6 KHz a 111,8 KHz.

• MIXANDO OS SONS

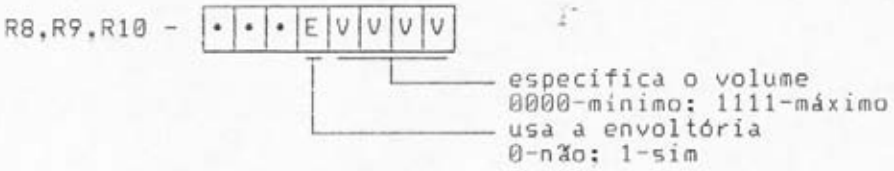
O registrador R7 é usado para habilitar ou desabilitar o som ou o ruído de cada uma das três vozes e também para controlar as operações de I/O do PSG.



Os dois bits mais altos de R7 não afetam o som. Eles controlam a direção dos dados das duas portas de 8 bits do PSG. No MSX, a porta A é usada como entrada e a B como saída; portanto o bit 6 de R7 é sempre 1 e o bit 7 é sempre 0.

• AJUSTANDO O VOLUME

Os registradores R8 a R10 são usados para especificar o volume de cada uma das três vozes e pode variar de 0 (volume mínimo) a 15 (volume máximo), ou então entregar o controle de volume ao gerador de envoltória. (R8-voz A; R9-voz B; R10-voz C).



Quando o bit E for 0, o volume é controlado pelo programador através dos quatro primeiros bits. Quando o bit E for 1, o volume é controlado pelo gerador de envoltória e os quatro primeiros bits são ignorados.

• AJUSTE DA FREQÜÊNCIA DA ENVOLTÓRIA

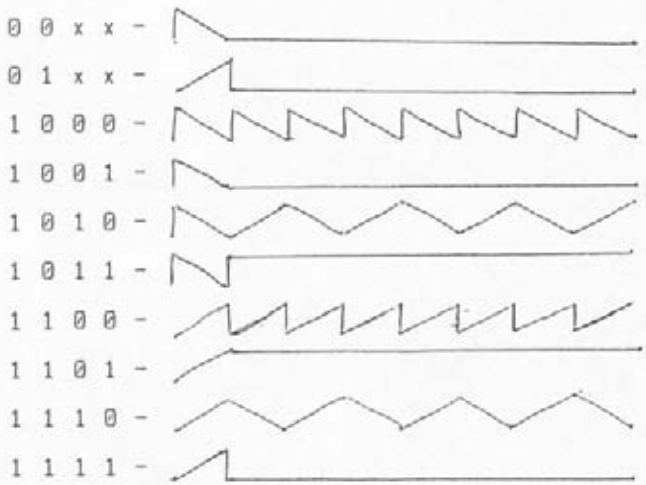
Os registradores R11 e R12 são usados como divisor de freqüência para o gerador de envoltória. Como todos os bits dos dois registradores são válidos, a freqüência é estipulada em 16 bits.

A freqüência central usada pelo gerador de envoltória para comandar o divisor de freqüências é de 6983,3 Hz; portanto a freqüência da envoltória pode variar de 6983,3 Hz a 0,107 Hz. Veja a ilustração abaixo.

• SELECIONANDO A FORMA DA ENVOLTÓRIA

A forma da envoltória é especificada nos quatro primeiros bits de R13. O intervalo "T" é o especificado nos registradores R11 e R12.

Veja na página seguinte os valores de R13 com as respectivas formas de onda envelope.



1.2 - O ACESSO AO PSG

O acesso ao PSG pela CPU é feito por portas de I/O. Entretanto, o padrão MSX determina que todos os acessos ao PSG devem ser feitos através de rotinas do BIOS, evitando, assim, problemas de sincronização ou "timing". As rotinas são as seguintes:

```
WRTPSG (0093H/Main)
  Função:  escreve um byte de dados em um registro do PSG
  Entrada: A - número do registro do PSG
           E - byte de dados a ser escrito
  Saída:   nenhuma

RDPSG  (0096H/Main)
  Função:  lê um byte de dados de um registro do PSG
  Entrada: A - número do registro do PSG a ser lido
  Saída:   A - byte lido
```

2 - GERAÇÃO DE SOM PELA PORTA 1-bit

O padrão MSX dispõe de outro método standard para a geração de sons. Estes são gerados ligando e desligando repetidamente uma porta de I/O de 1 bit. Este bit é responsável pelo "click" das teclas. O acesso a este bit também é feito por uma rotina do BIOS:

```
GHGSND (0135H/Main)
  Entrada: A - 0-desliga o bit; outro valor, liga o bit
  Saída:   nenhuma
```

Ligando e desligando repetidamente este bit, podem ser gerados diversos tipos de efeitos sonoros, inclusive reprodução grosseira da voz humana.

3 - O GERADOR FM (OPLL)

O gerador de sons FM pode gerar 9 vozes simultâneas ou 6 vozes simultâneas mais 5 peças de bateria. Sua qualidade sonora é muito superior à do PSG. O FM também é conhecido como OPLL, do inglês "FM OPerator type LL". O chip responsável pela geração de

sons FM é o YM2413 e surgiu como alternativa barata para o MSX-AUDIO, sendo standard para o MSX2+ em diante.

O OPLL possui 37 registradores de 8 bits designados pelos endereços $00H≈$07H, $0EH≈$18H, $20H≈$28H e $30H≈$38H.

Reg.	7	6	5	4	3	2	1	0	
$00H	AM	VIB	EGT	KSR	Múltiplo				(m)
$01H	AM	VIB	EGT	KSR	Múltiplo				(c)
$02H	KSL(m)		Nível total modul.(c)						Registros para instrumento do usuário
$03H	KSL(c)		•	DC	DM	Feedback			
$04H	Attack(m)				Decay(m)				
$05H	Attack(c)				Decay(c)				
$06H	Sustain(m)				Release(m)				(m)=modulador
$07H	Sustain(c)				Release(c)				(c)=carrier
$0EH	•	•	R	BD	SD	TOM	TCY	HH	Controle de bateria
$0FH	Modo do OPLL								Teste do OPLL
$10H	Freqüência LSB 8 bits								
$18H									
$20H	•	•	S	K	Oitava			F	Freq. MSB 1 bit
	•	•	u	e				r	Oitava
	•	•	s	y				e	Reg. Key on/off
$28H	•	•	t					q	Reg. Sustain on/off
$30H	Instrumentos				Volume				Registradores para seleção de instrumentos e de volume
$38H									

Mapa dos registradores para o modo bateria ($0EH, b5=1):

$36H	•	•	•	•	BD-vol	Registradores de volume da bateria
$37H	HH-vol				SD-vol	
$38H	TOM-vol				TCY-vol	

Conteúdo dos registradores:

Reg.	Bit	Conteúdo
00:(m)	b7	Liga/desliga a modulação de amplitude
01:(c)	b6	Liga/desliga o vibrato
	b5	0-tom percussivo; 1-tom constante
	b4	Razão da "Key Scale"
	b0≈b3	Controle multi-sample e harmônicos
02,03	b6≈b7	Nível da "Key Scale" - $02(m); $03(c)
02	b0≈b5	Nível total de modulação

03	b4:(c) b3:(m) b0≈b2	Distorção da forma de onda carrier Distorção da forma de onda modulada Constante de realimentação FM (m)
04:(m) 05:(c)	b4≈b7 b0≈b3	Controle do nível de "attack" da envoltória Controle do nível de "decay" da envoltória
06:(m) 07:(c)	b4≈b7 b0≈b3	Indicação de "decay"; nível de "sustain" Controle do nível "release" da envoltória
0E	b5 b0≈b4	1-modo bateria; 0-modo de melodia Liga/desliga instrumentos da bateria
10≈18	b0≈b7	Freqüência (LSB 8 bits)
20≈28	b5 b4 b1≈b3 b0	Liga/desliga o "Sustain" Liga/desliga a "Key" Seleciona a oitava Freqüência (MSB 1 bit)
30≈38	b4≈b7 b0≈b3	Seleção de instrumentos Controle de volume

O OPLL possui internamente 15 instrumentos pré-programados e mais um que pode ser definido pelo usuário, além de cinco peças de bateria. O instrumento que pode ser programado é o de número 0 (Original). Os instrumentos são os seguintes:

0: Original	8: órgão	Bateria:
1: Violino	9: Piston	
2: Violão	10: Sintetizador	BD: Bass drum
3: Piano	11: Cravo	SD: Snare drum
4: Flauta	12: Vibrafone	TOM: Tom-tom
5: Clarinete	13: Baixo elétrico	TCY: Top cymbal
6: Oboé	14: Baixo acústico	HH: High hat
7: Trompete	15: Guitarra elétrica	

3.1 - DESCRIÇÃO DOS REGISTRADORES E FUNCIONAMENTO

• REGISTRADOR DE TESTE

Este registrador é estabelecido somente para teste do chip YM2413. Normalmente seu valor é 0.

3.1.1 - REGISTROS PARA DEFINIÇÃO DE INSTRUMENTO

• AM/VIB/EG-TYP/KSR/MÚLTIPLO ($00H e $01H)

Estes registradores especificam o fator de multiplicação para as freqüências do modulador ($00H) e do carrier ($01H) com os componentes, como a envoltória, etc.

MÚLTIPLO (b0=b3)

As freqüências da onda carrier e da onda modulada, que geram a envoltória, são controladas de acordo com certos fatores de multiplicação, que podem ser vistos na tabela abaixo.

Valor do registro:	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Fator de multipl.:	½	1	2	3	4	5	6	7	8	9	10	10	12	12	15	15

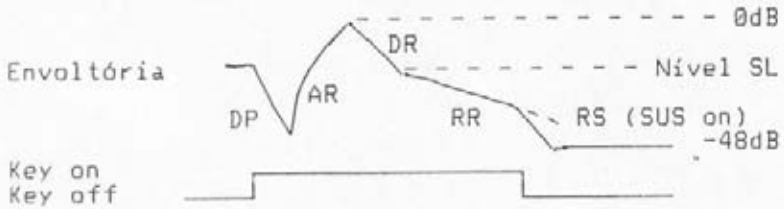
KSR (b4)

Este bit é uma flag que indica se será usada ou não a "Key Scale Rate", especificada pelos bits KSL. Após setar os tons musicais, estes podem ter seu nível alterado. Se KSR for igual a 0, o nível será o mesmo para todas as freqüências. Se KSR for igual a 1, haverá atenuação do som conforme a freqüência; quanto mais alta a freqüência gerada, maior será o nível de atenuação. Veja os bits KSL.

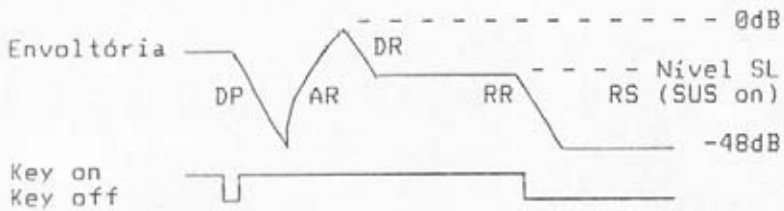
EG-TYP (b5)

Este bit seleciona a envoltória para tom constante ou tom percussivo. Se for 0, o tom será percussivo e se for 1, o tom será constante. Veja as ilustrações abaixo.

Tom percussivo (b5=0)



Tom constante (b5=1)



VIB (b6)

Flag usada para ativar ou desativar o vibrato. Se for 1, o vibrato estará ativo e se for 0, estará desligado. A freqüência do vibrato é de 6.4 Hz.

AM (b7)

Flag usada para ativar ou desativar a modulação de amplitude ou "trêmolo". Se for 1, a modulação de amplitude estará ativa e se for 0, estará desligada. A freqüência para a modulação

de amplitude é de 3,7 Hz.

• KSL/NíVEL TOTAL/DISTORÇÃO/NíVEL DE REALIMENTAÇÃO ($02H,$03H)

O Nível Total é usado para controlar o índice de modulação (tom) pela atenuação do sinal de saída do gerador de envoltória. Os bits KSL (Key Scale Level) são usados para regular a saída de modo que o som gerado pelo OPLL se aproxime dos instrumentos musicais reais.

	b7	b6	b5	b4	b3	b2	b1	b0
$02H -	KSL			Nível total				
$03H -	KSL		•	DC	DM	Realim.		

NíVEL TOTAL (b0≈b5)

Este registro permite controlar o nível de modulação através da atenuação do mesmo (envoltória). Com o valor 000000, não haverá atenuação e a modulação será máxima. Já com o valor 111111, a atenuação será máxima, de aproximadamente 48 dB.

$02H -	•	•	m	m	m	m	m	m

000000 - sem atenuação
000001 - atenuação mínima
111111 - atenuação máxima

KSL (b6≈b7)

Estes bits controlam o nível da "Key Scaling". No modo "Key Scale" (KSR = 1), o nível de atenuação progressiva do som pode variar de 0 dB/oitava até 6 dB/oitava, conforme a tabela.

b7	b6	Atenuação
0	0	0 dB/oitava
0	1	1,5 dB/oitava
1	0	3 dB/oitava
1	1	6 dB/oitava

DM (b3,$03H)

Quando este bit for igual a 1, a onda modulada é retificada para meia onda.

DC (b4,$03H)

Quando este bit for igual a 1, a onda carrier é retificada para meia onda.

REALIMENTAÇÃO (FEEDBACK) (b0≈b2,$03H)

índice de realimentação (porção do sinal de saída que é reinjetado na entrada) para a onda modulada.

001 - realimentação mínima
111 - realimentação máxima

• RELAÇÃO ATTACK/DECAY ($04H-$05H)

As relações de "attack" e "decay" são definidas pelos registradores $04H e $05H, conforme a figura abaixo. Quanto maior o valor, menor o tempo de "attack" e "decay" para a voz respectiva.

	attack (AR)				decay(DR)				
$04H -	b7	b6	b5	b4	b3	b2	b1	b0	modulador
$05H -	b7	b6	b5	b4	b3	b2	b1	b0	carrier

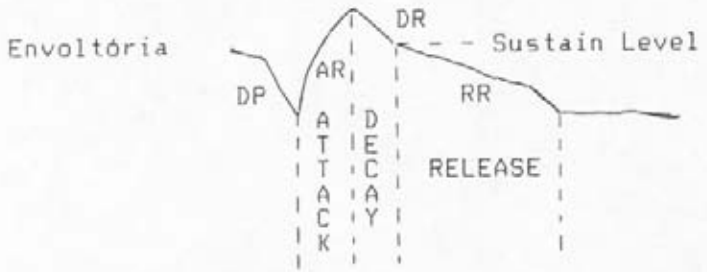
• SUSTAIN LEVEL / RELEASE RATE ($06H,$07H)

"Sustain Level" é o nivel no qual a envoltória permanece após ter sido atenuada pela "Decay Rate". Para o tom percussivo, este é o ponto de troca do modo "decay" para o modo "release". Quanto maior o valor do registro, mais baixo será o nível de "sustain".

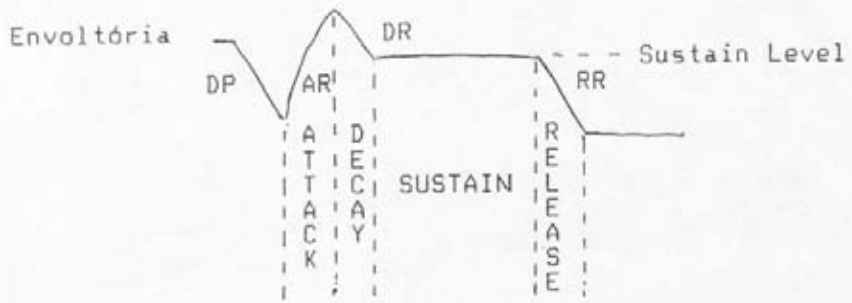
"Release Rate" é a relação de desaparecimento do som após o "Key off". Para o tom percussivo, é expressada pela atenuação após o "Sustain Level". Quanto maior o valor do registro, menor será a duração da "Release Rate".

	sustain				release(RR)				
$06H -	b7	b6	b5	b4	b3	b2	b1	b0	modulador
$07H -	b7	b6	b5	b4	b3	b2	b1	b0	carrier

Tom percussivo



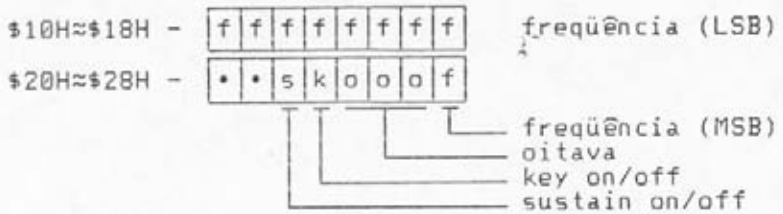
Tom constante



3.1.2 - REGISTRADORES DE SELEÇÃO

• OITAVA/FREQÜÊNCIA ($10H≈$18H, $20H≈$28H)

São nove grupos de dois registradores de 8 bits cada, formando pares, sendo numerados de $10H≈$18H e $20H≈$28H. Assim, os registradores $10H e $20H controlam a 1ª voz, os registradores $11H e $21H controlam a 2ª voz e assim por diante. São estes registradores que definem a freqüência de cada uma das 9 vozes que podem ser geradas pelo OPLL.



FREQÜÊNCIA ($1xH e bit 0 de $2xH)

Estes 9 bits definem uma escala de freqüências para cada oitava. Na tabela abaixo estão especificados os valores dos registadores para a quarta oitava (de um total de 8 oitavas), com a nota LÁ central de 440 Hz.

Cifrado		Freqüência	Registros	$2xH,b0	$1xH
Dó	C#	277,2 Hz	181	0	10110101
Ré	D	293,7 Hz	192	0	11000000
	D#	311,1 Hz	204	0	11001100
Mi	E	329,6 Hz	216	0	11011000
Fá	F	349,2 Hz	229	0	11100101
	F#	370,0 Hz	242	0	11110000
Sol	G	392,0 Hz	257	1	00000001
	G#	415,3 Hz	272	1	00010000
Lá	A	440,0 Hz	288	1	00100000
	A#	466,2 Hz	305	1	00110001
Si	B	493,9 Hz	323	1	01000011
Dó	C	523,3 Hz	343	1	01010111

Os valores das freqüências guardam entre si uma relação geométrica igual à 12ª raiz de 2, que vale 1.0594630943592. Pode-se usar este número para alterar os valores dos registradores a fim de aumentar ou diminuir a freqüência gerada, já que estes guardam entre si a mesma relação que as freqüências.

OITAVA ($2xH, b1≈b3)

Estes três bits definem a oitava. Podem ser definidas até 8 oitavas, com os valores 000 a 111, sendo que a quarta oitava é a de número 011.

KEY ($2xH, b4)

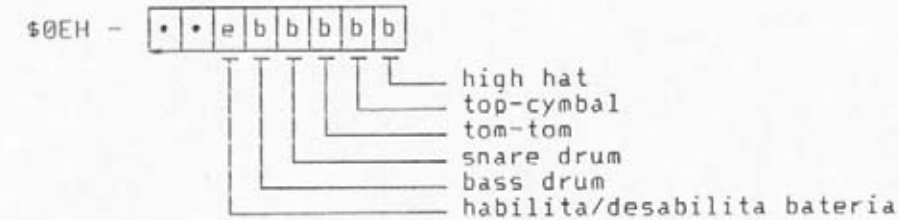
Este bit deve ser setado em "1" para que o som de cada uma das nove vozes seja habilitado. Quando for "0", o som da voz respectiva estará desativado (key off).

SUS ($2xH, b5)

Quando este bit estiver setado em "1", o valor de RR (Release Rate) decairá gradativamente quando o bit "key" respectivo for desligado; caso contrário, o som será cortado abruptamente.

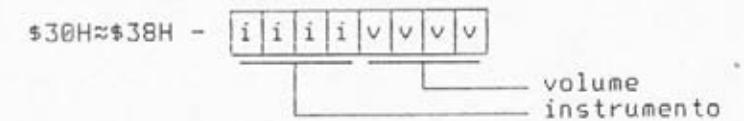
• CONTROLE DA BATERIA ($0EH)

Este registrador controla o modo de bateria do OPLL. Para ativá-lo, basta setar o bit 5 em "1". Os bits 0 a 4 habilitam ou desabilitam cada uma das 5 peças de bateria disponíveis. Quando estiver no modo bateria, somente as seis primeiras vozes do OPLL estarão disponíveis para o programador.



• INSTRUMENTOS/VOLUME ($30H≈$38H)

Estes registros selecionam o instrumento e o volume para cada uma das nove vozes disponíveis. Assim, $30H é usado para a primeira voz, $31H para a segunda, etc.



Os quatro bits menos significativos determinam o volume. O valor 0000 representa o volume máximo e 1111 o mínimo.

Os quatro bits mais significativos selecionam o instrumento, sendo que o valor 0000 seleciona o instrumento criado pelo usuário. Os 15 instrumentos disponíveis são os seguintes:

0001 - Violino	0110 - Oboé	1011 - Cravo
0010 - Violão	0111 - Trompete	1100 - Vibrafone
0011 - Piano	1000 - órgão	1101 - Baixo elétrico
0100 - Flauta	1001 - Piston	1110 - Baixo acústico
0101 - Clarinete	1010 - Sintetizador	1111 - Guitarra

No modo bateria, os registradores $36H, $37H e $38H determinam apenas o volume de cada uma das peças de bateria disponíveis, mas os registros $30H a $35H continuam com a mesma função, tendo, portanto, seis instrumentos simultâneos mais cinco peças de bateria.

	b7	b6	b5	b4	b3	b2	b1	b0	
$36H -	•	•	•	•	bass drum				Registradores
$37H -	high hat				snare drum				de volume
$38H -	tom-tom				top cymbal				da bateria

3.2 - O FM-BIOS

Normalmente, o OPLL vem acompanhado de uma ROM que permite acessar, através do BASIC, todos os registros do mesmo. Esse BASIC ampliado denomina-se MSX-MUSIC. Adicionalmente, há a definição de mais 48 instrumentos nessa ROM, que se chama FM-BIOS. Assim, tem-se acesso a até 63 instrumentos. É fácil perceber que apenas os 15 instrumentos internos do OPLL podem ser misturados uns com os outros livremente em qualquer uma das nove vozes. Já os instrumentos selecionados pelo FM-BIOS não podem ser misturados uns com os outros, já que o OPLL aceita a definição de apenas um instrumento externo. O FM-BIOS reserva o número 63 (silence) para a definição de instrumento pelo usuário, isso através do MSX-MUSIC. Para acesso direto, o instrumento definido pelo programador é o de número 0.

Veja na tabela abaixo os valores de definição para todos os instrumentos do FM-BIOS.

00 - Piano 1	dados do OPLL (3)
01 - Piano 2	30 10 0F 04 D9 B2 10 F4
02 - Violin	dados do OPLL (1)
03 - Flute 1	dados do OPLL (4)
04 - Clarinet	dados do OPLL (5)
05 - Oboe	dados do OPLL (6)
06 - Trumpet	dados do OPLL (7)
07 - Pipe organ	34 30 37 06 50 30 76 06
08 - Xylophone	17 52 18 05 88 D9 66 24
09 - Organ	dados do OPLL (8)
10 - Guitar	dados do OPLL (2)
11 - Santool 1	19 53 0C 06 C7 F5 11 03
12 - Electric guitar	dados do OPLL (15)
13 - Clavicode 1	03 09 11 06 D2 B4 F5 F6
14 - Harpsicode 1	dados do OPLL (11)
15 - Harpsicode 2	01 01 11 06 C0 B4 01 F7
16 - Vibraphone	dados do OPLL (12)
17 - Koto 1	13 11 0C 06 FC D2 33 84
18 - Taiko	01 10 0E 07 CA E6 44 24
19 - Engine 1	E0 F4 1B B7 11 F0 04 08
20 - UFO	FF 70 19 07 50 1F 05 01
21 - Synthesizer bell	13 11 11 07 FA F2 21 F5
22 - Chime	A6 42 10 05 FB B9 11 02
23 - Synthesizer bass	dados do OPLL (13)
24 - Synthesizer	dados do OPLL (10)
25 - Synthesizer percussion	01 03 0B 07 BA D9 25 06
26 - Synthesizer rhythm	40 00 00 07 FA D9 37 04
27 - Harm drum	02 03 09 07 CB FF 39 06
28 - Cowbell	18 11 09 05 F8 F5 26 26
29 - Close hi-hat	0B 04 09 07 F0 F5 01 27
30 - Snare drum	40 40 07 07 D0 D6 01 27
31 - Bass drum	00 01 07 06 CB E3 36 25
32 - Piano 3	11 11 08 04 FA B2 20 F5
33 - Wood bass	dados do OPLL (14)
34 - Santool 2	19 53 15 07 E7 95 21 03
35 - Brass	30 70 19 07 42 62 26 24
36 - Flute 2	62 71 25 07 64 43 12 26
37 - Clavicode 2	21 03 0B 05 90 D4 02 F6
38 - Clavicode 3	01 03 0A 05 90 A4 03 F6
39 - Koto 2	43 53 0E 85 B5 E9 85 04
40 - Pipe organ 2	34 30 26 06 50 30 76 06

41 - RhodsPLA	73	33	5A	06	99	F5	14	15
42 - RhodsPRA	73	13	16	05	F9	F5	33	03
43 - Orch L	61	21	15	07	76	54	23	06
44 - Orch R	63	70	1B	07	75	4B	45	15
45 - Synthesizer violin	61	A1	0A	05	76	54	12	07
46 - Synthesizer organ	61	78	0D	05	85	F2	14	03
47 - Synthesizer brass	31	71	15	07	B6	F9	03	26
48 - Tube	dados do OPLL (9)							
49 - Shamisen	03	0C	14	06	A7	FC	13	15
50 - Magical	13	32	81	03	20	85	03	B0
51 - Huwawa	F1	31	17	05	23	40	14	09
52 - Wander flat	F0	74	17	47	5A	43	06	FD
53 - Hardrock	20	71	0D	06	C1	D5	56	06
54 - Machine	30	32	06	06	40	40	04	74
55 - Machine V	30	32	03	03	40	40	04	74
56 - Comic	01	08	0D	07	78	F8	7F	FA
57 - SE-Comic	C8	C0	0B	05	76	F7	11	FA
58 - SE-Laser	49	40	0B	07	B4	F9	00	05
59 - SE-Noise	CD	42	0C	06	A2	F0	00	01
60 - SE-Star 1	51	42	13	07	13	10	42	01
61 - SE-Star 2	51	42	13	07	13	10	42	01
62 - Engine 2	30	34	12	06	23	70	26	02
63 - Silence	00	00	00	00	00	00	00	00

A tabela relaciona os 63 instrumentos do MSX-MUSIC; os oito bytes respectivos de cada instrumento devem preencher os oito primeiros registradores do OPLL ($00H a $07H), que são os responsáveis pela reprodução do instrumento criado pelo programador. A tabela traz também os instrumentos internos do OPLL e neste caso ao invés dos bytes traz a expressão "dados do OPLL".

3.3 - O FM ESTÉREO

Embora não previsto oficialmente pela Microsoft, o FM estéreo é praticamente padronizado pelo mercado devido a uma característica do OPLL. O chip responsável, o YM2413, possui duas saídas separadas para os sons. Uma é denominada "Melody output" e por ela o OPLL gera as seis primeiras vozes. A outra é denominada "Rhythm output" e por ela o OPLL gera as três vozes restantes ou as cinco peças de bateria. Convencionou-se então que o "Melody output" seria um dos canais estéreo e o "Rhythm output" mais o som do PSG seria o outro canal estéreo.

Assim, o FM estéreo pode gerar dois canais de seis vozes cada. Mesmo não reconhecido oficialmente, muitos programas, especialmente jogos, fazem uso do FM estéreo, produzindo um belíssimo som de suaves nuances.

3.4 - O ACESSO AO OPLL

O acesso ao OPLL é feito diretamente por duas portas de I/O da CPU, a 7CH e a 7DH. A porta 7CH acessa os registros ou endereços do OPLL, enquanto a porta 7DH escreve os dados nos registradores. Entretanto, o OPLL é lento. Entre um acesso e outro, deve haver uma pausa. A pausa para o clock do OPLL que é de aproximadamente 3,58 MHz (igual ao do Z80) é de 12 ciclos T após a escrita de um endereço e de 84 ciclos T após a escrita de um dado. Recomenda-se o uso de pausas tipo "EX (SP),HL" até que o OPLL esteja pronto para novo acesso.

Primeiramente, deve-se escrever o número do registrador (endereço) pela porta 7CH. Deve-se então dar uma pausa de no mínimo 12 ciclos T. Observe que a instrução "EX (SP),HL" demora 19 ciclos T para ser processada, porém esta instrução, quando usada como pausa, deve sempre ser feita em duplas, para evitar que o conteúdo da pilha seja alterado. Então, usa-se duas instruções "EX (SP),HL" após a escrita do dado (pausa de 38 ciclos T).

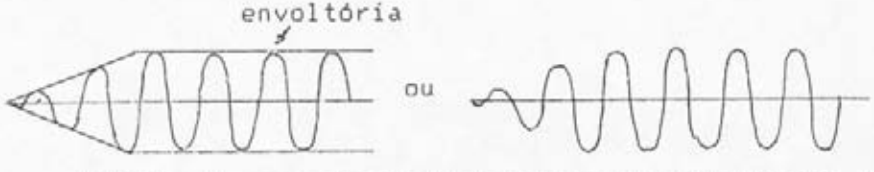
```
LD   A,ENDER       ;nº do registro do OPLL em A
OUT  07CH,A        ;escreve o endereço no OPLL
EX   (SP),HL       ;pausa
EX   (SP),HL       ;pausa
```

Logo em seguida, escreve-se o dado no registrador selecionado através da porta 7DH. A pausa agora deve ser de, no mínimo, 84 ciclos T, ou seja, pelo menos seis instruções EX (SP),HL. Então, o OPLL estará pronto para receber novo endereço.

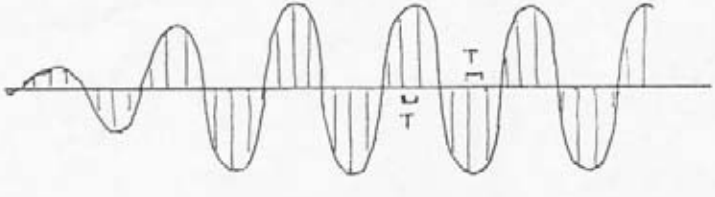
```
LD   A,ENDER       ;nº do registro do OPLL em A
OUT  07CH,A        ;escreve o endereço no OPLL
EX   (SP),HL       ;pausa
EX   (SP),HL       ;pausa
LD   A,DADO        ;dado a ser escrito no OPLL
OUT  07DH,A        ;escreve o dado no OPLL
EX   (SP),HL       ;pausa
EX   (SP),HL       ;pausa
EX   (SP),HL       ;pausa
EX   (SP),HL       ;pausa
EX   (SP),HL       ;pausa
EX   (SP),HL       ;pausa
```

4 - O PCM

PCM significa "Pulse Code Modulation", ou "Modulação por Código de Pulsos". O PCM não é um gerador de sons propriamente dito: funciona dom digitalizador ou "sampler" de sons. Dessa forma, é possível reproduzir sons de qualquer natureza, inclusive a voz humana, de forma praticamente perfeita. O PCM não tem registradores para especificar os sons. Estes são produzidos por amostragem. Para entender como funciona o PCM, vejamos um sinal típico gerado pelo PSG:



O PCM pode reproduzir qualquer som por amostragem. Vejamos no caso da onda acima como o PCM atua:



Perceba que a cada período "T" o PCM faz uma coleta do nível de som. O período "T" é fixo e a freqüência com a qual se processa a amostragem é chamada de "sampling rate". Para reproduzir o som, basta repetir os dados na mesma velocidade em que foram coletados. Outra característica do PCM é a resolução. No caso do MSX, a resolução é de 8 bits, ou seja, cada coleta tem 8 bits de resolução, sendo que o nível pode variar de 0 a 255. Assim, cada amostra ocupa um byte de memória. No MSX, há quatro taxas de amostragem (sampling rate) disponíveis: 3,9375 KHz, 5,25 KHz, 7,875 KHz e 15,75 KHz. Isso quer dizer que, por exemplo, na taxa de 5.25 KHz, há 5250 coletas de 8 bits a cada segundo, o que quer dizer que para cada segundo de reprodução sonora, são necessários cerca de 5 Kbytes de memória. Observe que o gasto de memória para o PCM é grande, tanto que os dados são armazenados na própria RAM do micro.

4.1 - O ACESSO AO PCM

O PCM pode ser acessado tanto pelo BIOS quanto diretamente, mas o acesso direto exige instruções específicas do R800 que não estão presentes no Z80, razão pela qual só será apresentado aqui o acesso pelo BIOS. Existem apenas duas rotinas do BIOS relacionadas ao PCM, que são as seguintes:

PCMPLY (0186H/Main)
  Função: reproduzir o som pelo PCM.
  Entrada: HL - endereço de início para leitura.
           BC - quantidade de bytes a reproduzir.
           A - [m|•|•|•|•|•|f|f]

- f,f: sampling rate:
  00 - 15,75 KHz
  01 - 7,875 KHz
  10 - 5,25 KHz
  11 - 3,9375 KHz
- m: memória para leitura:
  0-Main RAM; 1-VRAM

  Obs.: usar a sampling rate de 15,75 KHz apenas no modo R800 DRAM.
  Saída: Flag CY - 0-parou; 1-parou porque tem erro.
         A - 1-tem erro na frequência.
             2-foi pressionada a tecla STOP.

PCMREC (0189H/Main)
  Função: digitalizar os sons pelo PCM.
  Entrada: igual a PCMPLY, exceto para o registrador A:
           A - [m|t|t|t|t|c|f|f]

- f,f: sampling rate (igual a PCMPLY).
- c: modo de digitalização:
  0-normal; 1-compactada
- t,t,t,t: trigger level (nível de som no qual o PCM grava):
  1111 - sensibilidade mínima
  0000 - sensibilidade máxima
- m: memória para gravação:
  0-Main RAM; 1-VRAM

  Saída: mesma de PCMPLY.

5 - O MSX-AUDIO (OPL)

O MSX-AUDIO foi criado juntamente com o MSX2 em 1985, mas provavelmente por seu preço mais elevado, acabou não se tornando padrão, cedendo lugar mais tarde ao FM OPLL. Apenas alguns países da Europa utilizaram o chip do MSX-AUDIO, o Y8950.

De todos os geradores de som criados para o MSX, o MSX-AUDIO é o mais completo. Como o OPLL, possui 9 vozes de som FM, mas todas são redefiníveis. Também possui um canal ADPCM que funciona de forma semelhante ao PCM, mas é mais elaborado, ocupando muito menos memória que o PCM simples. Além disso, também pode ser conectado diretamente a até 256 Kbytes de memória externa, liberando a Main RAM e a CPU. Pode também ser conectado a um teclado musical diretamente. Estas e outras características fazem do MSX-AUDIO o melhor e mais completo gerador de áudio já criado para o sistema MSX.

O MSX-AUDIO possui 141 registradores de 8 bits numerados de $01H a $1AH, $20H a $35H, $40H a $55H, $60H a $75H, $80H a $95H, $A0H a $A8H, $B0H a $B8H, $BDH e $C0H a $C8H.

Reg.	7	6	5	4	3	2	1	0	
$01H	TESTE								Registr. de teste
$02H $03H	TIMER 1 TIMER 2								Registradores de tempo
$04H	IRQ	T1M	T2M	EOS	BR	•	ST2	ST1	Registr. de flags
$05H $06H	Teclado (in) Teclado (out)								Registradores para teclado externo
$07H $08H	STA CSM	REC SEL	MEM •	REP •	OFF SAM	• DAD	• 64K	RST ROM	Registradores de controle
$09H $0AH $0BH $0CH	Endereço inicial (low) Endereço inicial (high) Endereço final (low) Endereço final (high)								Endereços inicial e final para acesso pela CPU e ADPCM
$0DH $0EH	Sampling rate (low) Sampling rate (high)								Freqüência para o ADPCM
$0FH	Dados para o ADPCM								Registr. de dados
$10H $11H	Fator de interpolação (low) Fator de interpolação (high)								Fator de inperpolação para o ADPCM
$12H	Volume do ADPCM								Volume do ADPCM
$15H $16H $17H	Dados do DAC (high) DAC-low •	• •	• •	• •	• •	• SH2	• SH1	• SH0	Dados digitais para a conversão DA
$18H $19H	• •	• •	• •	• •	Controle I/O Dados I/O				Controle das portas de I/O de 4 bits
$1AH	Dados para o ADPCM								Registr. de dados

$20H $35H	A M	V I B	E G T	K S R	Múltiplo				Registradores para o gerador FM
$40H $55H	K S L				Nivel total				Registradores para o gerador FM
$60H $75H	Attack Rate (AR)				Decay Rate (DR)				Registradores para o gerador FM
$80H $95H	Sustain Level (SL)				Release Rate (RR)				Registradores para o gerador FM
$A0H $A8H	Freqüência das 9 vozes FM								Freqüência LSB 8 bits
$B0H $B8H	• • •	• • •	K E Y	oitava			Freq. MSB 2 bits		Freq. MSB 2 bits Oitava Reg. Key on/off
$BDH	AM	VIB	BAT	BD	SD	TOM	TC	HH	Controle de bateria
$C0H $C8H	• • •	• • •	• • •	• • •	Feedback			CON	Fator de realimentação e conexão
STAT	INT	T1	T2	EOS	BUF	•	•	PCM	Registr. de status

5.1 - REGISTRADOR DE TESTE ($01H)

Este registrador é usado somente para teste do chip do OPL (MSX-AUDIO), o Y8950. Normalmente, seu valor é 0.

5.2 - REGISTRADORES DE TEMPO ($02H,$03H)

Existem dois registradores de tempo: $02H com resolução de 80 µs e $03H com resolução de 320 µs. Eles são contadores de tempo de 8 bits, e podem realizar operações de início, parada e também gerar interrupções para a CPU. O tempo de contagem de cada registrador pode ser calculado pelas seguintes fórmulas:

```
T0 (ms) = (256 - $02) * 0,08
T1 (ms) = (256 - $03) * 0,32
```

5.3 - CONTROLE DE FLAGS ($04H)

O registrador de flags ($04H) é usado para controle de início, parada e interrupções dos registradores $02H e $03H, do ADPCM e da memória externa (de áudio). Cada bit desse registrador habilita ou desabilita uma função, como descrito abaixo:

b0 (ST1): Este bit controla as operações de início e parada de $02H. Quando for 0, $02H estará desativado. Quando for 1, $02H será carregado com o valor respectivo e iniciará a contagem.

b1 (ST2): Este bit controla as operações de $03H da mesma forma que b0.
b2: Não usado.
b3 (MASK BUF.RDY): Este bit controla o ADPCM e a memória de áudio. Quando for 0, a função estará desativada. Quando for 1, os dados de escrita e leitura estarão mascarados durante a transferência de dados entre o processador e o ADPCM ou memória de áudio.
b4 (MASK EDS): Este bit é usado para mascarar o bit b3, quando setado em "1".
b5 (MASK T2): Quando este bit for setado em 1, b1 será setado em 0.
b6 (MASK T1): Este bit é usado para mascarar o bit b0, quando setado em "1".
b7 (IRQ RESET): Cada flag do MSX-AUDIO é setada em "1" quando o respectivo evento ocorre e o sinal IRQ fica no nível 0 (as interrupções ficam desabilitadas). Este bit é usado para reabilitar as interrupções. Quando este bit for "1", todas as flags serão colocadas em "0". Se somente algumas flags devem ser resetadas, sete em "1" o bit MASK correspondente. Após todas as flags serem resetadas, o bit b7 é automaticamente resetado em "0".

5.4 - CONTROLE DE TECLADO, MEMÓRIA E ADPCM ($05H a $08H)

	b7	b6	b5	b4	b3	b2	b1	b0
$05H -	b7	b6	b5	b4	b3	b2	b1	b0
$06H -								

Estes dois registradores são usados para controlar o teclado musical externo, sendo que $05H é configurado como entrada e $06H como saída. Assim, o sinal emitido por cada bit de $06H pode ser lido por cada bit de $05H, formando uma matriz de 8 x 8 para o teclado.

	b7	b6	b5	b4	b3	b2	b1	b0
$07H -								

Este registrador é usado para controlar o início e parada do ADPCM e também para setar o acesso à memória de áudio. Cada bit é uma flag que realiza uma função diferente.

b0 (RESET): Quando este bit é setado em "1" durante a gravação de som pelo ADPCM usando a memória de áudio, estes são recolocados no estado inicial. Neste caso, o bit REPEAT é setado em "0".
b1: Não usado.
b2: Não usado.
b3 (SP-OFF): Quando este bit estiver setado em "1", o terminal de saída de som estará setado em "1". Este bit é usado para proteger o alto-falante durante a gravação pelo ADPCM ou conversão AD.
b4 (REPEAT): Durante a síntese de som pelo ADPCM usando a memória de áudio, este bit é setado em "1" para evitar que dados repetidos sejam lidos na mesma área da memória.
b5 (MEMORY DATA): Este bit é setado em "1" quando a memória de áudio é acessada.

b6 (REC): Este bit é setado em "1" para a gravação de sons pelo ADPCM ou para dados enviados pela CPU para a memória de áudio.

b7 (START): Este bit é setado em "1" para leitura ou gravação pelo ADPCM. O tempo de início difere de acordo com a localização dos dados (Main RAM ou memória de áudio). Se os dados estiverem armazenados na CPU, o ADPCM começará a leitura ou escrita do registrador $0FH. Se estiverem na memória de áudio, o ADPCM começará quando o bit START for setado em "1"; conseqüentemente deve-se carregar todos os registradores necessários antes de setar este bit em "1".

$08H -

b7	b6	b5	b4	b3	b2	b1	b0

b0 (ROM): Este bit é usado para identificar o tipo de memória de áudio: 0-RAM; 1-ROM.

b1 (64K): Este bit é usado para especificar a quantidade de memória de áudio disponível. 0-256K DRAM; 1-64K DRAM. Quando este bit for "1", a linha de endereço A8 é ignorada. Para ROM, este bit é setado em 0.

b2 (DA/AD): Este bit é usado em conjunto com o bit b3 (SAMPLE). Quando este bit for "1", os dados especificados em $15H a $17H são enviados. Quando for "0", estarão habilitados a conversão AD (b3=1) ou a saída de música (b3=0).

b3 (SAMPLE): Este bit é usado para habilitar o timer para conversão AD/DA. Quando for "1", a conversão AD se inicia.

b4: Não usado.

b5: Não usado.

b6 (NOTE SEL): Este bit é usado para especificar os pontos de separação de uma oitava para o teclado musical externo. Quando for "0", o ponto de separação é especificado pelos dois bits MSB de freqüência. Quando for "1", o ponto de separação é especificado pelo bit MSB da freqüência (registradores $B0H a $B8H).

b6=1

0	1	2	3	4	5	6	7	Oitava
1	1	1	1	1	1	1	1	Freq 1º MSB
0:1	0:1	0:1	0:1	0:1	0:1	0:1	0:1	Freq 2º MSB
0:1	2:3	4:5	6:7	8:9	A:B	C:D	E:F	Teclado (hex)

b6=0

0	1	2	3	4	5	6	7	Oitava
0:1	0:1	0:1	0:1	0:1	0:1	0:1	0:1	Freq MSB
0:1	2:3	4:5	6:7	8:9	A:B	C:D	E:F	Teclado (hex)

b7 (CSM): Este bit é setado em "1" para o modo de modulação senoidal composta. Para isso, todas as vozes devem estar setadas em Key-off.

5.5 - ENDEREÇOS DE ACESSO ($09H a $0CH)

Estes registros especificam o endereço inicial e final da memória de áudio a ser acessada. O valor destes registros difere um pouco de acordo com o tipo de memória (ROM ou RAM).

Os registradores $09H e $0AH especificam o endereço inicial, como ilustrado na página seguinte.

64K DRAM

Banco			Endereço CAS								Endereço RAS								
B2	B1	B0	A8	A7	A6	A5	A4	A3	A2	A1	A0	A8	A7	A6	A5	A4	A3	A2	A1	A0
- $0AH -												- $09H -								
b7	b6	b5	0	b3	b2	b1	b0	b7	b6	b5	b4	0	b2	b1	b0	0	0	0	0	0

256K DRAM

Banco			Endereço CAS								Endereço RAS								
B2	B1	B0	A8	A7	A6	A5	A4	A3	A2	A1	A0	A8	A7	A6	A5	A4	A3	A2	A1	A0
- $0AH -												- $09H -								
b7	b6	b5	b4	b3	b2	b1	b0	b7	b6	b5	b4	b3	b2	b1	b0	0	0	0	0	0

64K ROM

Banco			Endereço CAS								Endereço RAS								
B2	B1	B0	A8	A7	A6	A5	A4	A3	A2	A1	A0	A8	A7	A6	A5	A4	A3	A2	A1	A0
- $0AH -												- $09H -								
0	0	0	b4	b3	b2	b1	b0	b7	b6	b5	b4	b3	b2	b1	b0	0	0	0	0	0

Já os registradores $0BH e $0CH especificam o endereço final para acesso à memória de áudio.

64K DRAM

Banco			Endereço CAS								Endereço RAS								
B2	B1	B0	A8	A7	A6	A5	A4	A3	A2	A1	A0	A8	A7	A6	A5	A4	A3	A2	A1	A0
- $0CH -												- $0BH -								
b7	b6	b5	0	b3	b2	b1	b0	b7	b6	b5	b4	0	b2	b1	b0	1	1	1	1	1

256K DRAM

Banco			Endereço CAS								Endereço RAS								
B2	B1	B0	A8	A7	A6	A5	A4	A3	A2	A1	A0	A8	A7	A6	A5	A4	A3	A2	A1	A0
- $0CH -												-$0BH-								
b7	b6	b5	b4	b3	b2	b1	b0	b7	b6	b5	b4	b3	b2	b1	b0	1	1	1	1	1

64K ROM

Banco			Endereço CAS								Endereço RAS								
B2	B1	B0	A8	A7	A6	A5	A4	A3	A2	A1	A0	A8	A7	A6	A5	A4	A3	A2	A1	A0
- $0CH -												- $0BH -								
0	0	0	b4	b3	b2	b1	b0	b7	b6	b5	b4	b3	b2	b1	b0	1	1	1	1	1

5.6 - ACESSO AO ADPCM ($0DH A $01AH)

Os registradores $0DH e $0EH especificam a taxa de amostragem (sampling rate) para a conversão AD e DA do ADPCM. A taxa máxima é de 16 KHz e a mínima de 1,8 KHz. Seu valor pode ser obtido usando a fórmula abaixo:

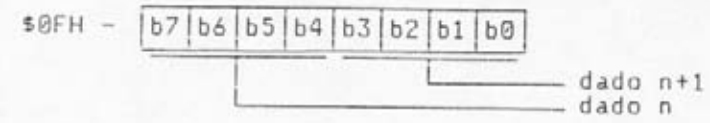
Taxa (KHz) = 3580 / NPRE

	b7	b6	b5	b4	b3	b2	b1	b0	
$0DH -									low
$0EH -	0	0	0	0	0	b2	b1	b0	high

16 KHz => $0DH = E0H e $0EH = 00H
1.8 KHz => $0DH = C4H e $0EH = 07H

O registrador $0FH é usado para intercâmbio dos dados do ADPCM com a CPU. Ele também é usado como buffer quando a memória de áudio é acessada pela CPU.

Quando usado para dados do ADPCM, este registrador contém dois dados, visto que os dados do ADPCM são compactados em quatro bits cada. Os quatro bits mais altos contêm o dado **n** e os quatro bits mais baixos contêm o dado **n+1**.



Os registradores $10H e $11H especificam o fator para a interpolação linear com a freqüência de 50 KHz do gerador FM durante os intervalos da síntese de sons pelo ADPCM. Este fator também é usado como taxa de amostragem para a síntese e neste caso os registradores $0DH e $0EH são ignorados. Para calcular o valor dos registradores, pode-se usar a seguinte fórmula simplificada:

DELTA-N = 1310.72 * taxa amostragem (KHz)

	b7	b6	b5	b4	b3	b2	b1	b0	
$10H -	b7	b6	b5	b4	b3	b2	b1	b0	DELTA-N (low)
$11H -	b7	b6	b5	b4	b3	b2	b1	b0	DELTA-N (high)

16 KHz => $10H = ECH e $11H = 51H
1.8 KHz => $10H = 37H e $11H = 09H

O registrador $12H é o controle de volume se saída do ADPCM em 256 níveis. O volume mínimo é obtido quando o registrador for 0 e o máximo quando for 255. O valor deste registrador não afeta a saída do gerador FM.

	b7	b6	b5	b4	b3	b2	b1	b0	
$12H -	b7	b6	b5	b4	b3	b2	b1	b0	volume do ADPCM

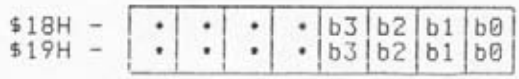
Os registradores $15H a $17H são usados para especificar os dados digitais para a conversão DA. Nos três registradores, apenas 13 bits são usados para calcular o valor da saída analógica. Para a conversão DA, os valores iniciais devem ser escritos nos registradores $15H~$17H antes de setar o bit 2 do registrador $08H. As fórmulas que descrevem os valores dos registradores são as seguintes:

SAiDA = (F9 + F8*2^(-1) + ... + F0*2^(-9) + 2^(-10) - 1) * 2^(-E)

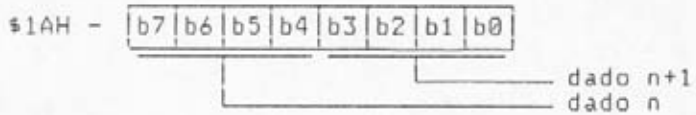
E = S2 * $\overline{4}$ + S1 * $\overline{2}$ + S0 (S0+S1+S2 > 0)

	b7	b6	b5	b4	b3	b2	b1	b0
$15H -	f9	f8	f7	f6	f5	f4	f3	f2
$16H -	f1	f0	•	•	•	•	•	•
$17H -	•	•	•	•	•	S2	S1	S0

Os registradores $18H e $19H são registradores de 4 bits usados para controlar as portas de I/O de uso geral do MSX-AUDIO. $18H especifica se é entrada ou saída. Deve ser setado em "1" para saída e em "0" para entrada. $19H é usado para transferir os dados pela porta de I/O de 4 bits.



O registro $1AH é usado para armazenar os dados processados pela conversão AD. Observe que para o ADPCM são armazenados dois dados de 4 bits, que foram compactados durante a digitalização dos mesmos.

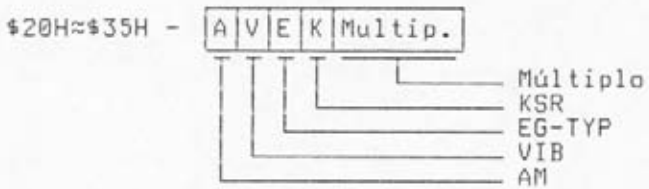


5.7 - ACESSO AO GERADOR FM

O OPL pode gerar até 9 vozes de som FM ou 6 vozes mais 5 peças de bateria, como o OPLL, mas todas as 9 vozes devem ser definidas pelo programador. Os registradores usados para geração de sons FM estão descritos abaixo.

• AM/VIB/EG-TYP/KSR/MúLTIPLO ($20H a $35H)

Estes registradores são usados para especificar a forma da envoltória e os fatores de multiplicação para as ondas carrier e modulada.



MúLTIPLO (b0≈b3)

Estes bits especificam os fatores de multiplicação usados para converter as ondas modulada e carrier. Os fatores de multiplicação podem ser vistos na tabela abaixo.

```
Valor do registro: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
Fator de multipl.: ½ 1 2 3 4 5 6 7 8 9 10 10 12 12 15 15
```

KSR (b4)

Este bit especifica a "Key Scale" para as razões de "Attack" e "Decay". A "Key Scale" é usada para fazer com que o som gerado pelo OPL se aproxime dos instrumentos musicais reais.

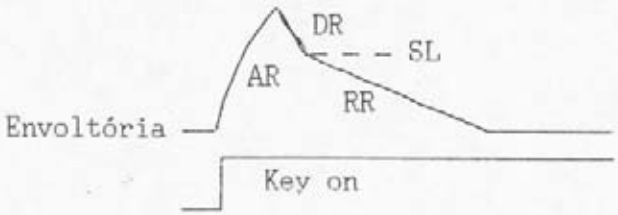
Key Scale Rate (KSR):

```
b4=0
Key scale: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
Fatores:   0 0 0 0 1 1 1 1 2 2  2  2  3  3  3  3

b4=1
Key Scale: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
Fatores:   0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
```

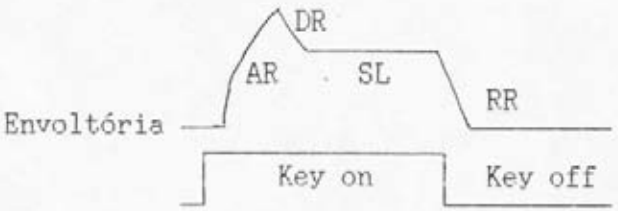
EG-TYP (b5)

Este bit seleciona entre tom constante ou tom percussivo. Se for igual a 0, o tom será percussivo e se for igual a 1 será constante. Veja as ilustrações abaixo.

Tom percussivo (b5=0)



Tom constante (b5=1)



AR=Attack Rate; DR=Decay Rate; SL=Sustain Level; RR=Release Rate

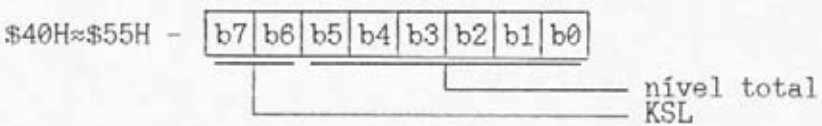
VIB (b6)

Este bit liga ou desliga o vibrato. Se for "1", o vibrato estará ativo e se for "0" estará desligado. A freqüência do vibrato é de 6.4 Hz. O grau do vibrato é setado pelo bit VIB--DEPTH do registrador $BDH.

AM (b7)

Este bit liga ou desliga a modulação de amplitude ou "trêmolo". Se for 1, a modulação de amplitude estará ativa e se for "0" estará desligada. A freqüência do trêmolo é de 3,7 Hz. O grau do trêmolo é setado pelo bit AM-DEPTH de $BDH.

• KSL / NíVEL TOTAL ($40H a $55H)



NíVEL TOTAL (b0≈b5)

Os seis bits do nível total são usados para controlar o grau de modulação da envoltória (nível da envoltória).

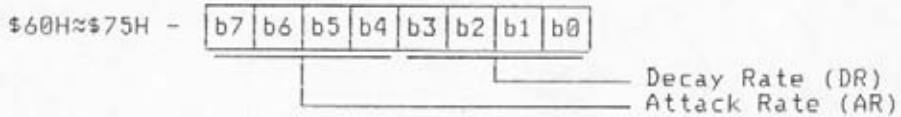
A resolução máxima de "decay" é de 0.75 dB (000001) e o nível de saída pode ser reduzido até 47.25 dB (111111).

KSL (b6≈b7)

Estes bits controlam o nível de saída pela "Key Scale". No modo "key scale", o nível de atenuação progressiva do som pela freqüência pode variar de 0 dB/oitava até 6 dB/oitava.

b6	b7	Atenuação
0	0	0 dB/oitava
1	0	1.5 dB/oitava
0	1	3 dB/oitava
1	1	6 dB/oitava

• RELAÇÃO ATTACK/DECAY ($60H a $75H)

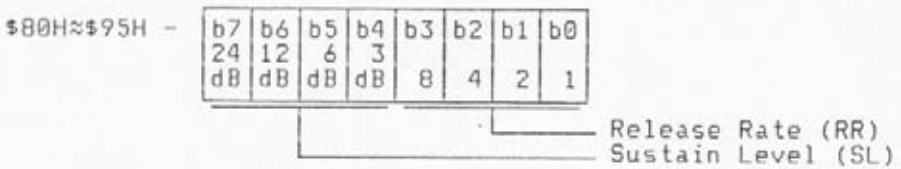


As razões de "attack" e "decay" são definidas pelos registradores $60H a $75H, sendo que os bits b0≈b3 definem o nível de "decay" e os bits b4≈b7 o de "attack". Quanto maior o valor, menor o tempo de "attack" e "decay".

• SUSTAIN LEVEL / RELEASE RATE ($80H a $95H)

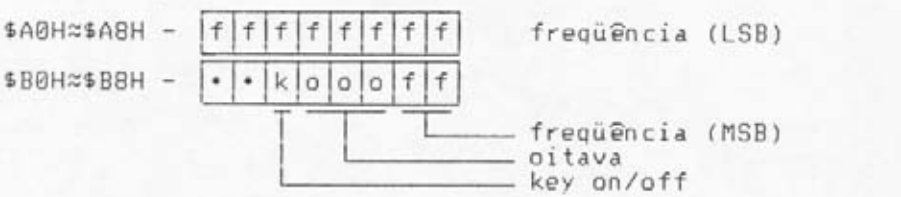
"Sustain Level" especifica o nível em que o som ficará depois do "Decay Rate". Para o tom percussivo, especifica o ponto de troca do modo decay para o modo release. Quanto maior o valor deste registrador, menor será o nível de "sustain".

A "Release Rate", para o tom constante, especifica a razão de "decay" após o "key off". Para o tom percussivo, especifica a razão de "decay" após o "sustain level". Quanto maior o valor do registrador, menor será a duração da "release rate".



• OITAVA/FREQÜÊNCIA ($A0H a $B8H)

São nove grupos de dois registradores cada, sendo que os registradores $A0H-$B0H controlam a primeira voz, $A1H-$B1H controlam a segunda voz e assim por diante.



FREQÜÊNCIA ($AxH e bits 0-1 de $BxH)

Estes 10 bits definem uma escala de freqüências para

cada oitava. Na tabela abaixo, estão especificados os valores dos registradores para a quarta oitava, de um total de 8 oitavas, com a nota LÁ central de 440 Hz.

Cifrado		Freqüência	Registros	$BxH-d1.d0	$AxH
Dó	C#	277.2 Hz	363	0 1	01101011
Ré	D	293.7 Hz	385	0 1	10000001
	D#	311.1 Hz	408	0 1	10011000
Mi	E	329.6 Hz	432	0 1	10110000
Fá	F	349.2 Hz	458	0 1	11001010
	F#	370.0 Hz	485	0 1	11100101
Sol	G	392.0 Hz	514	1 0	00000010
	G#	415.3 Hz	544	1 0	00100000
Lá	A	440.0 Hz	577	1 0	01000001
	A#	466.2 Hz	611	1 0	01100011
Si	B	493.9 Hz	647	1 0	10000111
Dó	C	523,3 Hz	686	1 0	10101110

Tanto os valores das freqüências quanto dos registradores guardam entre si uma relação geométrica igual à 12ª raiz de 2, que vale 1,0594630943592, podendo-se usar este número para alterar o valor dos registradores, a fim de aumentar ou diminuir a freqüência gerada.

OITAVA ($BxH, b2≈b4)

Estes três bits definem a oitava. Podem ser definidas até 8 oitavas, de 000 a 111, sendo que a quarta oitava é a de número 011.

KEY ($BxH, b5)

Este bit deve ser setado em "1" para que o som de cada uma das nove vozes seja habilitado. Quando este bit for setado em "0", provocará o "key off".

• **REALIMENTAÇÃO/CONEXÃO ($C0H a $C8H)**

$C0H≈$C8H -

CONEXÃO ($CxH, b0)

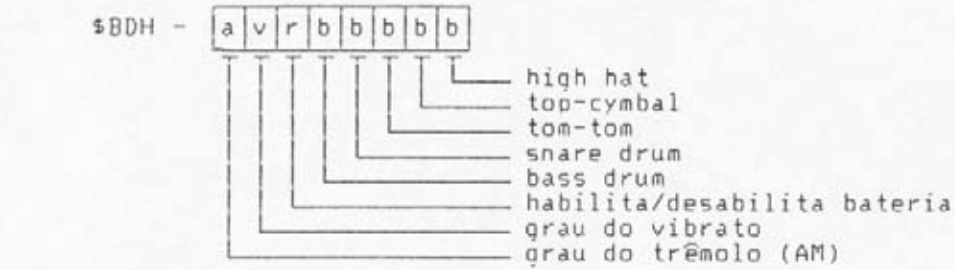
Este bit é usado para especificar o tipo de conexão entre os dois osciladores do OPL (modulador e carrier). Se este bit for "0", os osciladores estarão no modo FM. Se for "1", os osciladores estarão no modo de modulação senoidal composta.

REALIMENTAÇÃO ($CxH, b1≈b3)

Estes bits especificam o índice de realimentação (porção do sinal de saída que é reinjetado na entrada) para a onda modulada. Quanto maior o valor do registro, maior o fator de realimentação (feedback).

Valor do registr.:	0	1	2	3	4	5	6	7
Grau de modulação:	0	π/16	π/8	π/4	π/2	π	2 π	4 π

• BATERIA / AM.VIB-DEPTH ($BDH)



MODO DE BATERIA (b0≈b5)

Para ativar o modo bateria do OPL, basta setar em "1" o bit 5 de $BDH. Os bits b0 a b4 ativam (1) ou desativam (0) cada uma das cinco peças de bateria disponíveis. No modo bateria, apenas as seis primeiras vozes do OPL ficam disponíveis para o programador.

VIB-DEPTH (b6)

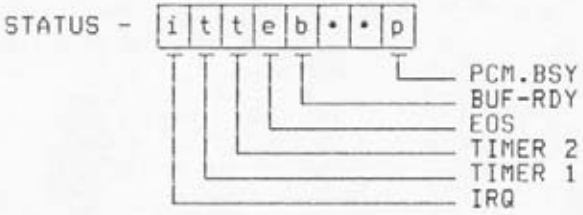
Este bit é usado para selecionar entre dois graus de vibrato: 14% (b6=1) ou 7% (b6=0).

AM-DEPTH (b7)

Este bit é usado para selecionar entre dois graus de modulação de amplitude (trêmolo): 4,8 dB (b7=1) ou 1 dB (b7=0).

5.8 - O REGISTRADOR DE STATUS

O MSX-AUDIO possui um registrador de status com flags para controlar dois timers e a memória de áudio, usadas durante a sintese ou análise de sons pelo ADPCM.



b0 (PCM.BSY): Durante a análise ou sintese de som pelo ADPCM, este bit será setado em "1" se o bit b7 de $07H for "1". Não será gerado sinal de interrupção.
b1: Não usado.
b2: Não usado.
b3 (BUF.RDY): Este bit será setado em "1" nos seguintes casos:
- Fim da análise de som pelo ADPCM ($07H, b5=0)
- Fim da síntese de som pelo ADPCM ($07H, b5=0)
- Fim da escrita na memória de áudio
- Fim da leitura da memória de áudio

b4 (EOS): Este bit é setado em "1" quando a análise ou síntese de som pelo ADPCM for completada ou quando houver lapso de tempo durante a conversão AD/DA.

b5 (TIMER 2): Este bit é setado em "1" após o lapso de tempo gerado pelo timer 2.
b6 (TIMER 1): Igual a b5, mas setado pelo timer 1.
b7 (IRQ): Este bit será setado em "1" quando um ou mais dos bits b3 a b6 for ou forem "1".

5.9 - SEQUÊNCIA PARA ACESSO À MEMÓRIA DE ÁUDIO E ADPCM

• ANÁLISE DE SOM (OPL -> CPU)

Reg.	Dados	R/W	Comentários
			•Inicialização
$04H	00H	W	Todas as flags são habilitadas
$04H	80H	W	Todas as flags são resetadas
$07H	C8H	W	ADPCM é habilitado e saída de som é deslig.
$08H	00H	W	
$0DH	C2H	W	"Sampling Rate" = 8 KHz (NPRE = 450)
$0EH	01H	W	
			•Inicia a análise
$0FH	-	R	Inicia com a leitura do "dummy"
			•Análise
$0FH	-	R	Quando BUF.RDY for "1", $0FH é lido, o dado
$04H	(80H)	(W)	é armazenado e a flag resetada. Quando BUF.RDY for 0, esperar.
			•Fim da análise
$07H	48H	W	Análise pelo ADPCM foi completada
$07H	00H	W	Registrador $07H é resetado

• SÍNTESE DE SOM (CPU -> OPL)

Reg.	Dados	R/W	Comentários
			•Inicialização
$04H	00H	W	Todas as flags são habilitadas
$04H	80H	W	Todas as flags são resetadas
$07H	80H	W	Síntese pelo ADPCM é habilitada
$08H	00H	W	
$10H	F6H	W	"Sampling Rate" = 8 KHz (DELTA-N = 10486)
$11H	28H	W	
$12H	xxH	W	Especificar o volume de saída
			•Início da síntese
$0FH	xxH	W	Escreve dado para o ADPCM em $0FH
			•Síntese
$0FH	xxH	W	Quando BUF.RDY for "1", o dado de síntese é
$04H	(80H)	(W)	escrito em $0FH e a flag resetada. Quando BUF.RDY for 0, esperar.
			•Final da síntese
$07H	00H	W	Síntese pelo ADPCM foi completada

• ANÁLISE DE SOM (OPL -> MEMÓRIA DE ÁUDIO)

Reg.	Dados	R/W	Comentários
			•Inicialização
$04H	08H	W	Somente a flag BUF.RDY é mascarada
$04H	80H	W	Todas as flags são resetadas
$07H	68H	W	Síntese pelo ADPCM é habilitada
$08H	02H/00H	W	Especificar o tipo de RAM

$09H	xxH	W	Endereço inicial da memória de áudio
$0AH	xxH	W	
$0BH	xxH	W	Endereço final da memória de áudio
$0CH	xxH	W	
$0DH	E1H	W	"Sampling Rate" = 16 KHz (NPRE = 225)
$0EH	00H	W	
			•Início da análise
$07H	E8H	W	Iniciar quando b7 de $07H for "1"
			•Análise
			A flag EOS está setada em "1" e espera até o final da análise
			•Fim da análise
$07H	68H	W	Análise pelo ADPCM foi completada
$07H	00H	W	Registrador $07H é resetado

• SÍNTESE DE SOM (MEMÓRIA DE AUDIO -> OPL)

Reg.	Dados	R/W	Comentários
			•Inicialização
$04H	08H	W	Somente a flag BUF.RDY é mascarada
$04H	80H	W	Todas as flags são resetadas
$07H	20H/30H	W	Síntese pelo ADPCM é habilitada
$08H	00H≈02H	W	Especificar o tipo de memória
$09H	xxH	W	Endereço inicial da memória de áudio
$0AH	xxH	W	
$0BH	xxH	W	Endereço final da memória de áudio
$0CH	xxH	W	
$10H	ECH	W	"Sampling Rate" = 16 KHz (DELTA-N = 20972)
$11H	51H	W	
$12H	xxH	W	Especificar o volume de saída
			•Início da síntese
$07H	A0H/B0H	W	Iniciar quando b7 de $07H for "1"
			•Síntese
			A flag EOS está setada em "1" e espera até o final da síntese
$07H	(A0H)	(W)	(Modo de repetição é estabelecido)
$07H	(A1H)	(W)	(Força interrupção da síntese)
			•Fim da síntese
$07H	20H	W	Síntese pelo ADPCM foi completada
$07H	00H	W	Registrador $07H é resetado

• ESCRITA NA RAM DE AUDIO (CPU -> MEMÓRIA DE ÁUDIO)

Reg.	Dados	R/W	Comentários
			•Inicialização
$04H	00H	W	Todas as flags são habilitadas
$04H	80H	W	Todas as flags são resetadas
$07H	60H	W	Habilitado modo de escrita na memória
$08H	00H/02H	W	Especifica o tipo de RAM
$09H	xxH	W	Especifica o endereço inicial
$0AH	xxH	W	
$0BH	xxH	W	Especifica o endereço final
$0CH	xxH	W	
			•Escrita na memória
$0FH	xxH	W	Dado a ser escrito
$04H	(80H)	(W)	(Quando BUF.RDY for "1", o dado é escrito; quando for "0", esperar. Quando o fim da memória chegar, a flag EOS será "1")

			•Reset
$07H	00H	W	O registrador $07H é resetado

• LEITURA DA RAM/ROM DE ÁUDIO (MEMÓRIA DE ÁUDIO -> CPU)

Reg.	Dados	R/W	Comentários
			•Inicialização
$04H	00H	W	Todas as flags são habilitadas
$04H	80H	W	Todas as flags são resetadas
$07H	20H	W	Modo de leitura é estabelecido
$08H	00H≈02H	W	Especifica o tipo de memória
$09H	xxH	W	Especifica o endereço inicial
$0AH	xxH	W	
$0BH	xxH	W	Especifica o endereço final
$0CH	xxH	W	
			•Leitura da memória
$0FH	-	R	Iniciar após ler o "dummy" duas vezes
$0FH	-	R	(Necessário checar a flag)
$0FH	xxH	R	O dado é lido
$04H	80H	W	Quando BUF.RDY for "1", o dado é lido; quando for "0", esperar. Ao terminarem os dados, a flag EOS será igual a "1"
			•Reset
$07H	00H	W	O registrador $07H é resetado

5.10 - ACESSO AO MSX-AUDIO

O acesso ao MSX-AUDIO é feito diretamente através de duas portas de I/O da CPU, a C0H e a C1H. A porta C0H seleciona os registradores ou endereços do OPL e a C1H escreve ou lê os dados nos registradores. Entretanto, tal como o OPLL, o MSX-AUDIO é lento. Deve haver uma pausa entre um acesso e outro. Esta pausa deve ser de 12 ciclos T após a escrita de um endereço ou acesso aos registradores $01H a $1AH e de 84 ciclos T após o acesso aos registradores restantes ($20H a $C8H). É recomendável o uso de pausas do tipo "EX (SP),HL".

Inicialmente, deve-se escrever o número do registrador (endereço) através da porta C0H. Deve-se então dar uma pausa de no mínimo 12 ciclos T. A instrução "EX (SP),HL" demora 19 ciclos T para ser processada, mas quando usada em pausas deve sempre estar em pares, para evitar que o conteúdo da pilha seja alterado. Então, usa-se duas instruções "EX (SP),HL" após a escrita do endereço (pausa de 38 ciclos T).

```
LD   A,ENDER      ;nº do registrador do OPL em A
OUT  (0C0H),A     ;escreve o endereco no OPL
EX   (SP),HL      ;pausa
EX   (SP),HL      ;pausa
```

Logo em seguida, escreve-se ou lê-se o dado no registrador selecionado através da porta C1H. A pausa agora deve ser de no mínimo 12 ciclos T para os registradores $01H a $1AH e de 84 ciclos T para os registradores $20H a $C8H.

Veja na página seguinte um pequeno programa em Assembly ilustrando o primeiro caso.

```
LD   A,ENDER      :ENDER = 01H a 1AH
OUT  (0C0H),A     :escreve o valor 01H=1AH no OPL
EX   (SP),HL      :pausa
EX   (SP),HL      :pausa
LD   A,DADO       :dado a ser escrito no OPL
OUT  (0C1H),A     :escreve o dado no OPL
EX   (SP),HL      :pausa
EX   (SP),HL      :pausa
```

Veja abaixo uma ilustração para o caso dos registradores $20H a $C8H. Estes registradores são bem mais lentos que os anteriores. portanto agora é necessária uma pausa de no minimo 84 ciclos T, ou seis instruções "EX (SP),HL" para a pausa (114 ciclos T).

```
LD   A,ENDER      :ENDER = 20H a C8H
OUT  (0C0H),A     :escreve o valor 20H=C8H no OPL
EX   (SP),HL      :pausa
EX   (SP),HL      :pausa
LD   A,DADO       :dado a ser escrito no OPL
OUT  (0C1H),A     :escreve o dado no OPL
EX   (SP),HL      :pausa
EX   (SP),HL      :pausa
EX   (SP),HL      :pausa
EX   (SP),HL      :pausa
EX   (SP),HL      :pausa
EX   (SP),HL      :pausa
```

Os registradores $01H a $1AH também podem ser lidos. Neste caso. deve-se proceder como ilustrado abaixo:

```
LD   A,ENDER      :ENDER = 01H a 1AH
OUT  (0C0H),A     :seleciona o registrador 01H=1AH do OPL
EX   (SP),HL      :pausa
EX   (SP),HL      :pausa
IN   A,(0C1H)     :lê o valor do registrador do OPL
EX   (SP),HL      :pausa
EX   (SP),HL      :pausa
```

Capítulo 6

O SISTEMA DE DISCO

Grande capacidade de armazenamento externo de massa aliada a alta velocidade de acesso e grande confiabilidade são requisitos necessários para um grande número de aplicações. O periférico que preenche estes requisitos é o disk-drive. O disk-drive é um periférico que é acionado pelas rotinas do BDOS (Basic Disk Operating System). No caso dos micros MSX, o acesso direto ao drive não é recomendado, uma vez que cada fabricante tem liberdade de escolher qualquer tipo de controlador para o sistema de disco. Todos os acessos ao drive devem ser feitos através do BDOS ou do BIOS, pelas rotinas PHYDIO e FORMAT.

O tipo de formatação adotado pelo DOS do MSX é o mesmo do MS-DOS da linha PC. Assim, o MSX-DOS pode ler arquivos do MS-DOS e vice-versa, mas a compatibilidade limita-se apenas a isto, visto que, no restante, o sistema MSX é completamente diferente do sistema do PC.

A unidade de disco padrão para o MSX, lançada em 1984, é a de 3½". Entretanto, no Brasil, devido à conjuntura econômica da época (1987), acabou não se respeitando essa determinação, e as unidades de disco são, em sua maioria, de 5¼".

Há dois sistemas de disco para o MSX: o DOS1 e o DOS2. O MSXDOS1 necessita de no mínimo 64 Kbytes de RAM e aceita até seis drives simultâneos, designados por A: a E:, mas é muito simples e não suporta disco rígido. O MSXDOS2 necessita de no mínimo 256 Kbytes de memória mapeada e aceita até 8 drives simultâneos, de A: até H:, sendo que o drive H: é configurado como RAMDISK. Este sistema manipula subdiretórios e aceita disco rígido, mas requer que o micro seja um MSX2 ou superior.

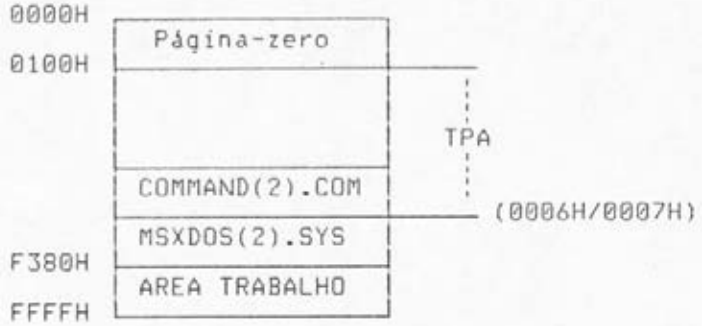
O sistema padrão é o de 3½", mas como no Brasil se usa principalmente o de 5¼", na tabela abaixo estes estão representados. Originalmente, havia a opção de 8 setores por trilha, mas o que acabou se tornando padrão no mercado foram os disquetes de 9 setores por trilha. Os disquetes que podem ser gravados só de um lado recebem a denominação de 1DD e os que podem ser gravados dos dois lados denominam-se 2DD.

	1DD3½	2DD3½	1DD5¼	2DD5¼	HD
ID midia (disco)	F8H	F9H	FCH	FDH	F0H
Número de lados	1	2	1	2	-
Trilhas por lado	80	80	40	40	-
Setores por trilha	9	9	9	9	-
Bytes por setor	512	512	512	512	512
Setores por cluster	2	2	1	2	1
Tamanho da FAT (em setores)	2	3	2	2	-
Número de FATs	2	2	2	2	-
Número máximo de arquivos	112	112	64	112	-

Atualmente no Brasil existem drives 5¼ HD, que gravam 80 trilhas no mesmo formato dos disquetes 3½".

1 - CARACTERÍSTICAS DO SISTEMA DE DISCO MSX

O MSXDOS (1 ou 2) consiste nos seguintes módulos: interface de disco com BDOS em ROM e dos arquivos MSXDOS.SYS e COMMAND.COM (para o MSXDOS2. são MSXDOS2.SYS e COMMAND2.COM). O sistema de disco do MSX difere de outros sistemas pelo fato de que o DOS propriamente dito não se encontra no disco de sistema, mas sim na ROM da interface de disco. Os arquivos MSXDOS.SYS ou MSXDOS2.SYS servem simplesmente como uma espécie de boot para setar os parâmetros necessários ao funcionamento do COMMAND.COM ou COMMAND2.COM, que são os responsáveis pela execução dos comandos do MSXDOS. A ROM da interface de disco inclui as rotinas para acionamento do disk-drive, o DOS Kernel e o interpretador DISK-BASIC, e se situa nos endereços 4000H a 7FFFH (página 1) para o MSXDOS1 e MSXDOS2, embora este último possua 4 páginas (64 K) que são intercambiadas unicamente na página física 1.



A área compreendida entre 0000H e 00FFH é a página-zero (system scratch area) e é de extema importância para o MSXDOS e para os programas aplicativos. Esta área será descrita em detalhes mais adiante. A área que começa em 0100H e termina no endereço indicado pelos bytes 0006H/0007H da página zero chama-se TPA (Transient Program Area), ou Área para Programas Transitórios. É nesta área que são carregados os programas que funcionam sob o DOS. O MSXDOS.SYS se inicia no primeiro endereço logo após esta área, mas o COMMAND.COM é colocado na parte superior da TPA.

O COMMAND.COM

O arquivo COMMAND.COM é o responsável pela execução dos comandos do MSXDOS. Estes comandos podem ser comandos internos, comandos batch ou comandos externos.

Os comandos internos são aqueles que residem no próprio COMMAND.COM. Ao serem chamados, são executados imediatamente.

Para comandos externos, o COMMAND.COM carrega a rotina do disco (que deve ter obrigatoriamente a extensão .COM) e a coloca na TPA a partir do endereço 0100H, sendo que a execução do comando se inicia neste mesmo endereço. Quando a execução do comando externo é terminada através de uma instrução RET, o MSXDOS .SYS examina se o COMMAND.COM foi destruído (no caso de rotinas externas muito grandes) e, se necessário, recarrega o COMMAND.COM e lhe passa o controle.

Os comandos batch são uma série de comandos gravados em

um arquivo (com a extensão .BAT) que o COMMAND.COM executa um a um. Os comandos presentes num arquivo batch podem ser tanto internos quanto externos.

O MSXDOS.SYS

O MSXDOS.SYS é o centro do MSX-DOS. Ele controla o acesso e a comunicação com os periféricos. As funções do MSXDOS .SYS são executadas pelo BDOS (Basic Disk Operating System). As operações do BDOS, portanto, não são realizadas pelo MSXDOS.SYS, mas sim pelo DOS Kernel, residente na ROM da interface de disco. O MSXDOS.SYS é apenas o intermediário entre as operações de I/O requeridas pelo COMMAND.COM ou comandos externos e o DOS Kernel.

O DOS Kernel

O DOS Kernel contém as rotinas básicas de I/O para acesso ao disk-drive. Ele reside na ROM da interface de disco e executa as funções do BDOS do MSXDOS.SYS. Atualmente, qualquer sistema que use o acesso ao disco pode funcionar usando apenas o DOS Kernel. O DISK-BASIC executa suas operações chamando o DOS Kernel diretamente, não necessitando do disco de sistema.

2 - ESTRUTURA DOS ARQUIVOS EM DISCO

As informações sobre a estrutura de dados do disco e como são controladas são importantes para o desenvolvimento de programas que acessam o disco. Esta seção contém todas as informações necessárias para isso.

2.1 - UNIDADES DE DADOS NO DISCO

SETORES

Cada tipo de disquete tem um determinado número de trilhas: assim, os disquetes de 5¼" têm 40 ou 80 trilhas e os de 3½" têm 80 trilhas. No sistema MSX, cada trilha é dividida em 9 partes chamadas setores. O DOS Kernel considera os setores com a unidade de dados básica do disco. Os setores são especificados por números, a partir do 0. Cada setor pode conter até 512 bytes.

CLUSTERS (AGLOMERADOS)

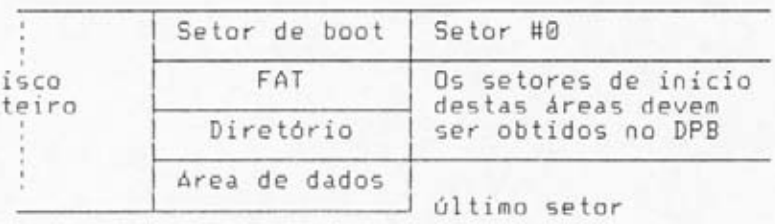
Embora consideradas as unidades básicas de informações do disco, não é por setores que o DOS Kernel controla os dados no disco, mas sim por unidades chamadas "clusters". Um cluster pode conter um ou mais setores. Normalmente, um cluster contém dois setores, tendo portanto o tamanho de 1 Kbyte.

OS DADOS NO DISCO

No MSXDOS, os setores do disco são divididos em quatro áreas, mostradas na tabela da página seguinte. Os dados propriamente ditos são colocados na "área de dados". O setor de boot é sempre o setor 0, mas os setores de início das outras áreas (FAT, diretório e área de dados) difere conforme o tipo de disco, mas estas informações estão contidas no DPB.

Conteúdo do disco

Setor de boot - Programa de partida do MSXDOS e informações
FAT - Controle físico da área de dados
Diretório - Informações sobre os arquivos no disco
Área de dados - Área para dados do usuário



O DPB E O SETOR DE BOOT

A sigla DPB vem do inglês "Drive Parameter Block", ou Bloco de Parâmetros do Drive. Para cada drive conectado, o MSXDOS aloca um DPB na RAM. As informações contidas do DPB são originalmente copiadas do setor de boot do disco durante a partida, mas alguns dados são diferentes entre o setor de boot e o DPB.

Off Set *SETOR DE BOOT*
0BH/0CH - Tamanho de um setor (em bytes)
0DH - Tamanho de um cluster (em setores)
0EH/0FH - Número de setores reservados
10H - Número de FATs
11H/12H - Número de entradas do diretório
13H/14H - Número de setores do disco
15H - Identificação do tipo de disco
16H/17H - Tamanho da FAT (em setores)
18H/19H - Número de setores por trilha
1AH/1BH - Número de faces do disco
1CH/1DH - Número de setores ocultos

Off Set *DPB*
+0 - Número do drive (0=A:, 1=B:, etc.)
+1 - Identificação do tipo de disco
+2/+3 - Tamanho do setor em bytes
+4 - Máscara do diretório
+5 - Tamanho do diretório em setores
+6 - Máscara do cluster
+7 - Tamanho do cluster em setores
+8/+9 - Primeiro setor da FAT
+10 - Número de FATs
+11 - Número de entradas do diretório
+12/+13 - Primeiro setor da área de dados
+14/+15 - Total de clusters do disco mais 1
+16 - Número de setores por FAT
+17/+18 - Primeiro setor da área do diretório
+19/+20 - Endereço da FAT na RAM

Para acessar as informações do DPB, use a função 1BH do BDOS, que entre outros dados, traz o endereço do DPB na RAM.

O FIB (MSXDOS2)

A sigla FIB vem do inglês "File Info Block", ou Bloco de Informações sobre o Arquivo. O FIB só existe para o MSXDOS2 e

é usado para operações mais complexas, como procurar diretórios de arquivos desconhecidos ou subdiretórios. É uma área de 64 bytes na RAM, contendo informações sobre as entradas de diretórios ou de determinado arquivo ou subdiretório. Para obter as informações do FIB, use as funções 40H, 41H ou 42H do MSXDOS2.

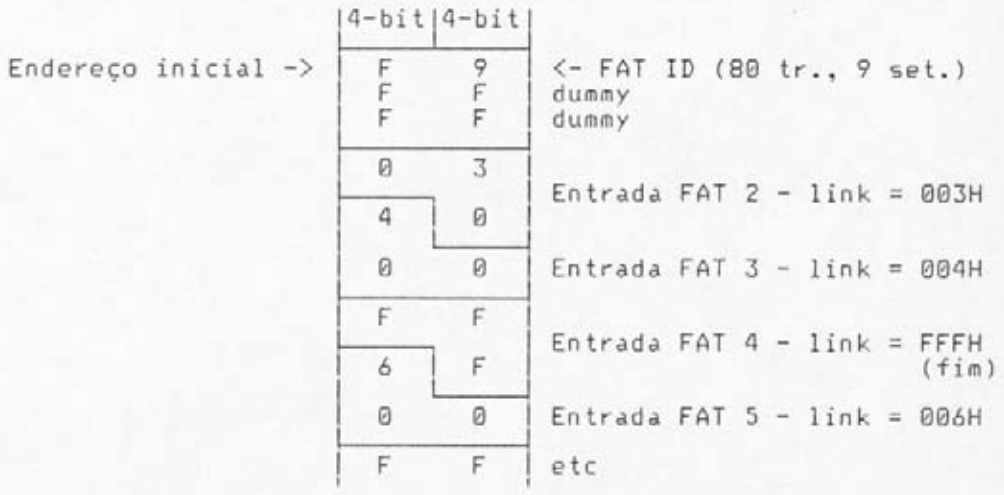
Off Set	*INFORMAÇÕES DO FIB*
+0	- Sempre FFH
+1/+13	- Nome do arquivo em ASCII
+14	- Byte de atributos do arquivo
+15/+16	- Hora da última modificação do arquivo
+17/+18	- Data da última modificação do arquivo
+19/+20	- Cluster inicial do arquivo
+21≈+24	- Tamanho do arquivo
+25	- Número do drive lógico
+26≈+63	- Informações internas (não modificar)

O byte FFH no início serve para distinguir o FIB de uma string pathname. Os dados do FIB são armazenados no mesmo formato dos dados do diretório. Veja a seção "DIRETÓRIO" mais adiante.

A FAT

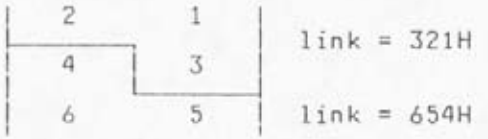
A sigla FAT quer dizer "File Allocation Table", ou Tabela de Alocação de Arquivos, e é uma espécie de mapa do disco. No MSXDOS, o cluster é a unidade básica de dados do disco. Para arquivos grandes, são usados vários clusters a fim de armazená-los. Porém, se vários arquivos são criados e apagados, ficam clusters vazios entre os arquivos não apagados. Quando um arquivo :aior é criado, ele é dividido em várias partes e estas são gravadas nos clusters disponíveis. É necessário, então, um meio para se saber em quantos e em quais setores está o arquivo desejado. Esta é a função da FAT.

Quando um cluster defeituoso é encontrado, a FAT também é usada para gravá-lo, mas o acesso não será mais possível. A informação sobre os clusters, principalmente os defeituosos, é necessária para o manuseio dos arquivos em disco. Sem essa informação, o disco inteiro não poderia ser usado. Por isso é que existem duas FATs, para recuperação em caso de apagamento acidental.



A figura da página anterior exemplifica a estrutura da FAT. O primeiro byte é chamado de "FAT ID" e indica o tipo de disco (o mesmo valor contido no setor de boot e no DPB). Os próximos dois bytes contêm o "dummy". A partir do quarto byte (endereço inicial + 3), a informação sobre os clusters (link) é gravada em um formato irregular de 12 bits por cluster. Cada 12 bits de informação é chamado de "entrada da FAT". O número de entrada da FAT é o número do cluster correspondente.

A informação "link" indica qual o próximo cluster do arquivo correspondente. O exemplo da página anterior mostra um arquivo com três clusters (cluster 2 -> cluster 3 -> cluster 4). Quando o valor "link" for FFFH, significa que o arquivo terminou. Na prática, os números "link" não ficam necessariamente em ordem numérica. Veja na ilustração abaixo como os números "link" são organizados na FAT.



O DIRETÓRIO

A FAT, descrita acima, armazena a localização física dos dados de um arquivo no disco, mas não contém qualquer informação sobre o conteúdo do mesmo. Por isso, existe uma seção do disquete chamada *diretório*, onde estão as informações sobre o arquivo. Cada entrada do diretório é composta por 32 bytes e contém o nome e atributos do arquivo, hora e data da criação do arquivo, número do primeiro cluster do arquivo e o tamanho do mesmo, como mostrado abaixo.

Off Set *ORGANIZAÇÃO DO DIRETÓRIO*

+0/+7	- Nome do arquivo (até 8 caracteres)
+8/+10	- Extensão (até 3 caracteres)
+11	- Byte de atributos do arquivo
+12≈+21	- Reservado (não utilizar)
+22/+23	- Hora da criação do arquivo
+24/+25	- Data da criação do arquivo
+26/+27	- Primeiro cluster do arquivo
+28≈+31	- Tamanho do arquivo em bytes

Byte de atributos -

b7	b6	b5	b4	b3	b2	b1	b0

b0 - Se este bit for "1", o arquivo poderá ser lido, mas não apagado ou modificado (somente MSXDOS2).

b1 - Se este bit for "1", o nome do comando não aparecerá no comando DIR ou FILES, mas poderá ser acessado normalmente (MSXDOS1 e MSXDOS2).

b2 - Igual a b1, mas as funções do BDOS não podem apagar ou modificar o arquivo e este não poderá ser acessado pelo COMMAND2.COM. Significa que é um arquivo de sistema. (Somente para MSXDOS2).

b3 - Se este bit for "1", os 11 bits do nome de arquivo conterão o nome do disco (volume name) e o restante do diretório será ignorado (Somente MSXDOS2).

b4 - Se este bit for "1", o arquivo é um subdiretório, e não poderá ser lido nem escrito normalmente. Quando

listado com o comando DIR, aparecerá a expressão "<DIR>" no lugar do tamanho do arquivo (MSXDOS1 e MSXDOS2, mas o MSXDOS1 não poderá acessar o subdiretório).

b5 - Se este bit for "1", o arquivo não poderá ser fechado antes de ser escrito (somente MSXDOS2).

b6 - Reservado (sempre 0).

b7 - Se este bit for "1", todos os outros serão ignorados e o FIB apontará para um caractere de dispositivo (ex: ".CON" - entrada de console). Só MSXDOS2.

	[23º byte]							[22º byte]						
Hora -	b7	b6	b5	b4	b3	b2	b1	b0	b7	b6	b5	b4	b3	b2	b1	b0
	h4	h3	h2	h1	h0	m5	m4	m3	m2	m1	m0	s4	s3	s2	s1	s0
	--hora(0≈23)--					----minuto(0≈59)---						segundo(0≈29)				

Para obter o valor correto dos segundos, multiplicar o valor do registro por 2.

	[25º byte]							[24º byte]						
Data -	b7	b6	b5	b4	b3	b2	b1	b0	b7	b6	b5	b4	b3	b2	b1	b0
	a6	a5	a4	a3	a2	a1	a0	m3	m2	m1	m0	d4	d3	d2	d1	d0
	------ano(0≈99)-----							--mês(1≈12)--				---dia(1≈31)--				

Para obter o ano correto, somar o valor do registro com o valor 1980 (1980 até 2079).

O primeiro setor do diretório pode ser obtido no DPB respectivo. Quando um arquivo é criado, a entrada respectiva do diretório é colocada na parte livre mais próxima do início da área do diretório. Cada entrada do diretório ocupa 32 bytes e é inicialmente preenchida com bytes 00H. Se um arquivo é criado e depois deletado (apagado), apenas o primeiro byte do diretório é modificado para E5H. Quando todas as entradas do diretório forem preenchidas, mais arquivos não podem ser criados mesmo que haja espaço disponível no disco. O número máximo de entradas do diretório também pode ser obtido no DPB respectivo.

2.2 - ACESSO AOS ARQUIVOS EM DISCO

O FCB

A sigla FCB vem do inglês "File Control Block", ou Bloco de Controle de Arquivo. Toda informação gravada no disco recebe o nome de arquivo. Cada arquivo recebe um nome, composto por até 8 caracteres mais uma extensão opcional de três (ex: MSXDOS .SYS). O acesso direto ao arquivo pelo diretório e pela FAT é muito complexo; por isso existe o FCB. O FCB ocupa 37 bytes de memória, bastando ao programador especificar o nome do arquivo e o drive para que seja possível acessar o arquivo desejado. O FCB pode estar localizado em qualquer lugar da memória, mas normalmente o MSXDOS utiliza o endereço 005CH para armazená-lo. Veja a seguir a estrutura do FCB.

Off Set	*COMENTARIOS*
+0	- Número do drive (0=default, 1=A:, 2=B:, etc.)
+1/+11	- Nome do arquivo e extensão
+12/+13	- Bloco atual
+14/+15	- Tamanho do registro aleatório (1 a 65535)
+16≈+19	- Tamanho do arquivo em bytes (1 a 4.294.967.297)

+20/+21 - Data (mesmo formato do diretório)
+22/+23 - Hora (mesmo formato do diretório)
+24 - ID do dispositivo
+25 - Localização do diretório
+26/+27 - Primeiro cluster do arquivo
+28/+29 - último cluster acessado
+30/+31 - Localização relativa do cluster
+32 - Registro seqüencial atual
+33/+36 - Número do registro aleatório

• Número do drive (00H)
Indica o disk-drive no qual está o disco que contém o arquivo (0=default; 1=A:; 2=B:; etc.).

• Nome do arquivo (01H a 08H)
O nome do arquivo pode conter até 8 caracteres. Quando tiver menos, os bytes restantes serão preenchidos com espaços.

• Extensão (09H a 0BH)
A extensão do nome do arquivo pode ter até 3 caracteres. Quando tiver menos, os bytes restantes serão preenchidos com espaços (20H). A extensão é opcional.

• Bloco atual (0CH a 0DH)
Indica o número do bloco atual para acesso seqüencial. Ver funções 14H e 15H do BDOS.

• Tamanho do registro aleatório (0EH a 0FH)
Especifica o tamanho da unidade de dados (registro) para leitura ou escrita aleatória (em bytes). Ver as funções 14H, 15H, 21H, 27H e 28H do BDOS.

• Tamanho do arquivo (10H a 13H)
Indica o tamanho do arquivo em bytes.

• Data (14H a 15H)
Indica a data do último acesso ao arquivo. O formato é igual ao do diretório.

• Hora (16H a 17H)
Indica a hora do último acesso ao arquivo. O formato é igual ao do diretório.

• ID do dispositivo (18H)
Quando um periférico é aberto como um arquivo, o valor listado na tabela abaixo é especificado neste byte. Para arquivos normais, o valor deste campo é de 40H + número do drive. Por exemplo, o ID do drive A: é 41H. (Para futuras expansões, programas aplicativos não devem usar o byte ID).

Byte ID	Dispositivo
FFH	CON (console ou teclado)
FEH	AUX (auxiliar)
FDH	NUL (nulo)
FCH	LST (listar na impressora)
FBH	PRN (impressora)

• Localização do diretório (19H)
Indica a posição da entrada no diretório do arquivo.

• Primeiro cluster do arquivo (1AH a 1BH)
Indica o número do primeiro cluster do arquivo no disco.

• último cluster acessado (1CH a 1DH)
Indica o número do último cluster acessado.

• Localização relativa do cluster (1EH a 1FH)
Indica a localização relativa do último cluster acessado a partir do primeiro cluster do arquivo.

• Registro seqüencial atual (20H)
Indica o número do registro atual para acesso seqüencial. Ver funções 14H e 15H do BDOS.

• Número do registro aleatório (21H a 24H)
Especifica o registro aleatório a ser acessado. Especificando o valor de 1 a 63 para o tamanho do registro, todos os 4 bytes, de 21H a 24H são usados, mas apenas três bytes, de 21H a 23H, têm significado quando o tamanho do registro é maior que 63. Ver as funções 14H, 15H, 21H, 22H, 27H e 28H do BDOS.

ABRINDO UM ARQUIVO

Um procedimento especial é necessário para abrir um arquivo usando o FCB. "Abrir um arquivo" significa transformar uma informação incompleta contida no FCB (apenas o nome do arquivo e o número do drive) em todas as informações que o FCB pode conter.

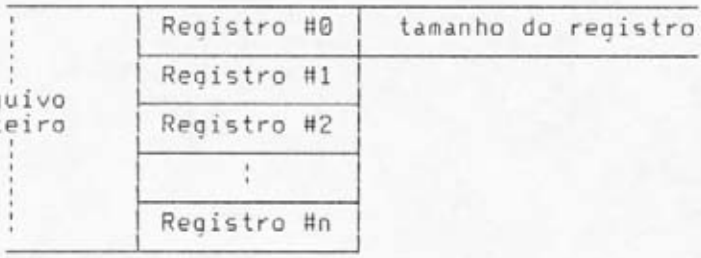
O FCB não aberto contém apenas o nome do arquivo e o número do drive. Ao ser aberto, o número do drive é convertido para drive real (1 a 6 ou 1 a 8) e os seguintes campos são preenchidos: tamanho do arquivo, data, hora, ID do dispositivo, localização do diretório, primeiro cluster do arquivo, último cluster acessado e localização relativa do cluster.

FECHANDO UM ARQUIVO

Quando um arquivo é aberto, a informação contida no diretório é transferida para o FCB. Durante o manuseio do arquivo (quando é feita uma gravação, por exemplo), o conteúdo dos campos do FCB vão sendo modificados. Por isso, após ter completado o manuseio do arquivo, é necessário fechá-lo. A operação de fechar um arquivo faz com que a informação contida no FCB volte para o diretório atualizada, a fim de possibilitar acessos posteriores.

ACESSO SEQÜENCIAL E ALEATÓRIO

O acesso seqüencial difere do acesso aleatório. Neste último, o acesso aos registros do arquivo pode ser feito livremente a qualquer um deles. Já no acesso seqüencial, como o próprio nome diz, os registros são acessados um após o outro, impreterivelmente. O tamanho de cada registro pode ser qualquer um, desde que seja igual ou maior que um byte. O registro pode ter, inclusive, o tamanho do arquivo inteiro (acesso seqüencial extremo) ou de apenas um byte (acesso aleatório extremo). O valor default para o tamanho de cada registro é de 128 bytes. Para maiores detalhes de como acessar os arquivos, ver as funções respectivas do BDOS. Veja na página seguinte uma ilustração de um arquivo com os respectivos registros.



ARQUIVOS HANDLE (MSXDOS2)

Um arquivo handle nada mais é que um número que o usuário associa a um dispositivo ou arquivo comum. O valor de um arquivo handle pode variar de 0 a 63. Usando apenas o número handle como referência, pode-se manipular o arquivo ou dispositivo a ele associado, usando funções acrescentadas para o MSXDOS2, como as funções 43H, 44H, 45H, 52H e outras.

A área de memória interna usada pelos arquivos handle é alocada em uma página (16K) fora da área da TPA, não reduzindo, portanto, o tamanho desta.

Os arquivos handle de 0 a 4 são pré-definidos, como descrito abaixo.

0 - Entrada standard (CON)
1 - Saída standard (CON)
2 - Entrada/saída standard de erro (CON)
3 - Entrada/saída auxiliar standard (AUX)
4 - Saída standard para impressora (PRN)

2.3 - ACESSO AO HD (WINCHESTER)

O acesso ao Hard Disk (HD), também conhecido como Winchester ou Disco Rígido, é feito da mesma forma que para os disquetes comuns, mudando o byte de ID para F0H do Hard Disk, mas apenas o MSXDOS2 suporta o acesso ao HD.

Observe, entretanto, que há uma limitação quanto à quantidade de memória que pode ser acessada no HD. Os setores no HD também possuem 512 bytes de tamanho, e os clusters têm 1 setor de tamanho. No diretório e no FCB o número de clusters é especificado em dois bytes, possibilitando o acesso a até 65536 clusters. Assim, pode ser acessado no máximo 65536 * 512 = 33.554.432 bytes, ou 32 Mbytes por partição, ou por ID de drive (A:, etc.). Como podem haver até 7 IDs de drive (de A: até G:, sendo que H: é a RAMDISK), podemos ter no máximo 32 * 7 = 224 Mbytes de HD.

Algumas interfaces usam o sistema de mapeamento particionado, que funciona de forma semelhante à Memória Mapeada. Se forem usados 4 bits, podemos mapear de 0 a 15 partições para cada ID de drive, dando um total de 15 * 32 = 480 Mbytes por ID, ou até 7 * 480 = 3360 Mbytes de Winchester. Se forem usados 8 bits, podemos mapear até 256 partições para cada ID, totalizando 256 * 32 = 8192 Mbytes por ID de drive, e até 7 * 8192 = 57344 Mbytes de memória de massa em HD.

É aconselhável, no entanto, usar apenas 6 IDs de drive, para deixar um ID livre para o drive comum de disquetes.

3 - AS FUNÇÕES DO BDOS (System Call Usage)

O BDOS consiste em um conjunto de rotinas que fazem as operações básicas de I/O para os dispositivos de disco. Essas rotinas permitem fácil acesso ao sistema de disco e residem na interface. Também são conhecidas como DOS Kernel.

As funções do BDOS estão disponíveis tanto para o MSXDOS quanto para o DISK-BASIC, variando apenas o endereço de chamada:

```
MSXDOS:     0005H
DISK-BASIC: F37DH (&HF37D)
```

Para executar as funções do BDOS, basta simplesmente fazer o seguinte:

1- Carregar o registrador C da CPU com o número da função desejada;
2- Carregar os registradores A, B, DE e HL (se necessário) com os valores adequados;
3- Fazer uma chamada (CALL) para o endereço do BDOS (0005H para o MSXDOS e F37DH para o DISK-BASIC).

Veja o exemplo abaixo:

```
LD   A,00H    ;carrega A com o valor 00H
LD   C,01FH   ;carrega C com a função número 1FH
CALL 00005H   ;executa a função
```

As funções do BDOS estão descritas da seguinte forma:

```
FUNÇÃO (xxH) -> xxH = número da função
  Função:  Resumo da função que o BDOS realiza
  Setup:   valores a colocar nos registradores ou na memória
           pelo programador
  Retorno: valores de retorno na memória ou registradores de-
           pois da função ser executada
```

É importante observar que as chamadas para o BDOS destroem o conteúdo dos registradores. Portanto, antes de chamar alguma função, salve o conteúdo dos registradores que não devem ser modificados (na pilha, por exemplo).

Existem 44 chamadas para o BDOS no caso do MSXDOS1 e 94 para o MSXDOS2 (que inclui todas as funções do MSXDOS1). As funções são numeradas de 00H a 70H mas existem algumas que não estão implementadas: 1CH a 20H, 25H, 29H e 32H a 3FH. Uma chamada a estas funções apenas retorna o valor 0 no registrador A.

Um detalhe importantíssimo é que sempre que se for acessar o drive, deve-se usar as funções do BDOS, ou no mínimo as rotinas PHYDIO e FORMAT do BIOS. A acesso direto ao FDC deve ser evitado ao extremo, já que cada fabricante pode usar o FDC que melhor lhe convier, e os programas não funcionariam em interfaces diferentes. O acesso direto só é admissível nos casos em que não haja rotinas do BDOS desenvolvidas para a função desejada, como

no caso de formatadores especiais, por exemplo.

3.1 - MANIPULAÇÃO DE I/O

CONIN (01H)
  Função: Entrada de teclado.
  Setup: Nenhum.
  Retorno: A <- código do caractere do teclado.
  NOTA: Quando não houver entrada, haverá espera para a entrada de caracteres pelo teclado, com eco na tela (o caractere será impresso na tela). As seguintes seqüências de controle são checadas por esta rotina:
  CTRL + C -> Retorna o sistema ao nivel de comandos.
  CTRL + P -> Liga o eco para a impressora. Tudo o que for escrito na tela sairá também na impressora.
  CTRL + N -> Desliga o eco para a impressora.
  CTRL + S -> Causa uma parada de apresentação dos caracteres até que uma tecla seja pressionada.

CONOUT (02H)
  Função: Saida de caractere para o monitor.
  Setup: E <- código do caractere.
  Retorno: Nenhum.
  NOTA: Esta função apresenta na tela o caractere contido no registrador E. As quatro seqüências de controle descritas acima também são checadas.

AUXIN (03H)
  Função: Entrada externa auxiliar.
  Setup: Nenhum.
  Retorno: A <- código de caractere do dispositivo auxiliar.
  NOTA: O dispositivo auxiliar pode ser qualquer um (modem, por exemplo). Entretanto, esta função só funciona em dispositivos que seguem o padrão MSX, e no Brasil existem muitos dispositivos que não seguem o padrão, incluindo os modems. As quatro seqüências de controle também são checadas.

AUXOUT (04H)
  Função: Saida para dispositivo externo.
  Setup: E <- código do caractere a enviar.
  Retorno: Nenhum.
  NOTA: Esta função também checa as seqüências de controle.

LSTOUT (05H)
  Função: Saida de caractere para a impressora.
  Setup: E <- código do caractere a ser enviado.
  Retorno: Nenhum.
  NOTA: Esta função checa as quatro seqüências de controle.

DIRIO (06H)
  Função: Entrada ou saida de string.
  Setup: E <- código do caractere a ser impresso no monitor.
  Se for FFH, o caractere será recebido.
  Retorno: Quando o registrador E contiver o valor FFH na entrada, o código da tecla pressionada retornará em A. Se A retornar o valor 00H, não foi pressionada nenhuma tecla.
  NOTA: Esta função não suporta caracteres de controle, mas checa as quatro seqüências de controle.

DIRIN (07H)
  Função: Leitura do teclado com espera I.
  Setup: Nenhum.
  Retorno: A <- código ASCII do caractere lido.
  NOTA: O caractere lido é impresso na tela. Esta função não suporta caracteres de controle.

INNOE (08H)
  Função: Leitura do teclado com espera II.
  Setup: Nenhum.
  Retorno: A <- código ASCII do caractere lido.
  NOTA: Esta função é idêntica à anterior, exceto que o caractere lido não é enviado para a tela.

STROUT (09H)
  Função: Saida de string.
  Setup: DE <- Endereço inicial da string a ser enviada.
  Retorno: Nenhum.
  NOTA: O caractere ASCII 24H ($) marca o final da string a ser enviada e não será impresso na tela. Esta função checa as quatro seqüências de controle.

BUFIN (0AH)
  Função: Entrada de string.
  Setup: DE <- deve apontar para um buffer com a seguinte estrutura:
    DE+0 -> número de caracteres a ler.
    DE+1 -> número de caracteres efetivamente lidos.
    DE+2 em diante: códigos dos caracteres lidos.
  Retorno: O segundo byte do buffer apontado por DE contém o número de caracteres efetivamente lidos e do terceiro byte em diante estão os códigos dos caracteres lidos.
  NOTA: A leitura dos caracteres termina ao ser pressionada a tecla RETURN. Se o número de caracteres ultrapassar o máximo apontado por DE, estes serão ignorados e será emitido um "beep" para cada caractere extra. Esta função checa as quatro seqüências de controle.

CONST (0BH)
  Função: Checagem do status do teclado.
  Setup: Nenhum.
  Retorno: Se alguma tecla foi pressionada, o registrador A retorna com o valor FFH, caso contrário, retorna com o valor 00H.
  NOTA: Esta função checa as quatro seqüências de controle.

3.2 - DEFINIÇÃO E LEITURA DE PARAMETROS

TERM0 (00H)
  Função: Reset do sistema.
  Setup: Nenhum.
  Retorno Nenhum.
  NOTA: Quanto esta função for chamada sob o DOS, provocará a recarga do MSXDOS. Quando for chamada sob o DISK-BASIC, provocará um reset total do sistema.

CPMVER (0CH)
  Função: Leitura da versão do sistema.
  Setup: Nenhum.

Retorno: HL <- 0022H
NOTA: Esta função apenas retorna em HL a versão do DOS instalado. No caso do MSX, retornará sempre o valor 0022H, indicando compatibilidade com o CP/M 2.2.

DSKRST (0DH)
Função: Reset do disco.
Setup: Nenhum.
Retorno: Nenhum.
NOTA: Esta função provoca uma atualização de todos os dasobre o disco contidos nos buffers do MSXDOS. Todos os buffers são apagados (FCB, DPB, etc.), o drive default será o A: e a DTA será setada em 0080H.

SELDSK (0EH)
Função: Selecionar o drive default.
Setup: E <- número do drive (A:=00H, B:=01H, etc.).
Retorno: A <- número de drives (de 1 a 8).
NOTA: Esta função simplesmente muda o número do drive default, ou seja, o drive que será acessado quando não houver especificação de drive. O número do drive corrente é armazenado no endereço 0004H.

LOGIN (18H)
Função: Leitura dos drives conectados ao micro.
Setup: Nenhum.
Retorno: HL <- drives conectados
NOTA: Esta função retorna nos bits de HL os drives que estão conectados ao micro, até um máximo de oito.

	[H]							[L]							
bit:	b7	b6	b5	b4	b3	b2	b1	b0	b7	b6	b5	b4	b3	b2	b1	b0
drive:	0	0	0	0	0	0	0	0	H:	G:	F:	E:	D:	C:	B:	A:

O bit conterá 0 se o drive não estiver conectado e 1 se o drive estiver conectado. Se B: contiver 1 e A: contiver 0 (b1=1 e b0=0), significa que há apenas um drive fisico funcionando como A: e B:. Observe que o registrador H sempre retornará com o valor 00H.

CURDRV (19H)
Função: Leitura do drive corrente (default).
Setup: Nenhum.
Retorno: A <- número do drive default (A:=00H: B:=01H, etc.).
NOTA: Esta função retorna no registrador A o número do drive atual (de 0 a 7).

SETDTA (1AH)
Função: Seta o endereço para transferência de dados.
Setup: DE <- endereço inicial da DTA (Disk Transfer Adress).
Retorno: Nenhum.
NOTA: No reset do sistema, a DTA é setada em 0080H, mas pode ser deslocada para qualquer área da memória com esta função. A DTA também é conhecida como DMA (Disk Memory Area).

ALLOC (1BH)
Função: Leitura de informações sobre o disco.
Setup: E <- número do drive desejado (0=default: 1=A:; etc.).

```
  Retorno: A = FFH se a especificação de drive for inválida,
               caso contrário:
           A =  número de setores lógicos por cluster:
           BC = tamanho do setor em bytes (normalmente 512):
           DE = número total de clusters no disco:
           HL = número de clusters livres (não usados):
           IX = endereço inicial do DPB na RAM;
           IY = endereço inicial da FAT na RAM.

GDATE  (2AH)
  Função:  Leitura da data.
  Setup:   Nenhum.
  Retorno: HL = ano (1980 a 2079);
           D =  mês (1=janeiro, 2=fevereiro, etc.);
           E =  dia do mês (1 a 31)
           A =  dia da semana (0=domingo, 1=segunda, etc.).

SDATE  (2BH)
  Função:  Modificar a data.
  Setup:   HL = ano (1980 a 2079);
           D =  mês (1=janeiro, 2=fevereiro, etc.);
           E =  dia do mês (1 a 31).
  Retorno: A = 00H se a especificação de data foi válida;
               FFH se a especificação foi inválida.

GTIME  (2CH)
  Função:  Leitura da hora.
  Setup:   Nenhum.
  Retorno: H = horas;
           L = minutos;
           D = segundos;
           E = centésimos de segundo.

STIME  (2DH)
  Função:  Modificar a hora.
  Setup:   H = horas;
           L = minutos;
           D = segundos;
           E = centésimos de segundo.
  Retorno: A = 00H se a especificação de hora foi válida;
               FFH se a especificação foi inválida.

VERIFY (2EH)
  Função:  Verificação de escrita no disco.
  Setup:   E <- igual a 0 para desativar o modo de verificação
                de escrita no disco;
           E <- qualquer valor diferente de 0 ativa a verifi-
                cação de escrita no disco.
  Retorno: Nenhum.
  NOTA:    Quando a verificação de escrita estiver ativada,
           logo após uma gravação no disco o sistema automa-
           ticamente fará uma checagem para verificar se a es-
           crita foi bem sucedida. Na carga do sistema, a
           função de verificação é desativada. Infelizmente,
           nas interfaces brasileiras, esta função pode va-
           riar, tornando-a incompatível com o padrão MSX.
           Dessa forma, fica ao encargo do programador a deci-
           são de utilizá-la ou não.
```

3.3 - LEITURA/ESCRITA ABSOLUTA DE SETORES

O MSX acessa o disco através de "setores lógicos". Os setores lógicos são definidos independentemente dos setores físicos do disco, e são numerados de 0 até um máximo que depende do tipo de disco:

```
40 trilhas, 1 face:  0 a 359
40 trilhas, 2 faces: 0 a 719
80 trilhas, 1 face:  0 a 719
80 trilhas, 2 faces: 0 a 1439
```

As funções do BDOS descritas abaixo acessam diretamente os setores lógicos do disco.

```
RDABS  (2FH)
  Função:  Leitura de setores lógicos do disco.
  Setup:   DE <- número do primeiro setor lógico a ler;
           H  <- número de setores a ler;
           L  <- número do drive (0=A:, 1=B:, etc.).
  Retorno: A <- se contiver 0, a leitura foi bem sucedida;
               outro valor será o código de erro.
  NOTA:    Esta função lê os setores continuamente até atingir
           o total especificado no registrador H ou detectar
           algum erro. Os setores lidos são colocados a partir
           da DTA.

WRABS  (30H)
  Função:  Escrita de setores lógicos no disco.
  Setup:   DE <- número do primeiro setor lógico a ser escrito;
           H  <- número de setores a escrever;
           L  <- número do drive (0=A:, 1=B:, etc.).
  Retorno: A <- se contiver 0, a escrita foi bem sucedida;
               outro valor será o código de erro.
  NOTA:    Os dados a serem escritos no disco serão lidos na
           RAM a partir do endereço inicial da DTA.
```

3.4 - ACESSO AOS ARQUIVOS USANDO O FCB

Acessar os arquivos do disco usando as funções do BDOS descritas até agora é um processo muito complicado. As funções do BDOS que acessam o disco usando o FCB tornam essas operações muito simples.

Existem três categorias de acesso aos arquivos através do FCB: acesso seqüencial, acesso aleatório e acesso aleatório em blocos. O acesso aleatório em blocos possui as seguintes facilidades: registros de qualquer tamanho podem ser especificados; o acesso aleatório pode ser feito em múltiplos registros e o tamanho do arquivo é controlado em bytes.

Uma observação importante é que três funções não funcionam corretamente quando o FCB estiver situado entre os endereços 4000H a 7FFFH para o MSXDOS1 e MSXDOS2:

1. Função 11H
2. Função 12H
3. Funções de I/O para dispositivos (CON,PRN,NUL,AUX)

FOPEN (0FH)
  Função: Abrir arquivo (FCB).
  Setup: DE <- endereço inicial de um FCB não aberto.
  Retorno: Se A contiver 00H, a operação foi bem sucedida;
  Se A contiver FFH, houve algum problema.
  NOTA: Quando um arquivo é aberto usando esta função, todos os campos do FCB (exceto o tamanho do registro, bloco atual, registro atual e registro aleatório) são preenchidos com os dados contidos no diretório do disco.

FCLOSE (10H)
  Função: Fechar arquivo (FCB).
  Setup: DE <- endereço inicial de um FCB aberto.
  Retorno: Se A contiver 00H, a operação foi bem sucedida;
  Se A contiver FFH, houve algum problema.
  NOTA: Ao ser executada, esta função transfere os dados contidos no FCB para o diretório. é absolutamente necessário chamar esta função após a gravação de novos registros em um arquivo, caso contrário as entradas do diretório não serão atualizadas, com a conseqüente perda de todos os dados do arquivo.

SFIRST (11H)
  Função: Procurar o primeiro arquivo.
  Setup: DE <- endereço inicial de um FCB não aberto.
  Retorno: Se A contiver 00H, o arquivo foi encontrado;
  Se A contiver FFH, o arquivo não foi encontrado.
  NOTA: Caso o arquivo tenha sido encontrado, a entrada correspondente no diretório é copiada na DTA e o número do drive do FCB é setado (33 bytes são usados), e o FCB pode ser aberto na própria DTA. Os caracteres coringa podem ser usados. Por exemplo, uma especificação como "????????.BAT" fará com que todos os arquivos com a extensão .BAT sejam procurados, e o primeiro a ser encontrado terá seus dados transferidos para a DTA. Para procurar todos os arquivos com a extensão .BAT, use a função 12H descrita abaixo.

SNEXT (12H)
  Função: Procurar o próximo arquivo.
  Setup: Nenhum.
  Retorno: Se A contiver 00H, o arquivo foi encontrado;
  Se A contiver FFH, nenhum arquivo foi encontrado.
  NOTA: Caso o arquivo tenha sido encontrado, a entrada correspondente do diretório é copiada na DMA e o número do drive do FCB é setado. Esta função foi especialmente criada para uso com caracteres coringa (ex.: ????????.BAT da função 11H), pois ao ser chamada novamente, ela procura o próximo arquivo no diretório que coincida com a especificação dada. Esta função só deve ser usada após o uso da função 11H para setar os parâ- metros necessários. Ela é útil quando se quer procurar vários arquivos que tenham partes de seus nomes iguais.

FDEL (13H)
  Função: Apagar arquivos.
  Setup: DE <- endereço inicial de um FCB aberto.

Retorno: Se A contiver 00H, a operação foi bem sucedida;
Se A contiver FFH, houve algum problema.
NOTA: Esta função também aceita caracteres coringa ("?") na especificação do FCB para apagar mais de um arquivo simultaneamente.

RDSEQ (14H)
Função: Leitura seqüencial.
Setup: DE <- endereço inicial de um FCB aberto.
Bloco atual no FCB -> bloco inicial para leitura.
Registro atual no FCB -> registro inicial para leitura.
Retorno: Se A contiver 00H, a leitura foi bem sucedida;
Se A contiver 01H, houve erro de leitura.
NOTA: Quando a leitura for bem sucedida, o registro lido será colocado na DTA. Além disso, o bloco e registro atuais do FCB são automaticamnete incrementados para facilitar a próxima leitura. O tamanho de cada registro é fixado em 128 bytes.

WRSEQ (15H)
Função: Escrita seqüencial.
Setup: DE <- endereço inicial de um FCB aberto.
Bloco atual no FCB <- bloco inicial para escrita.
Registro atual no FCB <- registro inicial para escrita.
128 bytes iniciais da DTA <- dados a serem escritos.
Retorno: Se A contiver 00H, a escrita foi bem sucedida;
Se A contiver 01H, houve erro de escrita.
NOTA: O bloco e registro atuais do FCB são automaticamente incrementados após a escrita a fim de facilitar a escrita seqüencial.

FMAKE (16H)
Função: Criar arquivos.
Setup: DE <- endereço inicial de um FCB não aberto.
Retorno: Se A contiver 00H, a operação foi bem sucedida.
Se A contiver FFH, houve erro na criação do arquivo.
NOTA: O tamanho do registro, o bloco e registro atuais e o registro aleatório do FCB podem ser setados após executar esta função.

FREN (17H)
Função: Renomear arquivos.
Setup: DE <- endereço inicial do FCB com o nome do arquivo a ser renomeado. Na primeira posição do FCB deve ser colocado o número do drive seguido do nome do arquivo a ser renomeado. A partir do 18º byte (FCB + 11H) até o 28º deve ser colocado o novo nome do arquivo.
Retorno: Se A contiver 00H, a renomeação foi feita com sucesso;
Se A contiver FFH, houve erro na renomeação.
NOTA: O caractere coringa "?" pode ser usado pelo atual e pelo novo nome de arquivo, para renomear vários arquivos simultaneamente. Por exemplo, especificando "????????.MAC" para os arquivos a renomear e "????????.OBJ" para o novo nome de arquivo, todos os arquivos com a extensão ".MAC" serão renomeados com a extensão ".OBJ".

RDRND (21H)
Função: Leitura aleatória.

```
  Setup:   DE <- endereço inicial de um FCB aberto.
           Registro aleatório no FCB <- nº do registro a ler.
  Retorno: Se A contiver 00H, a leitura foi bem sucedida;
           Se A contiver 01H, houve erro de leitura.
  NOTA:    O registro lido será colocado na área indicada
           pela DTA. O tamanho do registro é fixado em 128 bytes.

WRRND  (22H)
  Função:  Escrita aleatória.
  Setup:   DE <- endereço inicial de um FCB aberto.
           Registro aleatório no FCB <- nº do registro a escrever.
           128 bytes a partir da DTA <- dados a serem escritos.
  Retorno: Se A contiver 00H, a escrita foi bem sucedida;
           Se A contiver 01H, houve erro de escrita.

FSIZE  (23H)
  Função:  Ler o tamanho do arquivo.
  Setup:   DE <- endereço inicial de um FCB aberto.
  Retorno: Se A contiver 00H, a operação foi bem sucedida;
           Se A contiver FFH, houve erro de leitura.
  NOTA:    O tamanho do arquivo é especificado nos três pri-
           meiros bytes no campo de tamanho do arquivo alea-
           tório do FCB em incrementos de 128 bytes. Assim, se
           um arquivo conter de 1 a 128 bytes, o valor retor-
           nado será 1; se conter de 129 a 256 bytes, o valor
           retornado será 2; se conter de 257 a 384 bytes o
           valor retornado será 3, e assim por diante.

SETRND (24H)
  Função:  Setar campo do registro aleatório.
  Setup:   DE <- endereço inicial de um FCB aberto.
           Bloco atual no FCB <- número do bloco desejado.
           Registro atual no FCB <- número do registro desejado.
  Retorno: A posição do registro atual desejada, calculada a
           partir do registro e bloco contidos no FCB, é colo-
           cada no campo de registro aleatório. Os três pri-
           meiros bytes registro aleatório são setados.

WRBLK  (26H)
  Função:  Escrita aleatória em bloco.
  Setup:   DE <- endereço inicial de um FCB aberto.
           HL <- número de registros a serem escritos.
           DTA <- dados a serem escritos.
           Tamanho do registro no FCB <- tamanho dos registros
                                         a serem escritos.
           Registro aleatório no FCB <- número do primeiro re-
                                        gistro a ser escrito.
  Retorno: Se A contiver 00H, a escrita foi bem sucedida.
           Se A contiver 01H, houve erro de escrita.
  NOTA:    Após a escrita, o número do registro aleatório é
           automaticamente incrementado para facilitar even-
           tuais escritas posteriores. O tamanho do registro
           pode ser qualquer um, desde 1 byte até 65535 bytes,
           setando o campo respectivo no FCB.

RDBLK  (27H)
  Função:  Acesso aleatório em bloco.
```

```
  Setup:   DE <- endereço inicial de um FCB aberto.
           HL <- número de registros a serem lidos.
           DTA <- endereço inicial para os dados lidos.
           Tamanho do registro no FCB <- tamanho dos registros
                                         a serem escritos.
           Registro aleatório no FCB <- número do primeiro re-
                                        gistro a ser lido.
  Retorno: Se A contiver 00H, a leitura foi bem sucedida.
           Se A contiver 01H, houve erro de leitura.
           HL <- número de registros efetivamente lidos, caso
                 o fim de arquivo seja atingido antes de todos
                 os registros serem lidos.

WRZER  (28H)
  Função:  Escrita aleatória com bytes 00H.
  Setup:   DE <- endereço inicial de um FCB aberto.
           Registro aleatório no FCB <- registro a ser escrito.
           128 bytes a partir da DTA <- dados a serem escritos.
  Retorno: Se A contiver 00H, a escrita foi bem sucedida.
           Se A contiver 01H, houve erro de escrita.
  NOTA:    O tamanho dos registros é fixado em 128 bytes. Esta
           função é igual à 22H, exceto pelo fato de preencher
           os registros restantes do arquivo com bytes 00H, se
           o registro especificado não for o último do arquivo.
```

3.5 - FUNÇÕES ADICIONADAS PARA O MSXDOS2

As funções do BDOS descritas a seguir foram adicionadas para o MSXDOS2 e não estão implementadas no MSXDOS1. O modo de chamada destas funções é exatamente igual ao das funções descritas até agora.

```
DPARM  (31H)
  Função:  Lê os parâmetros do disco.
  Setup:   DE <- endereço inicial de um buffer de 32 bytes.
           L <- número do drive (0=default, 1=A:, etc.).
  Retorno: A <- código de erro (se for 0, não houve erro).
           DE <- endereço inicial do buffer de parâmetros.
  NOTA:    Esta função retorna uma série de parâmetros do dis-
           co especificado. O formato do buffer de parâmetros
           é o seguinte:
  Off Set      Descrição resumida
  DE + 0       número do drive físico (1=A:, etc.)
  DE + 1≈2     tamanho de um setor em bytes (normalmente 512)
  DE + 3       número de setores por cluster
  DE + 4≈5     número de setores reservados
  DE + 6       número de FATs (normalmente 2)
  DE + 7≈8     número de entradas do diretório
  DE + 9≈10    número total de setores lógicos
  DE + 11      ID do disco
  DE + 12      número de setores por FAT
  DE + 13≈14   primeiro setor do diretório
  DE + 15≈16   primeiro setor da área de dados
  DE + 17≈18   número máximo de clusters
  DE + 19      dirty disk flag
  DE + 20≈23   volume ID (-1 = sem ID de volume)
  DE + 24≈31   reservado (normalmente 0)
```

```
FFIRST (40H)
  Função:  Procura primeira entrada.
  Setup:   DE <- endereço inicial do FIB ou de uma
                 string ASCII "drive/path/arquivo".
           HL <- endereço inicial do nome de arquivo (somente
                 quando DE apontar para o FIB).
           B <- atributos para procura (igual ao do diretório).
           IX <- endereço inicial de um novo FIB.
  Retorno: A <- código de erro (se for 0, não houve erro).
           IX <- endereço inicial do novo FIB preenchido.
  NOTA:    O bit "somente leitura" do byte de atributos para
           procura é ignorado. O nome de arquivo pode conter
           os caracteres coringa "?" e "*".

FNEXT  (41H)
  Função:  Procura próxima entrada.
  Setup:   IX <- endereço inicial do FIB.
  Retorno: A <- código de erro (se for 0, não houve erro).
           IX <- endereco inicial do novo FIB preenchido.
  NOTA:    Esta função só deve ser usada após a função 40H.
           Ela também aceita os caracteres coringa "?" e "*".
           Difere da anterior pelo fato de procurar todos os
           arquivos que tenham partes de seu nome iguais espe-
           cificadas através dos caracteres coringa, um após
           outro.

FNEW   (42H)
  Função:  Procura nova entrada.
  Setup:   DE <- endereço inicial do FIB ou de uma
                 string ASCII "drive/path/arquivo".
           HL <- endereço inicial de um nome de arquivo (so-
                 mente se DE apontar para o FIB).
           B <- b0≈b6 = atributos:
                b7 = cria nova flag.
           IX <- endereço inicial do novo FIB contendo o nome
                 de arquivo padrão.
  Retorno: A <- código de erro (se for 0, não houve erro)
           IX <- endereço inicial do FIB preenchido com a nova
                 entrada.
  NOTA:    Esta função é parecida com a função 40H. Mas ao in-
           vés de procurar a entrada do diretório, ela criará
           uma nova entrada com o mesmo nome. O FIB apontado
           por IX será preenchido com as informações da nova
           entrada. Se houver caracteres coringa no nome de
           arquivo, eles serão trocados por caracteres apro-
           priados pelo "nome de arquivo padrão". Se o bit
           "diretório" estiver setado na entrada (registrador
           B), será criado um subdiretório. Os outros bits
           serão copiados.

OPEN   (43H)
  Função:  Abre arquivo handle.
  Setup:   DE <- endereço inicial do FIB ou
                 string ASCII "drive/path/arquivo".
           A <- Modo abertura: b0=1 - não escrita;
                               b1=1 - não leitura;
                               b2=1 - inheritable (herdado);
                               b3≈b7 - devem ser "0".
```

```
  Retorno: A <- código de erro (se for 0, não houve erro).
           B <- novo arquivo handle.
  NOTA:    O FIB ou a string drive/path/arquivo preferencial-
           mente referem-se a um subdiretório ou a um nome de
           volume. O arquivo especificado é aberto para escri-
           ta e/ou leitura (dependendo do valor do registrador
           A) e o novo arquivo handle retorna no registrador
           B. Se o bit "inheritable" de A estiver setado, o
           arquivo handle será aberto por outro processo (ver
           função 60H).

CREATE (44H)
  Função:  Criar arquivo handle.
  Setup:   DE <- Drive/path/arquivo ou string ASCII.
           A <- Modo abertura: b0=1 - não escrita;
                               b1=1 - não leitura;
                               b2=1 - inheritable (herdado);
                               b3≈b7 - devem ser "0".
           B <- b0≈b6 = atributos;
                b7 = cria nova flag.
  Retorno: A <- código de erro (se for 0, não houve erro).
           B <- novo arquivo handle.
  NOTA:    O arquivo criado por esta função será automatica-
           mente aberto (função 43H). Se o arquivo for um sub-
           diretório, este não será aberto. Caso o registrador
           B retorne com o valor FFH, o arquivo handle criado
           não é válido.

CLOSE  (45H)
  Função:  Fechar arquivo handle.
  Setup:   B <- arquivo handle a fechar.
  Retorno: A <- código de erro (se for 0, não houve erro).

ENSURE (46H)
  Função:  Proteger arquivo handle.
  Setup:   B <- arquivo handle a ser protegido.
  Retorno: A <- código de erro (se for 0, não houve erro).
  NOTA:    Se o arquivo handle estiver protegido, o apontador
           do arquivo corrente não poderá ser modificado, mas
           se este for escrito, os campos hora, data, atribu-
           tos e dados bufferizados serão transferidos para o
           disco.

DUP    (47H)
  Função:  Duplicar arquivo handle.
  Setup:   B <- arquivo handle.
  Retorno: A <- código de erro (se for 0, não houve erro).
           B <- novo arquivo handle.
  NOTA:    Esta função cria uma cópia do arquivo handle espe-
           cificado. O novo arquivo handle referirá ao mesmo
           arquivo que o original. Se um dos arquivos handle
           for fechado ou tiver o apontador de arquivo modifi-
           cado, o outro também o terá.

READ   (48H)
  Função:  Ler de um arquivo handle.
  Setup:   B <- arquivo handle.
           DE <- endereço inicial do buffer.
           HL <- número de bytes a ler.
  Retorno: A <- código de erro (se for 0, não houve erro).
```

NOTA: O número de bytes especificado é lido do arquivo corrente e copiado para o buffer indicado por DE. Se o fim de arquivo for detectado antes do término da leitura, o número de bytes lidos retornará em HL e não será gerado erro. As quatro seqüências de controle (Ctrl+P, Ctrl+N, Ctrl+S e Ctrl+C) são checadas por esta função.

WRITE (49H)
Função: Escrever por um arquivo handle.
Setup: B <- arquivo handle.
DE <- endereço inicial do buffer.
HL <- número de bytes a escrever.
Retorno: A <- código de erro (se for 0, não houve erro).
HL <- número de bytes efetivamente escritos.
NOTA: Esta função é parecida com a anterior, mas escreve os dados ao invés de ler. Se o arquivo handle foi aberto com as flags de "não escrita" ou "não leitura", a função retornará com erro. Se o fim de arquivo for encon- trado, ele será extendido até o valor necessário. Os dados a escrever são retirados do buffer apontado por DE.

SEEK (4AH)
Função: Mover apontador do arquivo handle.
Setup: B <- arquivo handle.
A <- código do método.
DE:HL <- sinalização de offset.
Retorno: A <- código de erro (se for 0, não houve erro).
DE:HL <- novo apontador de arquivo.
NOTA: O apontador de arquivo associado com o arquivo handle será alterado de acordo com o código do método como descrito abaixo:
A=0 -> relativo ao início do arquivo;
A=1 -> relativo à posição corrente;
A=2 -> relativo ao final do arquivo.
Se houver mais de um arquivo handle criado pela função 47H, todos serão alterados da mesma forma.

IOCTL (4BH)
Função: Controle para dispositivos de I/O.
Setup: B <- arquivo handle.
A <- código de subfunção:
00H - ler status do arquivo handle;
01H - setar modo ASCII/binário;
02H - testa se disposit. está pronto p/ entrada;
03H - testa se disposit. está pronto p/ saída;
04H - calcula tamanho da screen.
DE <- outros parâmetros.
Retorno: A <- código de erro (se for 0, não houve erro).
DE <- outros valores de retorno.

NOTA: Esta função retorna vários aspectos dos arquivos handle, principalmente se este se refere a um arquivo ou a um dispositivo. Se A for igual a 0 na entrada, então o regitrador DE deve ser carregado com os seguintes parâmetros:

• Para dispositivos: b0=1 - dispositivo de entrada;
b1=1 - dispositivo de saída;
b2≈b4 - reservados;
b5=1 - modo ASCII;
b5=0 - modo binário;
b6=1 - fim de arquivo;
b7=1 - (sempre 1) = dispositivo;
b8≈b15 - reservados.

• Para arquivos: b0≈b5 - número do drive (0=A:, etc.);
b6=1 - fim de arquivo;
b7=0 - (sempre 0) = arquivo de disco;
b8≈b15 - reservados.

No retorno, DE apresentará os mesmos valores. Se A for igual a 1, deve ser especificado apenas o bit 5 de DE; os demais bits serão ignorados. Se A for igual a 2 ou 3, o registrador E retornará com o valor 00H se o dispositivo não estiver pronto e com FFH se o dispositivo estiver pronto. Se A for igual a 4, DE retornará com o valor lógico do tamanho da tela para o arquivo handle (D=número de linhas e E=número de colunas). Para dispositivos que não a tela, DE retornará com o valor 0000H.

HTEST (4CH)
Função: Testar arquivo handle.
Setup: B <- arquivo handle.
DE <- apontador para o FIB ou para string ASCII "drive/path/arquivo".
Retorno: A <- código de erro (se for 0, não houve erro).
B <- 00H = não é o mesmo arquivo;
FFH = é o mesmo arquivo.
NOTA: Esta função testa se o arquivo handle em B se refere ao arquivo apontado por DE. Se se referir ao mesmo arquivo, B retornará com o valor FFH, caso contrário retornará com 00H.

DELETE (4DH)
Função: Apagar arquivo ou subdiretório.
Setup: DE <- apontador para o FIB ou para string ASCII "drive/path/arquivo".
Retorno: A <- código de erro (se for 0, não houve erro).
NOTA: Um subdiretório só poderá ser apagado se não conter nenhum nome de arquivo. Se um nome de arquivo for especificado, não retornará erro, mas, é claro, o dispositivo não será "apagado".

RENAME (4EH)
Função: Renomear arquivo ou subdiretório.
Setup: DE <- apontador para o FIB ou para string ASCII "drive/path/arquivo".
HL <- apontador para o novo nome em ASCII.
Retorno: A <- código de erro (se for 0, não houve erro).
NOTA: O novo nome apontado por HL não deverá conter a especificação de drive e/ou diretório path. Se um nome de dispositivo for especificado, não retornará código de erro, mas o nome de dispositivo não será modificado. O FIB não será modificado.

```
MOVE   (4FH)
  Função:  Mover arquivo ou subdiretório.
  Setup:   DE <- apontador para o FIB ou para string ASCII
                 "drive/path/arquivo".
           HL <- apontador para nova string path em ASCII.
  Retorno: A <- código de erro (se for 0, não houve erro).
  NOTA:    Esta função move o arquivo ou subdiretório
  apontado por DE para o diretório especificado pela string
  path apontada por HL. A string path não deve conter
  especificação de drive. Se um subdiretório for movido, todas
  as suas entradas com os respectivos arquivos serão movidos
  junto. Um arquivo não poderá ser movido se o arquivo handle
  respectivo estiver aberto. O FIB do arquivo movido não será
  atualizado.

ATTR   (50H)
  Função:  Setar ou ler atributos de um arquivo.
  Setup:   DE <- apontador para o FIB ou para string ASCII
                 "drive/path/arquivo".
           A <- 0 = lê atributos; 1 = escreve atributos.
           L <- novo byte de atributos (se A = 1).
  Retorno: A <- código de erro (se for 0, não houve erro).
           L <- byte de atributos atual.
  NOTA:    Se A=0, o byte de atributos do arquivo ou sub-
  diretório retornará no registrador L. Os atributos de um
  arquivo não podem ser modificados se o arquivo handle
  correspondente estiver aberto.

FTIME  (51H)
  Função:  Ler ou setar data e hora em um arquivo.
  Setup:   DE <- apontador para o FIB ou para string ASCII
                 "drive/path/arquivo".
           A <- 0 = ler data e hora;
                1 = setar data e hora.
           IX <- nova hora (se A=1).
           HL <- nova data (se A=1).
  Retorno: A <- código de erro (se for 0, não houve erro).
           DE <- hora do arquivo corrente.
           HL <- data do arquivo corrente.
  NOTA:    Se A=1, a data e hora do arquivo serão modificadas
  de acordo com o valor dos registradores IX e HL. Se A=0, a
  data e hora do arquivo apontado por DE retornarão em DE e
  HL. O formato da data e da hora é igual ao do diretório.

HDELET (52H)
  Função:  Apagar arquivo handle.
  Setup:   B <- arquivo handle.
  Retorno: A <- código de erro (se for 0, não houve erro).
  NOTA:    Esta função apaga um arquivo handle. Se houver
  outro arquivo handle aberto para o mesmo arquivo, então esse
  não poderá ser apagado.

HRENAM (53H)
  Função:  Renomear arquivo handle.
  Setup:   B <- arquivo handle.
           HL <- novo nome de arquivo em ASCII.
  Retorno: A <- código de erro (se for 0, não houve erro).
```

NOTA: Esta função renomeia o arquivo associado com o arquivo handle especificado. O arquivo não poderá ser renomeado se houver outro arquivo handle aberto para o mesmo arquivo. Esta função é idêntica à função 4EH, exceto pelo fato do registrador HL não poder apontar para um FIB.

HMOVE (54H)
Função: Mover arquivo handle.
Setup: B <- arquivo handle.
HL <- nova path em ASCII.
Retorno: A <- código de erro (se for 0, não houve erro).
NOTA: Esta função move o arquivo associado ao arquivo handle especificado para o diretório especificado pela nova string path apontada por HL. O arquivo não poderá ser movido se houver outro arquivo handle aberto para o mesmo arquivo. Esta função é idêntica à função 4FH, exceto pelo fato do registrador HL não poder apontar para um FIB.

HATTR (55H)
Função: Ler ou setar atributos do arquivo handle.
Setup: B <- arquivo handle.
A <- 0 = ler atributos:
1 = setar atributos.
L <- novo byte de atributos (se A=1).
Retorno: A <- código de erro (se for 0, não houve erro).
L <- byte de atributos corrente.
NOTA: Esta função seta ou lê os atributos do arquivo associ- ado ao arquivo handle especificado. O byte de atributos não poderá ser setado se houver outro arquivo handle aberto para o mesmo arquivo. Esta função é parecida com a 50H.

HFTIME (56H)
Função: Ler ou setar hora e data do arquivo handle.
Setup: B <- arquivo handle.
A <- 0 = ler data e hora:
1 = setar data e hora.
IX <- nova hora (se A=1).
HL <- nova data (se A=1).
Retorno: A <- código de erro (se for 0, não houve erro).
DE <- hora corrente do arquivo.
HL <- data corrente do arquivo.
NOTA: Esta função lê ou seta a data e a hora do arquivo associado ao arquivo handle especificado. Se houver outro arquivo handle aberto para o mesmo arquivo, a data e a hora não poderão ser modificadas. Esta função é idêntica à função 51H, exceto pelo fato de não haver apontador: somente o arquivo handle.

GETDTA (57H)
Função: Ler o endereço da DTA (Disk Transfer Adress).
Setup: Nenhum.
Retorno: DE <- endereço inicial da DTA.

GETVFY (58H)
Função: Ler flag de verificação de escrita.
Setup: Nenhum.
Retorno: B <- 0 = verificação de escrita desativada;
1 = verificação de escrita ativada.

```
GETCD  (59H)
  Função:  Ler diretório ou subdiretório corrente.
  Setup:   B <- número do drive (0=default; 1=A:, etc.).
           DE <- endereço inicial de um buffer de 64 bytes.
  Retorno: A <- código de erro (se for 0, não houve erro).
           DE <- preenchido de acordo com a path corrente.
  NOTA:    Esta função simplesmente retorna no buffe apontado
  por DE o nome do diretório corrente em ASCII. Não são in-
  cluídos o nome do drive e o caractere "\". Se não houver di-
  retório corrente, o buffer será preenchido com bytes 00H.

CHDIR  (5AH)
  Função:  Trocar o subdiretório corrente.
  Setup:   DE <- string ASCII "drive/path/nome".
  Retorno: A <- código de erro (se for 0, não houve erro).
  NOTA:    Esta função simplesmente troca o diretório ou sub-
  diretório corrente pelo apontado pelo registrador DE.

PARSE  (5BH)
  Função:  Analisa pathname (nome da path).
  Setup:   B <- flag do nome do volume (bit 4).
           DE <- string ASCII para análise.
  Retorno: A <- código de erro (se for 0, não houve erro).
           DE <- apontador para o caractere de finalização.
           HL <- apontador para o início do último item.
           B <- flags de análise.
           C <- drive lógico (1=A:, etc.).
  NOTA:    O bit 4 do registrador B na entrada deve estar
  setado para string "drive/volume" ou resetado (0) para
  string "drive/ path/arquivo". O valor retornado em HL apon-
  tará para o primeiro caractere do último item da string. Por
  exemplo, para uma string "A:\XYZ\P.Q /F", DE apontará para o
  espaço em branco antes de "/F" e HL apontará para "P". As
  flags retornadas no registrador B são as seguintes:
  b0=1 se algum caractere apontar para outro nome de drive;
  b1=1 se algum diretório path for especificado;
  b2=1 se nome de drive for especificado;
  b3=1 se arquivo mestre for especificado no último item;
  b4=1 se extensão do nome do arquivo for especificada
       no último item;
  b5=1 se o último item for ambíguo;
  b6=1 se o último item for "." ou "..";
  b7=1 se o último item for "..".

PFILE  (5CH)
  Função:  Analisar nome de arquivo.
  Setup:   DE <- string ASCII a ser analisada.
           HL <- apontador para um buffer de 11 bytes.
  Retorno: A <- sempre 00H.
           DE <- apontador para o caractere final.
           HL <- apontador para o buffer preenchido.
           B <- flags de análise.
  NOTA:    A string ASCII apontada por DE deve ser um nome de
  arquivo simples, sem especificação de drive. Podem ser
  usados caracteres coringa (? e *). O significado das flags
  do registrador B são idênticos aos da função 5BH, exceto
  que os bits 0, 1 e 2 sempre serão 0.

CHKCHR (5DH)
  Função:  Checa caractere.
```

Setup: D <- flags do caractere.
E <- caractere a ser checado.
Retorno: A <- Sempre 00H.
D <- flags atualizadas do caractere.
E <- caractere checado.
NOTA: Esta função também checa caracteres de 16 bits e manipula nomes de arquivos. As flags do caractere são as seguintes:
b0=1 para suprimir o caractere;
b1=1 se for o primeiro byte de um caractere de 16 bits;
b2=1 se for o segundo byte de um caractere de 16 bits;
b3=1 nome do volume ou preferencialmente nome de arquivo;
b4=1 caractere de arquivo/volume não válido;
b5=b7 reservados (sempre 0).
Se o bit 0 for 1, o caractere retornado em E será sempre o mesmo; se for 0, poderá ser trocado de acordo com a linguagem setada na máquina. Para analisar um caractere de dois bytes, deve-se enviar o primeiro byte e depois o segundo, setando a flag correspondente. O bit 4 será setado no retorno se o caractere for um terminador de nome de arquivo ou volume.

WPATH (5EH)
Função: Ler string path completa.
Setup: DE <- apontador para um buffer de 64 bytes.
Retorno: A <- código de erro (se for 0, não houve erro).
DE <- preenchido com a string path completa.
HL <- apontador para o início do último item.
NOTA: Esta função simplesmente copia a string path ASCII corrente para o buffer apontado por DE. A string retornada não contém a especificação de drive e o caractere "\" inicial. O registrador HL aponta para o primeiro caractere do último item, exatamente como na função 5BH. Para maior confiabilidade, deve- -se primeiro chamar a função 40H ou 41H e depois chamar WPATH duas vezes, já que outras funções podem alterar os dados.

FLUSH (5FH)
Função: Descarregar buffers de disco.
Setup: B <- especificação de drive (0=default; 1=A:, etc.).
D <- 00H = somente descarregar;
FFH = descarregar e invalidar.
Retorno: A <- código de erro (se for 0, não houve erro).
NOTA: Esta função descarrega todos os buffers para o drive especificado, ou para todos os drives se B=FFH na entrada. Se o registrador D for FFH, todos os buffers do drive especificado serão também invalidados.

FORK (60H)
Função: Ramificar arquivos em árvore.
Setup: Nenhum.
Retorno: A <- código de erro (se for 0, não houve erro).
B <- ID do processo de ramificação.
NOTA: Novos arquivos handle são criados e os arquivos handle correntes que estão abertos no modo "inheritable" (ver função 43H) são copiados para os novos arquivos handle. Os arquivos handle standard (00H=05H) são copiados impreterivelmente. Pelo fato de haver uma cópia dos arquivos handle originais, se algum deles for fechado, poderá ser reaberto sem problemas.

JOIN (61H)
Função: Reunir arquivos em árvore.
Setup: B <- ID do processo de ramificação (ou 0).
Retorno: A <- código de erro (se for 0, não houve erro).
B <- código de erro primário do ramo.
C <- código de erro secundário do ramo.
NOTA: Esta função retorna para o arquivo handle original o arquivo handle copiado pela função anterior. O arquivo copiado é automaticamente fechado e o arquivo handle original é reativado. Se o registrador B for 00H na entrada, uma reinicialização parcial do sistema é feita: todos os arquivos handle copiados são fechados e os arquivos handle originais são reativados. Se esta função for chamada pelo endereço F37DH, os registradores B e C não retornarão o código de erro. Ver a função 62H.

TERM (62H)
Função: Finalizar com código de erro.
Setup: B <- código de erro para finalização.
Retorno: Nenhum.
NOTA: Esta função termina o programa com o código de erro especificado. A operação desta função é diferente conforme o endereço de chamada (0005H para MSXDOS ou F37DH para DISK-BASIC). Se for chamada por 0005H, a rotina de saída deve ser definida pela função 63H com o código de erro especificado (0 no caso de código de erro secundário) e se não houver rotina de saída definida pelo usuário, o sistema fará um jump para o endereço 0000H, provocando uma partida a quente do DOS. O interpretador de comandos somente imprimirá a mensagem de erro na tela se esta estiver entre 20H e FFH, mas não abaixo de 20H. Se esta função for chamada por F37DH, o controle será passado para o Interpretador BASIC que imprimirá a mensagem de erro.

DEFAB (63H)
Função: Definir rotina de abortagem (saída).
Setup: DE <- endereço inicial da rotina de abortagem; o endereço default é 0000H.
Retorno: A <- sempre 00H.
NOTA: Esta rotina somente estará disponível se for chamada por 0005H. Ela não deve ser chamada por F37DH. A rotina aponta- da por DE também será chamada no caso do sistema detectar as teclas Ctrl-C ou Ctrl-STOP pressionadas ou se houver erro de disco abortado.

DEFER (64H)
Função: Definir rotina para erro de disco.
Setup: DE <- endereço inicial da rotina de erro de disco. O valor default é 0000H.
Retorno: A <- Sempre 00H.
NOTA: Esta função especifica o endereço de uma rotina criada pelo usuário caso ocorra algum erro de disco. Esta função deve ser usada com muito cuidado. A especificação dos parâmetros e resultados da rotina estão especificados abaixo.
Parâmetros: A <- código de erro:
B <- número do drive físico:

```
                C <- b0=1 se for erro de escrita:
                     b1=1 se ignorar o erro (não recomendado):
                     b2=1 se for sugerida abortagem automática:
                     b3=1 se o número do setor é válido:
               DE <- número do setor do disco (se b3 de C for 1).
  Retorno:     A <- 0 = chama rotina de erro do sistema:
                     1 = Aborta:
                     2 = Tenta novamente:
                     3 = Ignora.

ERROR  (65H)
  Função:  Pegar código de erro antecipadamente.
  Setup:   Nenhum.
  Retorno: A <- sempre 00H.
           B <- código de erro da função.
  NOTA:    Esta função pode ser usada para prevenir o tipo de
  erro que poderá ocorrer na próxima chamada de função.

EXPLN  (66H)
  Função:  Pegar mensagem do código de erro.
  Setup:   B <- código de erro.
           DE <- apontador para um buffer de 64 bytes.
  Retorno: A <- sempre 00H.
           B <- código de erro ou 00H.
           DE <- buffer preenchido com a mensagem de erro.
  NOTA:    Esta função simplesmente retorna no buffer apontado
  por DE a mensagem ASCII de erro. Se a mensagem de erro for
  do tipo "System error 194" ou "User error 45", o registrador
  B retornará com o valor 0.

FORMAT (67H)
  Função:  Formatar um disco.
  Setup:   B <- número do drive (0=default; 1=A:, etc.).
           A <- 00H = retorna mensagem de escolha:
                01H≈09H = formata com esta escolha:
                0AH≈0DH = ilegal:
                FEH≈FFH = novo setor de boot.
          HL <- apontador para o buffer (se A = 1≈9).
          DE <- tamanho do buffer (se A = 1≈9).
  Retorno: A <- código de erro (se for 0, não houve erro).
           B <- slot da mensagem escolhida (só se A=0 na
                entrada).
           HL <- endereço da mensagem escolhida (só se A=0).
  NOTA:    Esta função é usada para formatar um disco e tem
  três diferentes opções de acordo com o valor passado em A.
  Se A=0, os registradores B e HL retornarão com o número do
  slot e endereço da mensagem ASCII interna do DOS. Se A for
  igual a 01H≈09H o sistema formatará o disco sem apresentar
  mensagem e neste caso os registradores HL e DE devem especi-
  ficar o buffer usado pelo disk-drive. Se A=FFH, o disco não
  será formatado, mas será atualizado para o MSXDOS2. Se
  A=FEH, o disco também não será formatado e somente os para-
  metros do disco serão atualizados para o MSXDOS2.

RAMD   (68H)
  Função:  Criar ou apagar a ramdisk.
  Setup:   B <- 00H = apaga a ramdisk:
                01H≈FEH = cria nova ramdisk:
                FFH = retorna tamanho da ramdisk.
```

Retorno: A <- código de erro (se for 0, não houve erro).
B <- tamanho da ramdisk.
NOTA: Se o registrador B for FFH na entrada, ele somente retornará com o número de segmentos de 16K (nº de páginas lógicas) alocados para a ramdisk. Se for 00H, apagará a ramdisk. Se B contiver entre 01H e FEH na entrada, será criada a ramdisk usando o número de páginas lógicas (segmentos de 16K) especificado em B. A ramdisk sempre será o drive "H:".

BUFFER (69H)
Função: Alocar buffers.
Setup: B <- 00H = retorna número de buffers alocados:
01H≈A5H = aloca o número especificado de buffers.
Retorno: A <- código de erro (se for 0, não houve erro).
B <- número corrente de buffers.
NOTA: Se o registrador B for 00H na entrada, o sistema simplesmente retornará em B o número atual de buffers. Se B conter de 1 a 20, o sistema alocará o número de buffers requerido: ca- so a memória seja insuficiente, será alocado o número possível de buffers retornando o número em B. Não será gerado código de erro. Note que o número máximo de buffers que podem ser alocados é 20. Cada buffer ocupa uma página lógica (16 Kbytes) fora do segmento normal de 64 Kbytes de RAM, não afetando, portanto, o tamanho da TPA.

ASSIGN (6AH)
Função: Atribuir drive lógico.
Setup: B <- número do drive lógico (1=A:, etc.).
D <- número do drive físico (1=A:, etc.).
Retorno: A <- código de erro (se for 0, não houve erro).
D <- número do drive físico.
NOTA: Esta função atribui o drive lógico ao drive físico especificado. Se B e D variarem de 1 a 7, então uma nova atribuição será feita. Se B e D forem 0, todas as atribuições serão canceladas. Se D for 0 e B conter de 1 a 7, a atribuição do drive lógico respectivo será cancelada. Se D for FFH e B conter de 1 a 7, o número de drive lógico especificado em B simples- mente retornará em D.

GENV (6BH)
Função: Ler item externo.
Setup: HL <- apontador para nome string em ASCII.
DE <- apontador do buffer para valor.
B <- tamanho do buffer.
Retorno: A <- código de erro (se for 0, não houve erro).
DE <- apontador para o buffer preenchido.
NOTA: Esta função lê o valor corrente do item externo cujo nome é apontado pelo registrador HL. Se o tamanho do buffer for pequeno, o valor de retorno será truncado com o último caractere valendo 00H. Um buffer de 255 bytes sempre será suficiente.

SENV (6CH)
Função: Setar item externo.
Setup: HL <- apontador para o nome em ASCII.
DE <- apontador para o valor a ser setado.
Retorno: A <- código de erro (se for 0, não houve erro).

NOTA: Esta função seta um novo item externo. A string de valor não pode conter mais de 255 caracteres e deve ser terminada com um byte 00H. Se a string de valor for nula, o item externo será removido.

FENV (6DH)
Função: Procurar item externo.
Setup: DE <- número do item externo.
HL <- apontador do buffer para o nome em ASCII.
Retorno: A <- código de erro (se for 0, não houve erro).
HL <- apontador para o buffer preenchido.
NOTA: Esta função é usada para procurar o item externo cujo número está no registrador DE. O primeiro item corresponde a DE=1. O nome do item externo especificado em DE retornará no buffer apontado por HL, sendo o último caractere um byte 00H.

DSKCHK (6EH)
Função: Ativar ou desativar checagem do disco.
Setup: A <- 00H = ler valor de checagem do disco:
01H = setar valor de checagem do disco.
B <- 00H = ativa (se A=01H):
01H = desativa (se A=01H).
Retorno: A <- código de erro (se for 0, não houve erro).
B <- valor de checagem do disco corrente.
NOTA: Se A=00H, o valor de checagem do disco corrente retornará em B. Se B for 0, a checagem do disco está ativada; se for outro valor, a checagem do disco está desativada. O valor default é ativada. Quando a checagem estiver ativada, o sistema recarregará o boot, a FAT, o FIB, o FCB, etc. do disco toda vez que este for trocado. Se a checagem estiver desativada, isto não ocorrerá. Portanto, é conveniente sempre deixar a checagem de disco ativa.

DOSVER (6FH)
Função: Ler o número da versão do MSXDOS.
Setup: Nenhum.
Retorno: A <- código de erro (se for 0, não houve erro).
BC <- versão do DOS Kernel.
DE <- versão do MSXDOS2.SYS.
NOTA: Os valores retornados nos registradores BC e DE estarão em BCD. Assim, se a versão for 2.34, por exemplo, o valor retornado será 0234H. Para compatibilidade com o MSXDOS1 verifique primeiro se houve algum erro (A<>0). Se houver erro, o MSXDOS não está totalmente instalado. Se não houver erro, verifique o registrador B. Se for menor que 2, a versão é anterior à 2.00 e os valores de C e DE são indefinidos. Se B for igual ou maior que 2, os valores de BC e DE serão válidos.

REDIR (70H)
Função: Ler ou setar o estado de redirecionamento.
Setup: A <- 00H = ler estado de redirecionamento:
01H = setar estado de redirecionamento.
B <- novo estado: b0 - entrada standard:
b1 - saída standard.

NOTA: Esta função foi implementada primariamente para rotinas de erro de disco e outros caracteres e I/O que devem ser redirecionados. As funções 01H a 0BH normalmente se referem ao console. Mas elas podem ser redirecionadas para arquivos em disco. O efeito desta função é temporário, no caso de A=01H e B=00H na entrada. Isto protegerá subseqüentes chamadas das fun- ções 01H a 0BH, que voltarão a serem direcionadas normalmente ao console. Se necessário, as funções podem ser redirecionadas novamente.

4 - ROTINAS DA INTERFACE DE DISCO

Existem algumas rotinas do BDOS que são chamadas diretamente da interface de disco. Estas rotinas possuem sua entrada na página 1, e por isso nunca devem ser chamadas diretamente, pois sob o MSXDOS a página 1 contém a RAM e sob o BASIC a página 1 contém a ROM do Interpretador e portanto a ROM do DOS Kernel nunca estará ativa normalmente.

Por isso, as rotinas da interface devem ser chamadas pela rotina CALSLT do BIOS, que está ativa tanto sob o MSXDOS como sob o MSX-BASIC. A seqüência de chamada deve ser a seguinte:

```
CALSLT: EQU  0001CH      :Endereço da rotina CALSLT
HFILE:  EQU  0FE7BH      :Endereço do hook do comando FILES
CALBAS: EQU  04022H      :Endereço da rotina CALBAS
        LD   IX,CALBAS   :Carrega IX com o endereço de CALBAS
        LD   IY,HFILE    :Carrega IY com o slot da interface
        CALL CALSLT      :Executa a rotina CALBAS
```

Observe que é necessário saber o slot onde se encontra a ROM da interface de disco para utilizar a rotina CALSLT. Um ponto seguro para se obter esta informação são os hooks dos comandos de disco, que contêm o slot da interface em seu segundo byte. Observe que são os 8 bits mais altos de IY que devem ser carregados com o ID do slot da interface de disco, portanto basta carregar IY com o endereço inicial do hook escolhido. No exemplo, foi escolhido o hook HFILE, do comando BASIC FILES.

As rotinas estão listadas da seguinte forma:

```
NOME   (ENDEREÇO)
  Função:  Função da rotina.
  Entrada: Parâmetros de chamada da rotina.
  Saída:   Parâmetros de retorno da rotina.
```

Todos os registradores serão modificados, portanto salve os registradores que não devem ser modificados na pilha antes de chamar qualquer rotina da interface de disco.

4.1 - DESCRIÇÃO DAS ROTINAS DA INTERFACE

```
DISKIO (4010H / Interface de disco)
  Função:  Leitura/escrita direta de setores.
  Entrada: HL - apontador para a TPA.
           DE - número do primeiro setor a ser lido ou escrito
           B - número de setores a ler ou escrever.
           C - ID da formatação do disco (F0H = Hard Disk -
               somente para MSXDOS2).
```

A - número do drive (0=A:, etc.).
Flag CY - resetada para efetuar leitura;
setada para efetuar escrita.

Saída: B - número de setores efetivamente transferidos.
A - código de erro (ver lista abaixo).
Flag CY - setada em caso de erro:
resetada se não houve erro.

OBS.: *Os códigos de erro retornados em A são os seguintes:*
0 - protegido contra escrita;
2 - não pronto;
4 - erro de CRC (setor não acessível);
6 - erro de busca;
8 - cluster não encontrado;
10 - erro de escrita;
12 - erro de disco (ou drive não SCSI para MSXDOS2);
Códigos de erro adicionados para o MSXDOS2:
18 - disco não DOS;
20 - versão do MSXDOS incorreta;
22 - disco não formatado;
24 - disco trocado;
26 - erro de usuário 10;
Restantes: erro de disco.

DSKCHG (4013H / Interface de disco)

Função: Checar estado da troca de disco.

Entrada: A - número do drive (0=A:, etc.).
B - ID de formatação do disco (00H para MSXDOS2).
C - ID de formatação do disco (F0H = Hard Disk - somente para MSXDOS2).
HL - apontador para o DPB respectivo.

Saída: A - código de erro (ver listagem na rotina anterior).
B - se não houve erro: 00H - desconhecido;
01H - disco não trocado;
FFH - disco trocado.
Flag CY - setata em caso de erro:
resetada se não houve erro.

OBS.: Se o disco foi ou será (desconhecido) trocado, leia o setor de boot (ID de formatação) e transfira o novo DPB com a rotina GETDPB (4016H).

GETDPB (4016H / Interface de disco)

Função: Ler o DPB do disco.

Entrada: A - número do drive (0=A:, etc.).
B - primeiro byte da FAT (ID do disco).
C - ID de formatação do disco (F0H = Hard Disk - somente para MSXDOS2)
HL - apontador para o DPB.

Saída: HL - apontador para o DPB atualizado.
A - código de erro (ver rotina DISKIO - 4010H)
Flag CY - setada em caso de erro:
resetada se não houve erro.

CHOICE (4019H / Interface de disco)

Função: Mensagem para formatação de disco.

Entrada: Nenhuma.

Saída: HL - endereço do byte 00H que termina a string com o texto que contém a mensagem para formatação. Se não houver escolha (somente um tipo de formatação é suportado), HL retorna com 0000H.

```
DSKFMT (401CH / Interface de disco)
  Função:  Formatar um disco.
  Entrada: A - escolha especificada pelo usuário (ver CHOICE).
           D - número do drive (0=A:, etc.).
           HL - apontador para o início da área de trabalho.
           BC - tamanho da área de trabalho.
  Saída:   A - código de erro (ver listagem abaixo)
           Flag CY - setada em caso de erro:
                     resetada se não houve erro.
  OBS.:    Na formatação, o boot é escrito no setor 0, toda a FAT
           é limpa e à área do diretório é preenchida com zeros.
           Códigos de erro:
           0 - protegido contra escrita;
           2 - não pronto;
           4 - erro de CRC (setor não formata);
           6 - erro de busca;
           8 - cluster não encontrado;
           10 - falha de escrita (ou drive não SCSI p/ MSXDOS2);
           12 - parâmetro incorreto;
           14 - memória insuficiente;
           16 - outros erros.

CALBAS (4022H / Interface de disco)
  Função:  Chamar o Interpretador BASIC.
  Entrada: Nenhuma.
  Saída:   Nenhuma.

FORMAT (4025H / Interface de disco)
  Função:  Formatar um disco apresentando mensagem.
  Entrada: Nenhuma.
  Saída:   Nenhuma.

STPDRV (4029H / Interface de disco)
  Função:  Parar o motor dos drives.
  Entrada: Nenhuma.
  Saída:   Nenhuma.

SLTDOS (402DH / Interface de disco)
  Função:  Retorna o ID do slot do DOS Kernel.
  Entrada: Nenhuma.
  Saída:   A - ID do slot.

HIGMEM (4030H / Interface de disco)
  Função:  Retorna o endereço mais alto disponível da RAM.
  Entrada: Nenhuma.
  Saída:   HL - endereço mais alto disponível da RAM.

BLKDOS (40FFH / Interface de disco - somente MSXDOS2)
  Função:  Retorna bloco corrente do DOS2.
  Entrada: Nenhuma.
  Saída:   A - número do bloco corrente.

  OBS.:    Os 64 Kbytes da ROM do DOS Kernel 2 são divididos
           em 4 segmentos de 16 Kbytes cada. Estes segmentos
           podem estar ativos somente na página física 1.
           Portanto, e- les são trocados constantemente duran-
           te o processamento. Os valores retornados podem ser
           0, 1, 2 ou 3.
```

5 - A PÁGINA-ZERO

A página-zero é a área de memória situada entre os endereços 0000H e 00FFH, ocupando 256 bytes. Esta área só é ativa sob o MSXDOS e é de extrema importância para os programas aplicativos. Algumas rotinas do BIOS estão disponíveis nesta área. A página-zero é mapeada como descrito abaixo.

WBOOT (0000H)
Warm Boot. Ao se chamar esta rotina, promove-se uma partida a quente do MSXDOS, ou seja, sem que o micro seja totalmente resetado, o MSXDOS é recarregado.

DRIVE (0004H)
Este byte armazena o drive default (0=A:, 1=B:, etc.).

BDOS (0005H)
Este é o ponto de entrada para as rotinas do BDOS, descritas na seção 3.

RDSLT (000CH)
Esta rotina lê um byte em qualquer slot. Esta chamada é exatamente igual à rotina RDSLT do BIOS.

WRSLT (0014H)
Escreve em byte de dados em qualquer slot. Esta chamada é exatamente igual à rotina WRSLT do BIOS.

CALSLT (001CH)
Chama uma rotina em qualquer slot. No presente caso, pode ser usada para chamar rotinas do BIOS. Esta chamada é exatamente igual à rotina CALSLT do BIOS.

ENASLT (0024H)
Habilita uma página de qualquer slot. Esta chamada é exatamente igual à rotina ENASLT do BIOS.

CALLF (0030H)
Chama uma rotina em qualquer slot, com parâmetros em linha. No presente caso, pode ser usada para chamar rotinas do BIOS. Esta chamada é exatamente igual à rotina CALLF do BIOS.

INTPRT (0038H)
Chama rotina do manipulador de interrupção. Esta entrada não deve ser utilizada pelo programador.

NOTA: As rotinas descritas devem ser chamadas exatamente como se faz no BIOS, ou seja, através de uma instrução CALL ou RST, exceto a rotina WBOOT (0000H), que deve ser chamada com um JP 0000H.

CHSLTS (003BH a 005BH)
Rotina para troca de slots secundários. Não deve ser usada pelo programador.

FCBDOS (005CH a 007FH)
Esta área contém o FCB usado pelo BDOS.

DTA (0080H) Endereço inicial da DTA.

A área compreendida entre 0080H e 00FFH é onde é colocada uma linha coletada pelo COMMAND.COM. Por exemplo, se digitarmos um comando externo tipo "PROG ABC(1)", o COMMAND.COM procurará no disco o programa de nome PROG.COM e, se encontrar, o colocará a partir do endereço 0100H (primeiro byte após a página-zero), sendo que a execução do programa se inicia neste mesmo endereço. O argumento "ABC(1)" será colocado a partir do endereço 0080H, com a estrutura mostrada detalhadamente abaixo.

```
0080H -> byte 20H (espaço em branco)
0081H -> byte 0Dh (carriage return)
0082H -> "A"
0083H -> "B"
0084H -> "C"
0085H -> "("
0086H -> "1"
0087H -> ")"
0088H -> byte 0DH (carriage return)
0089H -> byte 00H (fim do argumento)
```

O conteúdo dos endereços 0006H e 0007H na página-zero é o endereço mais alto (+1) disponível para a TPA. A TPA (Transient Program Área) é a área onde o MSXDOS coloca os programas feitos pelo usuário e os executa. Ela se inicia no endereço 0100H, e é neste mesmo endereço que se inicia a execução do programa.

6 - ÁREA DE SISTEMA DE DISCO

A área de sistema de disco ocupa uma boa parte de memória logo abaixo da área de trabalho de sistema, que se inicia em F38 0H. O MSXDOS1 ocupa mais memória nessa área porque copia a FAT do disquete que está sendo utilizado e também do drive virtual B:, por isso ao pressionar a tecla CTRL durante o RESET, desativando o drive B:, há um aumento de 1.5 Kbytes na memória disponível. Já o MSXDOS2 copia a FAT em outra área de memória, e a economia em se desativar o drive B: é de apenas 21 bytes, referente ao DPB respectivo.

6.1 - ÁREA DE SISTEMA PARA O MSXDOS1

F197H.21
  DPB do drive A:

F1ACH.21
  DPB do drive B:

F1BDH.2
  Setor inicial do diretório no drive atualmente ativo.

F1BFH.2
  Endereço da FAT em RAM do drive atualmente ativo.

F1C9H.24
  Rotina para impressão na tela de uma string terminada por "$".
  Req. DE -> endereço inicial da string.

F1E2H.6
  Rotina para abortar o programa em caso de erro.

F1E8H.12
Rotina para acessar a DISKROM (???)

F1F4H.3
Jump para rotina de checagem do nome de arquivo.
Reg. HL -> endereço do primeiro caractere do
nome de arquivo.

F1F7H.4
Nome de dispositivo "PRN ".

F1FBH.4
Nome de dispositivo "LST ".

F1FFH.4
Nome de dispositivo "NUL ".

F203H.4
Nome de dispositivo "AUX ".

F207H.4
Nome de dispositivo "CON ".

F20BH.11
Reservado para novos nomes de dispositivos.

F216H.1
Número do dispositivo atual.
PRN = -5, LST = -4, ... CON = -1.

F22BH.12
Tabela contendo os códigos dos meses do ano.

F22B [1F] Janeiro	F231 [1F] Julho
F22C [1C] Fevereiro	F232 [1F] Agosto
F22D [1F] Março	F233 [1E] Setembro
F22E [1E] Abril	F234 [1F] Outubro
F22F [1F] Maio	F235 [1E] Novembro
F230 [1E] Junho	F236 [1F] dezembro

F237H.4
Usada internamente pela função 10 do BDOS.

F23BH.1
Flag para indicar se os caracteres devem ir para a impressora.
0=não: outro valor, sim

F23DH.2
Endereço atual da DTA.

F23FH.4
Número do setor atual do disco.

F243H.2
Apontador para o endereço do DPB do drive atual.

F245H.1
Setor atual relativo do diretório a partir do primeiro (0).

F246H.1
Drive que contém o setor atual do diretório (0=A, 1=B, etc).

F247H.1
  Drive default (0=A, 1=B, etc).

F24FH.3
  Jumo para a rotina que apresenta a mensagem "Insert disk for drive". Reg. A: número do drive (41H = A:, 42H = B:, etc).

F252H.3
  Hook chamado antes da execução de uma função do BDOS.

F264H.3
  Hook da rotina da função 'OPEN'.

F26AH.3
  Hook da rotina 'GETDPB' da interface.

F26DH.3
  Hook da rotina da função 'CLOSE'.

F270H.3
  Hook da rotina da função 'RDABS'

F273H.3
  Hook da rotina de erro no acesso a disco.

F273H.3
  Hook da rotina da função 'WRABS'.

F27CH.3
  Hook da rotina de multiplicação (HL=DE*BC).

F27FH.3
  Hook da rotina de divisão (BC=BC/DE, HL=resto).

F2ACH.3
  Hook da rotina da função 'BUFIN'

F2AFH.3
 Hook da rotina da função 'CONOUT'

F2B5H.3
  Hook da rotina de identificação do mês de fevereiro (28/29 dias).

F2B9H.11
  Nome de arquivo.

F2C4H.1
  Byte de atributos.

F2E1H.1
  Drive atual para escrita/leitura absoluta.

F304H.2
  Armazena o valor do registrador SP (Stack Pointer).

F306H.1
  Drive default para o MSXDOS (0=A:, 1=B:, etc).

F30DH.1
  Verificar flag (0 = desligada; NZ = ligada)

F30EH.1
  Formato da data: 0=aammdd; 1=mmddaa; 2=ddmmaa.

F30FH.4
  Usada pelo modo Kanji.

F338H.1
  Flag para indicar a presença de relógio interno
  (0 = não; NZ = sim).

F341H.1
  Slot da página 0 da RAM (formato igual a RDSLT - 000CH/BIOS)

F342H.1
  Slot da página 1 da RAM (formato igual a RDSLT - 000CH/BIOS)

F343H.1
  Slot da página 2 da RAM (formato igual a RDSLT - 000CH/BIOS)

F344H.1
  Slot da página 3 da RAM (formato igual a RDSLT - 000CH/BIOS)

F346H.1
  Flag para indicar a presença do MSXDOS no disquete
  (0=não; NZ=sim).

F347H.1
  Número total de drives lógicos no sistema.

F348H.1
  ID slot do DOS Kernel (formato igual a RDSLT - 000CH/BIOS)

F349H.2
  Apontador para uma cópia da FAT do drive B: (1.5 Kbytes)
  seguida por uma cópia da FAT do drive A: (1.5 Kbytes).

F34DH.2
  Apontador para uma cópia da FAT do drive default (1.5 Kbytes).

F34FH.2
  Apontador para uma área de 512 bytes usada como DTA do
  DISK-BASIC.

F351H.2
  Apontador para um buffer de 512 bytes usado para tranferên-
  cia de setores do disco.

F353H.2
  Apontador para o FCB do arquivo atual.

F355H.2
  Endereço do DPB do drive A:.

F357H.2
  Endereço do DPB do drive B:.

F359H.2
  Endereço do DPB do drive C:.

F35BH.2
  Endereço do DPB do drive D:.

F35DH.2
  Endereço do DPB do drive E:.

F35FH.2
  Endereço do DPB do drive F:.

F361H.2
  Endereço do DPB do drive G:.

F363H.2
  Endereço do DPB do drive H:.

F365H.3
  Rotina para leitura de slots primários.

F374H.3
  Jump para rotina de saida de dispositivo auxiliar.

F377H.3
  Jump para a rotina do comando "BLOAD".

F37AH.3
  Jump para a rotina do comando "BSAVE".

F37DH.3
  Jump para a chamada do BDOS.

6.1 - ÁREA DE SISTEMA PARA O MSXDOS2

F1E5H.3
  Jump para o manipulador de interrupção, somente durante o processamento das funções do BDOS.

F1E8H.3
  Jump para a rotina do BIOS RDSLT, somente durante o processamento das funções do BDOS.

F1EBH.3
  Jump para a rotina do BIOS WRSLT, somente durante o processamento das funções do BDOS.

F1EEH.3
  Jump para a rotina do BIOS CALSLT, somente durante o processamento das funções do BDOS.

F1F1H.3
  Jump para a rotina do BIOS ENASLT, somente durante o processamento das funções do BDOS.

F1F4H.3
  Jump para a rotina do BIOS CALLF, somente durante o processamento das funções do BDOS.

F1F7H.3
Jump para troca para o "Modo DOS" (páginas 0 e 2 para os segmentos do sistema).

F1FAH.3
Jump para troca para o "Modo Usuário".

F1FDH.3
Jump para a seleção dos segmentos do DOS Kernel na página 0.

F200H.3
Aloca um segmento de 16 Kbytes.

F203H.3
Libera um segmento de 16 Kbytes.

F206H.3
Lê um byte cujo endereço está no registrador HL. A <- byte lido.

F209H.3
Escreve um byte cuho endereço está no registrador HL. E <- byte a ser escrito.

F20CH.3
Chamada inter-segmento. Endereço em IX e IYh.

F20FH.3
Chamada inter-segmento. Endereço em linha após a instrução CALL.

F212H.3
Colocar segmento na página indicada no registrador HL.

F215H.3
Ler página do segmento atual. Retorno no registrador HL.

F218H.3
Colocar segmento na página 0.

F21BH.3
Ler segmento atual da página 0.

F21EH.3
Colocar segmento na página 1.

F221H.3
Ler segmento atual na página 1.

F224H.3
Colocar segmento na página 2.

F227H.3
Ler segmento atual da página 2.

F22AH.3
Página 3 não suporta mudança de segmento.

F22DH.3
Ler segmento atual na página 3.

F32CH.1
Drive lógico atual.

F23DH.2
Endereço atual da DTA.

F23FH.4
Número do setor atual para acesso.

F243H.2
Endereço do DPB do drive atual.

F245H.1
Número relativo do setor atual da área do diretório.

F246H.1
Número do drive do diretório atual (0=A:, 1=B:, etc).

F247H.1
Número do drive default (0=A:, 1=B:, etc).

F24FH.3
Jump para a rotina que apresenta a mensagem "Insert disk for drive" Reg. A: número do drive (41H = A:, 42H = B:, etc).

F252H.3
Hook chamado antes da execução de uma função do BDOS.
Página 0 -> mapa do bloco (F2D0H): página 2 -> mapa do bloco (F2CFH).

F261H.3
Hook da função 02H do BDOS.

F2B3H.2
Endereço da TPA definido pelo usuário. Os 32 bytes iniciais da TPA são usados para funções especiais.

Off set	*Descrição*
00H≈02H	- Reservados
03H	- Usado pelo VDP SPEED (bit 3 de F2B6H)
04H≈1FH	- Reservados
20H	- Expansão do BDOS e rotinas de interrupção

F2B6H.1
Byte de flags: b0 - reservado
b1 - reservado
b2 - reservado
b3 - VDP rápido - 0-sim, 1-não
b4 - End. TPA do usuário - 0-sim, 1-não
b5 - Reset - 0-não, 1-sim
b6 - Busreset - 0-sim, 1-não
b7 - Reboot - 0-não, 1-sim

F2B7H.1
Número da versão (normalmente 10H = v1.0).

F2C0H.5
Segundo hook da rotina de interrupção (usado pela DISK-ROM).

F2C5H.2
Endereço da tabela de mapeamento.

F2C7.1
Bloco atual da mapper na página 0.

F2C8.1
Bloco atual da mapper na página 1.

F2C9.1
Bloco atual da mapper na página 2.

F2CA.1
Bloco atual da mapper na página 3 (não pode ser trocado).

F2CBH.1
Cópia de F2C7H durante execução das rotinas do BDOS.

F2CCH.1
Cópia de F2C8H durante execução das rotinas do BDOS.

F2CDH.1
Cópia de F2C9H durante execução das rotinas do BDOS.

F2CEH.1
Cópia de F2CAH durante execução das rotinas do BDOS.

F2CFH.1
Número do último bloco de 16K disponivel da memória mapeada. Durante a execução das rotinas do BDOS, os blocos são trocados na página 2 (segmento de buffer).

F2D0H.1
Número do último bloco de 16K disponivel da memória mapeada. Durante a execução das rotinas do BDOS, os blocos são trocados página 0 (segmento de código).

F2D5H.5
Segundo hook EXTBIO (rotina do hook FCALL - FFCAH).

F2DAH.4
Endereço da segunda ROM BDOS para manipulação de funções.

F2DEH.4
Endereço da ROM do DOS2 para manipulação das funções do BDOS.

F2E6H.2
Buffer usado para armazenamento temporário do registrador IX.

F2E8H.2
Buffer usado para armazenamento temporário do registrador SP.

F2EAH.1
Status dos slots primários após a execução de uma função do BDOS.

F2EBH.1
Mesmo que F2EAH, mas para slots secundários.

F2ECH.1
Flag para checagem do status do disco (00H=off, FFH=on).

F2FBH.2
  Ponteiro para um buffer temporário durante a interpretação
  de um código de erro.

F2FDH.1
  Drive do qual o MSXDOS.SYS deverá ser carregado
  (1=A:, 2=B:, etc).

F2FEH.2
  Ponteiro do topo da pilha do buffer do DOS.

F300H.1
  Verificação de flag (00H=off, FFH=on).

F30DH.1
  Verificação da flag do disco (00H=off, FFH=on).

F313H.1
  Versão do DOS2 (ex. 22H = v2.2).

F33DH.3
  Comando BASIC LEN (acesso aleatório a arquivos).

F341H.1
  Slot da página 0 da RAM (formato igual a RDSLT - 000CH/BIOS)

F342H.1
  Slot da página 1 da RAM (formato igual a RDSLT - 000CH/BIOS)

F343H.1
  Slot da página 2 da RAM (formato igual a RDSLT - 000CH/BIOS)

F344H.1
  Slot da página 3 da RAM (formato igual a RDSLT - 000CH/BIOS)

F377H.3
  Jump para o segmento de sistema na página 0.
  O registrador HL deve conter o endereço.

F37AH.3
  Jump secundário para o segmento de sistema na página 0.

F37DH.3
  Jump para o manipulador de funções do BDOS.

7 - O SETOR DE BOOT

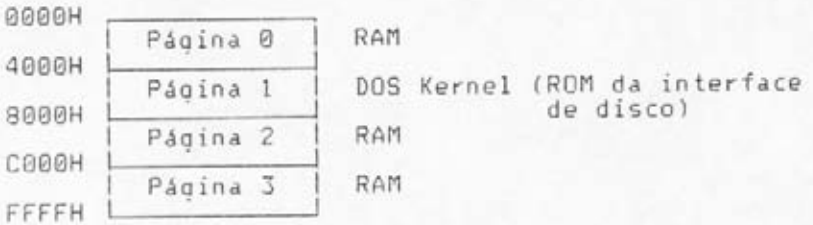
Em todos os disquetes, temos o chamado "setor de boot", que é sempre o setor 0 do disco. Toda vez que o micro é resetado, o DOS Kernel residente na ROM da interface de disco verifica se há algum disquete no drive. Se não houver, ativa o BASIC, se houver, carrega o setor de boot no endereço C000h (início da página 3 da RAM) e executa a rotina contida a partir do endereço C01EH. veja abaixo como o setor de boot fica na memória:

C000H -> byte 55H (byte de ID)
C001H/C002H -> FEH,90H (instrução de partida do DOS, usada no
                boot a quente - WBOOT).
C003H/C00AH -> Nome do fabricante ou identificação de formatação.
          em ASCII. Pode ser modificado pelo programador.

C00BH≈C01DH -> Dados do boot. Estes dados estão detalhadamente descritos na seção 2.
C01EH≈C0FFH -> Rotina de inicialização. Está descrita em detalhes mais adiante.
C100H≈C1FFH -> Área reservada - não utilizar.

Observe que, apesar do setor ter 512 bytes, as instruções contidas no mesmo só podem ter até 256 bytes (C000H a C0FFH) pois logo após a carga do boot, o DOS Kernel preenche a área a partir de C100H com rotinas específicas. Se o disco não for um disco de sistema, a página-zero também será preenchida, excetuando a rotina WBOOT (0000H) e a entrada do BDOS (0005H). Neste caso, use a chamada do BDOS no endereço F37DH.

O mapeamento da memória quando da execução do boot está ilustrado na página seguinte.



7.1 - A ROTINA DE INICIALIZAÇÃO

Logo após setar todos os dados necessários, o DOS Kernel passa o controle à rotina contida a partir do endereço C01EH. A rotina padrão, usada pelo DOS da Microsoft para o MSXDOS1, é a seguinte:

```
        BDOS:   EQU  0F37DH
C01E            RET  NC
C01F            LD   (BOOT1+1),DE
C023            LD   (0C0C4H),A
C026            LD   (HL),056H
C028            INC  HL
C029            LD   (HL),0C0H
C02B    BOOT0:  LD   SP,0F51FH
C02E            LD   DE,FCBDOS
C031            LD   C,00FH
C033            CALL BDOS
C036            INC  A
C037            JP   Z,BOOT2
C03A            LD   DE,00100H
C03D            LD   C,01AH
C03F            CALL BDOS
C042            LD   HL,00001H
C045            LD   (0C0ADH),HL
C048            LD   HL,03F00H
C04B            LD   DE,FCBDOS
C04E            LD   C,027H
C050            CALL BDOS
C053            JP   00100H
                :
C056            LD   E,B
C057            RET  NZ
```

```
C058  BOOT1:  CALL 00000H
C05B          LD   A,C
C05C          AND  0FEH
C05E          CP   002H
C060          JP   NZ,BOOT3
C063  BOOT2:  LD   A,(0C0C4H)
C066          AND  A
C067          JP   Z,04022H
              ;
C06A  BOOT3:  LD   DE,ERROR
C06D          LD   C,009H
C06F          CALL BDOS
C072          LD   C,007H
C074          CALL BDOS
C077          JR   BOOT0
C079  ERROR:  DEFB 'Boot error',00DH,00AH
              DEFB 'Press any key for retry',00DH,00AH,024H
C09F  FCBDOS: DEFB 000H,'MSXDOS  SYS'
C0AB          END
```

O restante do setor de boot é preenchido com bytes 00H.

Antes de mais nada, observe o seguinte: ao executar a rotina contida no boot, o DOS Kernel preenche com certos endereços os registradores DE e HL e seta um valor no registrador A.

O valor do registrador A é salvo para posterior utilização na rotina que chama o BASIC (BOOT2) ou na que promove o erro de boot (BOOT3).

O endereço que vem no registrador DE é colocado na instrução CALL de uma rotina que é chamada internamente pela interface de drive (logo à frente de BOOT1). No endereço apontado por HL, é colocado o endereço desta rotina que é chamada internamente pela interface (que no caso se inicia em C056H).

Logo após, a pilha (registrador SP) é colocada no endereço F51FH.

Após setar todos estes parâmetros, a rotina de boot procura o arquivo MSXDOS.SYS no disquete e caso o encontre carrega e executa o mesmo no endereço 0100H (a parte da rotina que vai do endereço C02EH a C053H). Caso não exista o arquivo MSXDOS.SYS, o controle é transferido para a rotina da interface que chama o BASIC (BOOT2).

A rotina de boot pode ser modificada pelo programador para adequá-la ao programa que quiser por no disco. Deve-se apenas ficar atento para o detalhe de que deve haver uma instrução RET NC no início, aos valores contidos em A, DE e HL, ao mapeamento inicial de memória (com a ROM do DOS Kernel na página 1) e à escassa área de memória que pode ser usada pelo boot, de apenas 222 bytes, do endereço C01EH a C0FFH.

Observe também que os três bytes iniciais do setor de boot (EBH, FEH, 90H) não devem ser modificados, apesar do sistema modificar o primeiro byte (EBH no disco) para 55H na memória. Os dados do setor de boot também devem estar setados (endereços C00BH a C01DH).

Observe abaixo a rotina padrão usada pelo MSXDOS2 da ASCII e preste atenção na semelhança entre as duas.

```
        BDOS:   EQU  0F37DH
  C01E          JR   BOOT0
  C020          DEFB 'VOL_ID'
  C026          DEFB 000H.015H.075H.005H.01BH
  C02B          DEFB 000H.000H.000H.000H.000H
                :
  C030  BOOT0:  RET  NC
  C031          LD   (BOOT2+1).DE
  C035          LD   (BOOT3+1).A
  C038          LD   (HL).067H
  C03A          INC  HL
  C03B          LD   (HL).0C0H
  C03D  BOOT1:  LD   SP.0F51FH
  C040          LD   DE.0C0ABH
  C043          LD   C.00FH
  C045          CALL BDOS
  C048          INC  A
  C049          JR   Z.BOOT3
  C04B          LD   DE.00100H
  C04E          LD   C.01AH
  C050          CALL BDOS
  C053          LD   HL.00001H
  C056          LD   (0C0B9H).HL
  C059          LD   HL.03F00H
  C05C          LD   DE.FCBDOS
  C05F          LD   C.027H
  C061          CALL BDOS
  C064          JP   00100H
                :
  C067          LD   L.C
  C068          RET  NZ
  C069  BOOT2:  CALL 00000H
  C06C          LD   A.C
  C06D          AND  0FEH
  C06F          SUB  02H
  C071  BOOT3:  OR   00H
  C073          JP   Z.04022H
  C076          LD   DE.ERROR
  C079          LD   C.009H
  C07B          CALL BDOS
  C07E          LD   C.007H
  C080          CALL BDOS
  C083          JR   BOOT1
                :
  C085  ERROR:  DEFB 'Boot error'.00DH.00AH
                DEFB 'Press any key for retry'.00DH.00AH.024H
  C0AB  FCBDOS: DEFB 000H.'MSXDOS  SYS'
  C0B6          END
```

Observe que, apesar das aparentes discrepâncias, esta rotina de boot é bem parecida com a anterior, sendo que todas as explicações dadas são válidas aqui, observando os endereços que foram modificados.

Esta rotina é mais versátil que a anterior, pois se encontrar apenas o MSXDOS1, o carregará e se encontrar o MSXDOS2, o carregará preferencialmente ao MSXDOS1. Além disso, se não encon-

trar o MSXDOS no disco, entrará para o DISK-BASIC versão 2.xx se o micro o possuir, senão entrará para o DISK-BASIC versão 1.xx. Em um micro MSX turbo R, vai mais além: entra para o DISK-BASIC versão 2.xx e ativa o modo Turbo (R800 DRAM). O modo Turbo também é ativado ao carregar o MSXDOS2, mas não no caso do MSXDOS1.

Capítulo 7
O RELÓGIO E A SRAM

Nos micros MSX2 ou superior, é usado o CLOCK-IC para as funções de relógio. Como o CLOCK-IC é alimentado por bateria, ele está sempre ativo, mesmo com o micro desligado. O relógio dispõe de uma pequena RAM adicional que é usada para armazenar algumas funções que o MSX realiza automaticamente ao ser ligado.

1 - FUNÇÕES DO CLOCK-IC

RELÓGIO

- Ler e atualizar os dados do ano, mês, dia do mês, dia da semana, horas, minutos e segundos.
- Apresentação da hora em 12 ou 24 horas.
- Meses de 30 e 31 dias são reconhecidos; o mês de fevereiro (28 dias) e os anos bissextos também são reconhecidos.

ALARME

- Quando o alarme estiver ativo, o relógio gera um sinal na hora escolhida.
- O alarme é setado como "XXdia, XXhoras, XXminutos".

MEMÓRIA

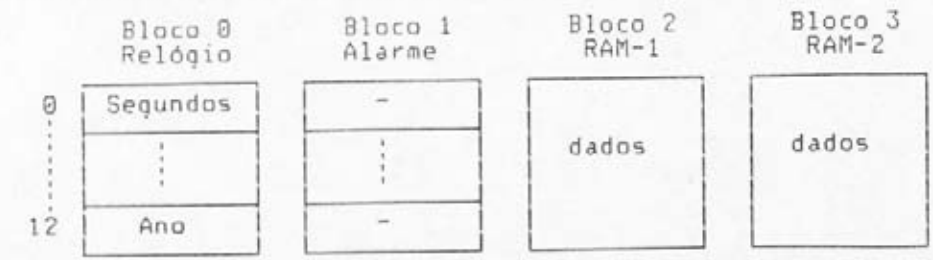
Existem 26 registradores de 4 bits no relógio para uso geral. O MSX armazena os seguintes dados nesta memória:

- Ajuste de tela (Set Adjust);
- Valores iniciais de SCREEN, WIDTH e COLOR;
- Volume e tom do beep;
- Cor da tela inicial;
- Código do país;
- Senha, prompt do BASIC ou título da tela inicial.

2 - ESTRUTURA E REGISTRADORES DO CLOCK-IC

O CLOCK-IC possui quatro blocos de memória sendo que cada um consiste em 13 registradores de 4 bits cada, endereçados de 0 a 12. Possui também mais três registradores de 4 bits, para a seleção de blocos e controle das funções, sendo acessados pelos endereços 13 a 15.

Os registradores dos blocos (#0 a #12) e o registrador de modo (#13) podem ser lidos ou escritos. Os registradores de teste (#14) e de reset (#15) só podem ser escritos.



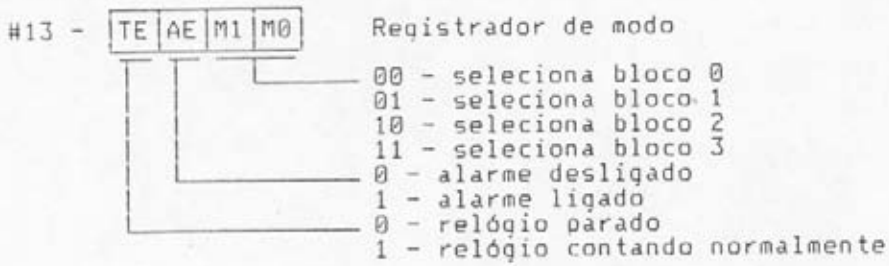
O REGISTRADOR DE MODO

O registrador de modo tem três funções.

A primeira é a seleção de blocos. Os quatro blocos de 13 registradores de 4 bits endereçados de #0 a #12 são selecionados pelos dois bits mais baixos do registrador de modo. Os registradores de endereçamento #13 a #15 são acessados independentemente do bloco selecionado.

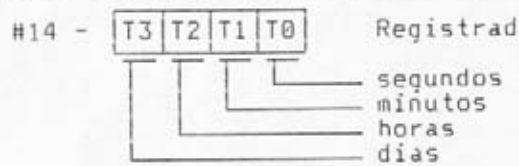
A segunda função é ligar ou desligar a saída de alarme. O bit 2 do registrador de modo é usado para isto. Porém, o MSX2 standard não suporta a função de alarme, sendo que a alteração deste bit não causa efeito algum.

A terceira função é a parada do relógio. Escrevendo 0 no bit 3 do registrador de modo, a contagem em segundos é interrompida e a função de relógio terminada. Setando o bit 3 em 1, a contagem é retomada.



O REGISTRADOR DE TESTE

O registrador de teste (#14) é usado para incrementar rapidamente e confirmar a data e hora do relógio. Setando em 1 cada bit deste registrador, pulsos de 16384 Hz são diretamente inseridos nos registradores de dia, hora, minuto e segundo.



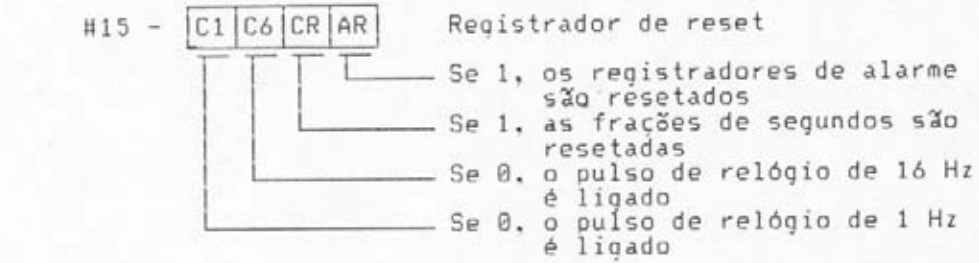
O REGISTRADOR DE RESET

O registrador de reset tem as seguintes funções:

Para resetar o alarme, basta setar o bit 0 em 1: todos os registradores de alarme serão resetados em 0.

O bit 1, quando setado em 1, causa o reset das frações do contador de segundos. Esta função é útil para acertar os segundos corretamente.

Setando o bit 2 em 0, o pulso do relógio de 16 Hz é ativado e setando o bit 3 em 0, é setado o pulso de 1 Hz para o relógio.



2.1 - ACERTANDO O RELÓGIO E O ALARME

O bloco 0 de memória é usado para o relógio. Selecionado o bloco 0 pelo registrador de modo e escrevendo os dados nos registradores corretos, pode-se acertar a data e a hora.

Já o bloco 1 é usado para o alarme. Observe que no alarme só podemos definir os dias, horas e minutos.

No relógio, o ano é representado por 2 dígitos apenas (registradores #11 e #12). Para obter o ano correto, deve-se somar 80 ou 1980 a este valor. Por exemplo, setando os registros #11 e #12 em 0, o ano correto será 1980.

O dia da semana é representado pelos dígitos 0 a 6, no registrador #6.

Bloco 0 - Relógio

	Bits --->	3	2	1	0
0	Seg. 1º dígito	x	x	x	x
1	Seg. 2º dígito	•	x	x	x
2	Min. 1º dígito	x	x	x	x
3	Min. 2º dígito	•	x	x	x
4	Hor. 1º dígito	x	x	x	x
5	Hor. 2º dígito	•	•	x	x
6	Dia da semana	•	x	x	x
7	Dia 1º dígito	x	x	x	x
8	Dia 2º dígito	•	•	x	x
9	Mês 1º dígito	x	x	x	x
10	Mês 2º dígito	•	•	•	x
11	Ano 1º dígito	x	x	x	x
12	Ano 2º dígito	x	x	x	x

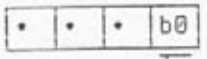
Bloco 1 - Alarme

	Bits --->	3	2	1	0
0	- - -	•	•	•	•
1	- - -	•	•	•	•
2	Min. 1º dígito	x	x	x	x
3	Min. 2º dígito	•	x	x	x
4	Hor. 1º dígito	x	x	x	x
5	Hor. 2º dígito	•	•	x	x
6	Dia da semana	•	x	x	x
7	Dia 1º dígito	x	x	x	x
8	Dia 2º dígito	•	•	x	x
9	- - -	•	•	•	•
10	12/24 horas	•	•	•	x
11	Ano bissexto	•	•	x	x
12	- - -	•	•	•	•

Obs.: Os bits indicados com "•" devem ser sempre 0 e não podem ser modificados.

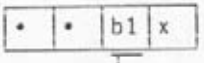
Dois modos podem ser selecionados para a contagem de horas: 12 horas ou 24 horas. Por exemplo, no modo 24 horas, quando for 1 hora da tarde, o relógio indicará 13:00 horas e no modo 12 horas indicará 1:00 pm. O registrador #10 do bloco 1 é usado

para esta seleção.



Registrador #10 / Bloco 1

0 - 12 horas
1 - 24 horas

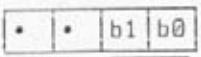


Registrador #5 / Bloco 0

0 - Antes do meio-dia (a.m.)
1 - Depois do meio-dia (p.m.)

A flag am/pm no registrador #5 do bloco 0 só pode ser usada no caso de seleção de 12 horas pelo registrador #10 do bloco 1.

O registrador #11 do bloco 1 é um contador de 4 (0 a 3) incrementado a cada ano. Quando os dois bits baixos deste registrador forem 00, o ano será considerado bissexto e serão contados 29 dias para o mês de Fevereiro. A referência para este contador é o ano de 1980, que foi bissexto.



Registrador #11 / Bloco 1

Bits 00 representam ano bissexto

2.2 - CONTEÚDO DA MEMÓRIA MANTIDA A BATERIA

Os blocos 2 e 3 de RAM do CLOCK-IC não têm função para o relógio. No MSX, eles são usados para armazenar alguns dados que o micro reconhece quando ligado e executa automaticamente algumas rotinas baseadas nesses dados. Cada bloco oferece 13 posições de 4 bits cada.

Conteúdo do Bloco 2

	bit 3	bit 2	bit 1	bit 0
0		ID		
1		Adjust X (-8 a +7)		
2		Adjust Y (-8 a +7)		
3	•	•	Modo entrelç.	Modo Screen
4	Largura inicial da tela (WIDTH) - low			
5	Largura inicial da tela (WIDTH) - high			
6	Código da cor de fundo inicial			
7	Código da cor de frente inicial			
8	Código da cor da borda inicial			
9	•	Tipo impres.	Click teclas	Teclas func.
10	Tom do beep		Volume do beep	
11	•	•	Cor da tela inicial	
12		Código Nativo		

O bloco 3 pode ter três funções diferentes, dependendo do conteúdo da posição ID (registrador #0 do bloco 3). Se o ID for igual a 0, o micro apresentará um título de até 6 caracteres na tela inicial. Se for igual a 1, o bloco 3 armazenará uma senha (password) de até 6 caracteres que deve ser digitada ao ligar o micro para que este possa ser acessado. Se ID for igual a 2, será armazenado um novo prompt para o BASIC, no lugar do "Ok".

ID = 0 -> apresenta um titulo na tela inicial

0	0
1	1º caractere - low
2	1º caractere - high
	⋮
11	6º caractere - low
12	6º caractere - high

ID = 1 -> armazena a senha (password)

0	1
1	Uso ID = 1
2	Uso ID = 2
3	Uso ID = 3
4	Senha ┐
5	Senha ├ A senha é armazenada
6	Senha ├ compactada em 4 x 4 bits
7	Senha ┘
8	Key Cartridge Flag
9	
10	Key Cartridge
11	Value
12	

ID = 2 -> armazena novo prompt do BASIC

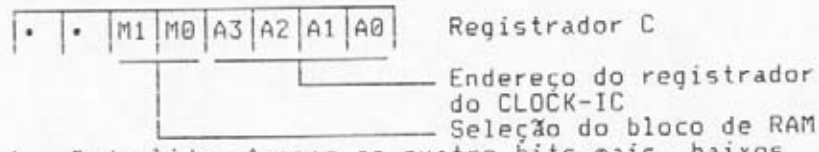
0	2
1	1º caractere - low
2	1º caractere - high
	⋮
11	6º caractere - low
12	6º caractere - high

3 - ACESSO AO CLOCK-IC

O acesso ao relógio e à memória mantida a bateria é feito através de duas rotinas do BIOS, na SUBROM, sendo necessário usar a chamada inter-slot para acessá-las.

A descrição completa dessas duas rotinas do BIOS pode ser vista na página seguinte.

RDCLK (01F5H / SUBROM)
Função: Ler um registro do CLOCK-IC.
Entrada: C - endereço do CLOCK-IC.



Saída: A - Dado lido. Apenas os quatro bits mais baixos são válidos.

WRTCLK (01F9H / SUBROM)
Função: Escrever um dado em um registrador do CLOCK-IC.
Entrada: C - Endereço do CLOCK-IC. Ver a rotina RDCLK.
A - Dado a ser escrito. Apenas os quatro bits mais baixos serão escritos.
Saída: Nenhuma.

Capítulo 8

O MSX TURBO R

Nos novos modelos MSX, foi introduzida uma nova CPU de 16 bits totalmente compatível com o Z80 a nível de software. A CPU R800 é contruida em um chip LSI com encapsulamento DIL de 100 terminais. O clock do R800 , nos primeiros modelos MSX turbo R, é de 28,64 MHz, mas pode chegar a 40 MHz, e de quebra ainda tem saida para multitarefa, infelizmente desligada nos primeiros modelos MSX turbo R.

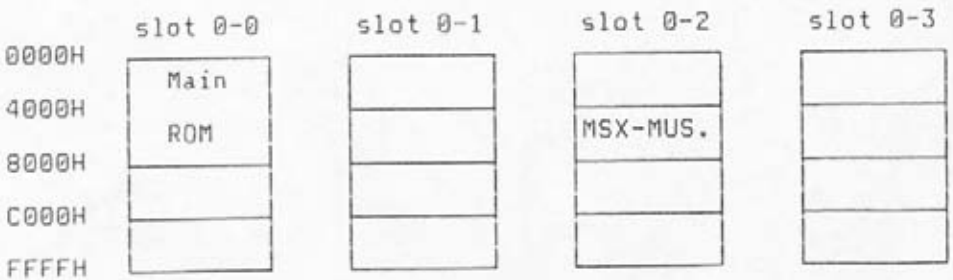
O set de instruções do R800 engloba todas as instruções do Z80 e acrescenta mais algumas, como multiplicação direta de 8 e 16 bits e tratamento dos registradores de índice IX e IY como se fossem dois registradores de 8 bits cada, denominados por .ixl, .ixh, .iyl e .iyh.

Como o R800 é totalmente compatível com o Z80 a nível de instruções, é possivel fazer um programa que funcione no MSX2, digamos um CAD, e colocar uma pequena rotina que detecta se o programa está residente em um MSX turbo R, e nesse caso ativar o R800 para acelerar em até 10 vezes a execução do programa. Alguns cuidados, entretanto, devem ser tomados. No caso de acesso direto por portas de I/O, mesmo aos componentes internos como o VDP ou o OPLL, o modo turbo deve ser desligado, pois haverá dessincronização devido à maior velocidade do R800. O único acesso direto que pode ser feito sem problemas é o da Memória Mapeada (portas de I/O FCH, FDH, FEH e FFH). Qualquer outro acesso direto deve ser feito através do Z80. Já no caso de acesso pelo BIOS, BDOS ou BASIC, não há problemas, pois o BIOS compensa as diferenças de "timing" quando necessário.

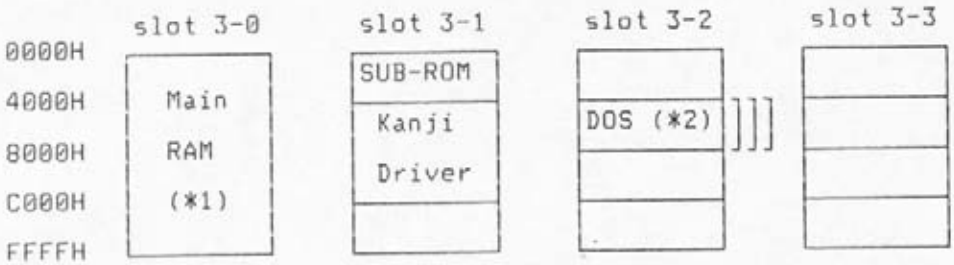
1 - ORGANIZAÇÃO DOS SLOTS

No MSX turbo R, a organização de slots foi padronizada, por causa da RAM, que é conectada diretamente ao R800, o que também facilita o desenvolvimento de software específico. Os slots primários 0 e 3 são reservados para o sistema, e os slots 1 e 2 são slots externos para o usuário. Os slots 0 e 3 são expandidos, e sua organização é a seguinte:

SLOT PRIMARIO 0 EXPANDIDO (interno)



SLOT PRIMÁRIO 3 EXPANDIDO (interno)



(*1) O slot 3-0 deve conter no mínimo 256 Kbytes de RAM mapeada.
(*2) O DOS Kernel ocupa 4 segmentos de 16 Kbytes que são trocados exclusivamente na página 1. Os primeiros três segmentos são para o MSXDOS2 e o último para o MSXDOS1.

2 - WAIT STATES

Na operação no modo R800, geralmente não são gerados wait states, mas em algumas condições eles são gerados.

Quando um slot externo é acessado, são gerados 3 wait states. Isto é necessário para que todo hardware desenvolvido até o momento funcione corretamente, já que a alta velocidade no modo R800 poderia inviabilizar tais periféricos.

Quando a ROM interna é acessada, são gerados 2 wait states, devido à relativa lentidão dos chips de ROM.

Quando a DRAM interna é acessada, é gerado 1 wait state. Por isso, o acesso é bem mais rápido na DRAM que na ROM.

3 - MODOS DE OPERAÇÃO

O MSX turbo R tem duas CPU's: o tradicional Z80 e o novo R800. A manutenção do Z80 foi necessária para que o MSX turbo R fosse compatível com seus antecessores.

As duas CPU's podem ser trocadas livremente durante o processamento, mas não podem ser ativadas simultaneamente. Duas combinações específicas entre o DOS e as CPU's são recomendadas: Z80/DOS1 ou R800/DOS2, mas nada impede que o DOS1 possa funcionar sob o R800. Quando o sistema inicializa, verifica o boot do disco para entrar no modo correto. Se não houver disco, o sistema entrará automaticamente no modo R800 DRAM, a menos que a tecla "1" seja pressionada durante o reset, o que força o sistema a entrar no modo Z80.

Uma observação importante é que há dois modos de operação do R800, o ROM e o DRAM. No acesso com ROM, toda a memória mapeada de 256 Kbytes fica livre. Já no modo DRAM, o sistema transfere para as quatro últimas páginas da memória mapeada o conteúdo da Main ROM (32K), da SUB-ROM (16K) e da primeira parte do Kanji Driver. A vantagem disso é que as rotinas do BIOS passam a ser processadas com muito mais rapidez, já que a ROM é bem mais lenta em comparação à DRAM. Em vista disso, há uma perda de 64 Kbytes de RAM disponível. Entretanto, se o programa do usuário fizer muitos acessos ao BIOS, a perda de memória em troca do ga-

nho de velocidade pode ser vantajosa. É aí que o programador deve optar pelo modo mais vantajoso, o modo R800 ROM ou R800 DRAM. Caso opte pelo modo DRAM, terá à sua disposição 192 Kbytes de memória e se optar pelo modo ROM, terá todos os 256 Kbytes disponíveis. Os 64 Kbytes do modo DRAM sempre ficam nas páginas mais altas da memória mapeada e não podem ser escritos.

Um modelo de rotina que pode ser incluida nos programas para que estes utilizem a velocidade do R800 é a seguinte:

```
RDSLT:  EQU 0000CH
CALSLT: EQU 0001CH
CHGCPU: EQU 00180H
SLTROM: EQU 0FCC1H
;
;--- VERIFICA VERSÃO ---
;
LD    A,(SLTROM)
LD    HL,0002DH
CALL  RDSLT
SBC   A,2
JR    C,NAOTUR
;
;--- PREPARA TROCA DE MODO ---
;--- ESCOLHA UMA DAS TRÊS OPÇÕES ---
;
; MODO Z80:
LD    A,11001110B
AND   002H
XOR   082H
;
; MODO R800 ROM
LD    A,01000100B
AND   002H
XOR   081H
;
; MODO R800 DRAM
LD    A,11001101B
AND   002H
XOR   082H
;
;--- TROCA DE MODO ---
;
LD    IY,(SLTROM-1)
LD    IX,CHGCPU
CALL  CALSLT
;
NAOTUR:
;
END
```

Observe que a rotina faz um teste para verificar se está rodando em um MSX turbo R ou não. Se não estiver, pula para a label NAOTUR (termina) e se estiver executa a rotina CHGCPU do BIOS, que troca os processadores de acordo com o valor passado no registrador A. Esta rotina funciona tanto sob o DOS quanto sob o BASIC, em qualquer endereço.

A tabela da página seguinte demonstra o ganho de velocidade relativo e absoluto quando se usa o R800 no lugar do Z80.

INSTRUÇÕES		Z80 (µS)	R800(µS)	GANHO ABSOLUTO	GANHO RELATIVO
LD	r,s	1.40	0.14	x 10.0	900 %
LD	r,(HL)	2.23	0.42	x 5.3	430 %
LD	r,(IX+n)	5.87	0.70	x 8.4	740 %
PUSH	qq	3.35	0.56	x 6.0	500 %
LDIR	(BC<>0)	6.43	0.98	x 6.6	560 %
ADD	A,r	1.40	0.14	x 10.0	900 %
INC	r	1.40	0.14	x 10.0	900 %
ADD	HL,ss	3.35	0.14	x 24.0	2300 %
INC	ss	1.96	0.14	x 14.0	1300 %
JP		3.07	0.42	x 7.3	630 %
JR		3.63	0.42	x 8.7	770 %
DJNZ	(B<>0)	3.91	0.42	x 9.3	830 %
CALL		5.03	0.84	x 6.0	500 %
RET		3.07	0.56	x 5.5	450 %
MULTU	A,r	160	1.96	x 81.6	8060 %
MULTUW	HL,rr	361	5.03	x 71.7	7070 %

O ganho de velocidade em relação ao Z80 é muito grande, atingindo uma média de 7 vezes ou 600 %. Observe que as instruções MULTU e MULTUW são exclusivas do R800, não existindo no Z80. Para obtenção do tempo em microssegundos, foram usadas rotinas otimizadas para o Z80. A instrução MULTU efetua multiplicação de operandos de 8 bits e a MULTUW multiplica operandos de 16 bits.

3.1 - INSTRUÇÕES ESPECÍFICAS DO R800

As instruções que foram acrescentadas para o R800 e que não existem no Z80 são as seguintes:

Memônico	Ilustração	Flags S Z H P N C	Binário 7 6 5 4 3 2 1 0	Hex	M
ld u,u'	u <- u'	• • • • • •	1 1 0 1 1 1 0 1 0 1 u u'	DDH	2
ld v,v'	v <- v'	• • • • • •	1 1 1 1 1 1 0 1 0 1 v v'	FDH	2
ld u,n	u <- n	• • • • • •	1 1 0 1 1 1 0 1 0 0 u 1 1 0 ------ n ------	DDH	3
ld v,n	v <- n	• • • • • •	1 1 1 1 1 1 0 1 0 0 v 1 1 0 ------ n ------	FDH	3
add .a,p	.a <- .a+p	! ! ! V 0 !	1 1 0 1 1 1 0 1 1 0 0 0 0 p	DDH	2
add .a,q	.a <- .a+q	! ! ! V 0 !	1 1 1 1 1 1 0 1 1 0 0 0 0 q	FDH	2
addc .a,p	.a <- .a+p+C	! ! ! V 0 !	1 1 0 1 1 1 0 1 1 0 0 0 1 p	DDH	2
addc .a,q	.a <- .a+q+C	! ! ! V 0 !	1 1 1 1 1 1 0 1 1 0 0 0 1 q	FDH	2

sub .a,p	.a <- .a-p	; ; ; V 1 ;	1 1 0 1 1 1 0 1 1 0 0 1 0 p	DDH	2
sub .a,q	.a <- .a-q	; ; ; V 1 ;	1 1 1 1 1 1 0 1 1 0 0 1 0 q	FDH	2
subc .a,p	.a <- .a-p-C	; ; ; V 1 ;	1 1 0 1 1 1 0 1 1 0 0 1 1 p	DDH	2
subc .a,q	.a <- .a-q-C	; ; ; V 1 ;	1 1 1 1 1 1 0 1 1 0 0 1 1 q	FDH	2
dec p	p <- p-1	; ; ; V 1 •	1 1 0 1 1 1 0 1 0 0 p 1 0 1	DDH	2
dec q	q <- q-1	; ; ; V 1 •	1 1 1 1 1 1 0 1 0 0 q 1 0 1	FDH	2
and .a,p	.a <- .a p	; ; 1 P 0 0	1 1 0 1 1 1 0 1 1 0 1 0 0 p	DDH	2
and .a,q	.a <- .a q	; ; 1 P 0 0	1 1 1 1 1 1 0 1 1 0 1 0 0 q	FDH	2
or .a,p	.a <- .aVp	; ; 0 P 0 0	1 1 0 1 1 1 0 1 1 0 1 1 0 p	DDH	2
or .a,q	.a <- .aVq	; ; 0 P 0 0	1 1 1 1 1 1 0 1 1 0 1 1 0 q	FDH	2
xor .a,p	.a <- .aVp	; ; 0 P 0 0	1 1 0 1 1 1 0 1 1 0 1 0 1 p	DDH	2
xor .a,q	.a <- .aVq	; ; 0 P 0 0	1 1 1 1 1 1 0 1 1 0 1 0 1 q	FDH	2
cmp .a,p	.a - p	; ; ; V 1 ;	1 1 0 1 1 1 0 1 1 0 1 1 1 p	DDH	2
cmp .a,q	.a - q	; ; ; V 1 ;	1 1 1 1 1 1 0 1 1 0 1 1 1 q	FDH	2
mulub .a,r	.hl <- .a*r	0 ; • 0 • ;	1 1 1 0 1 1 0 1 1 1 r 0 0 1	EDH	14
muluw .hl,ss	de:hl<-hl*ss	0 ; • 0 • ;	1 1 1 0 1 1 0 1 1 1 ss 0 0 1 1	EDH	36
in .f,(c)	(.c)	; ; 0 P 0 •	1 1 1 0 1 1 0 1 0 1 1 1 0 0 0 0	EDH 70H	3

Convenção dos registradores:

	000	001	010	011	100	101	110	111
u	.b	.c	.d	.e	ixh	ixl	•	.a
v	.b	.c	.d	.e	iyh	iyl	•	.a
p	•	•	•	•	ixh	ixl	•	•
q	•	•	•	•	iyh	iyl	•	•

	000	001	010	011	00	11
r	.b	.c	.d	.e	•	•
ss	•	•	•	•	bc	sp

4 - A MSX-MIDI

A partir do segundo modelo MSX turbo R, a MSX-MIDI foi padronizada. MIDI quer dizer "Musical Instruments Digital Interface", ou seja, interface digital para instrumentos musicais. Com ela é possível controlar instrumentos musicais que tenham entrada MIDI.

A MSX-MIDI é controlada diretamente por portas de I/O. As portas reservadas são E8H a EFH quando a MIDI for interna e mais três se a MIDI for externa: E0H a E2H

A função de cada porta é a seguinte:

```
E0H - Transmissão/recepção de dados (interface externa)
E1H - Porta de controle (interface externa)
E2H - Porta de seleção
E8H - Transmissão/recepção de dados
E9H - Porta de controle
EAH - Latch dos sinais (escrita somente)
EBH - Imagem de EAH
ECH - Contador 0
EDH - Contador 1
EEH - Contador 2
EFH - Controle dos contadores (escrita somente)
```

4.1 - DESCRIÇÃO DAS PORTAS DA MIDI

• MIDI EXTERNA

E0H - | b7 | b6 | b5 | b4 | b3 | b2 | b1 | b0 | — Porta de dados

b0-b7 - dados
escrita -> transm. (TXD)
leitura -> recebe (RXD)

E1H - | b7 | b6 | b5 | b4 | b3 | b2 | b1 | b0 | — Porta de controle

Leitura:

```
b7 - DSR   8253 data set ready (1=pronto)
b6 - BRK   8251 parada detectada (1=detectada)
b5 - FE    8251 flag de erro de frame (1=erro)
b4 - OE    8251 flag de erro de overrun (1=erro)
b3 - PE    8251 flag de erro de paridade (1=erro)
b2 - EMPTY 8251 buffer de transmissão vazio (1=vazio)
b1 - RRDY  8251 status de recepção (1=dado presente)
b0 - TRDY  8251 status de transmissão (1=pronto)
```

E2H - | b7 | • | • | • | • | • | • | b0 |

Escrita:

```
b7 - EN - habilitação da MIDI externa (0=habilita)
          [na inicialização, b7 é setado em "1"]
b0 - E8 - seleção de endereço da interface MIDI
          (0=E8H/E9H; 1=E0H/E1H)
```

• MIDI INTERNA

E8H - | b7 | b6 | b5 | b4 | b3 | b2 | b1 | b0 | Porta de dados

* Organização idêntica à porta E0H.

E9H - | b7 | b6 | b5 | b4 | b3 | b2 | b1 | b0 | Porta de controle

* Organização idêntica à porta E1H para leitura.

Escrita:
Modo: b7=S2; b6=S1; b5=EP; b4=PEN;
b3=L2; b2=L1; b1=B2; b0=B1.
Comando:
b7 = EH - normalmente "0";
b6 = IR - normalmente "0";
b5 = RIE - habilita transmissão MIDI IN (1=habilita);
b4 = ER - reseta erro (1=reseta flags de erro;
2=sem operação);
b3 = SBRK - normalmente "0";
b2 = PE - habilita recepção MIDI IN (1=habilita);
b1 = TIE - timer 8253 (contador #2) - habilita transmissão (1=habilita);
b0 = TEN - Habilita transmissão MIDI OUT (1=habilita).

Obs: Quando um dado for escrito no modo comando, é necessário uma espera de 16 ciclos T (3.58 MHz) para o resultado. Quando for escrita uma seqüência de comandos na porta de comando, é necessária a espera antes de escrever os dados.

EAH - | b7 | b6 | b5 | b4 | b3 | b2 | b1 | b0 | - dados 8253

Escrita:
8253 OUT2 - latch dos sinais do terminal

Leitura: sem efeito

EBH - | b7 | b6 | b5 | b4 | b3 | b2 | b1 | b0 |

Esta porta é uma imagem de EAH.

ECH (R/W) - | b7 | b6 | b5 | b4 | b3 | b2 | b1 | b0 |

Leitura/escrita: contador 0

EDH (R/W) - | b7 | b6 | b5 | b4 | b3 | b2 | b1 | b0 |

Leitura/escrita: contador 1

EEH (R/W) - | b7 | b6 | b5 | b4 | b3 | b2 | b1 | b0 |

Leitura/escrita: contador 2

EFH - | b7 | b6 | b5 | b4 | b3 | b2 | b1 | b0 |

Leitura: sem efeito

Escrita:
b7-b6 - (SC1,SC0) seleciona contador:
b5-b4 - (RW1,RW0) modo leitura/escrita do contador;
b3-b1 - (M2,M1,M0) modo do contador:
b0 - seleciona contador binário / contador BCD.

4.1 - CHECAGEM DA PRESENÇA DA MSX-MIDI

A MSX-MIDI pode já vir internamente ao MSX turbo R como também pode ser implementada através de cartucho, mas somente para o MSX turbo R em diante (o byte 002DH deve ser maior que 2). Se a MSX-MIDI for interna, o bit 0 do endereço 002EH da ROM estará setado em "1".

A diferença entre a MIDI interna ou externa pode ser obtida no endereço 4018H, conforme mostrado abaixo:

Endereço	Interna	Externa
4018H	41H(A)	??H(?)
4019H	50H(P)	??H(?)
401AH	52H(R)	??H(?)
401BH	4CH(L)	??H(?)
401CH	4FH(O)	4DH(M)
401DH	50H(P)	49H(I)
401EH	4CH(L)	44H(D)
401FH	4CH(L)	49H(I)

A interface MIDI também altera alguns hooks, e estes são diferentes conforme a MIDI seja interna ou externa.

Se a MIDI for interna, os hooks redirecionados serão:

Endereço	Novo nome	Nome antigo	Função
FF75H	HMDIN	HOKNO	MIDI IN
FF93H	HMDTM	HFRQI	timer do 8253

No caso de MIDI externa, os hooks acima nao podem ser usados; neste caso pode ser utilizado o hook HKEYI (FD9AH).

APÊNDICE

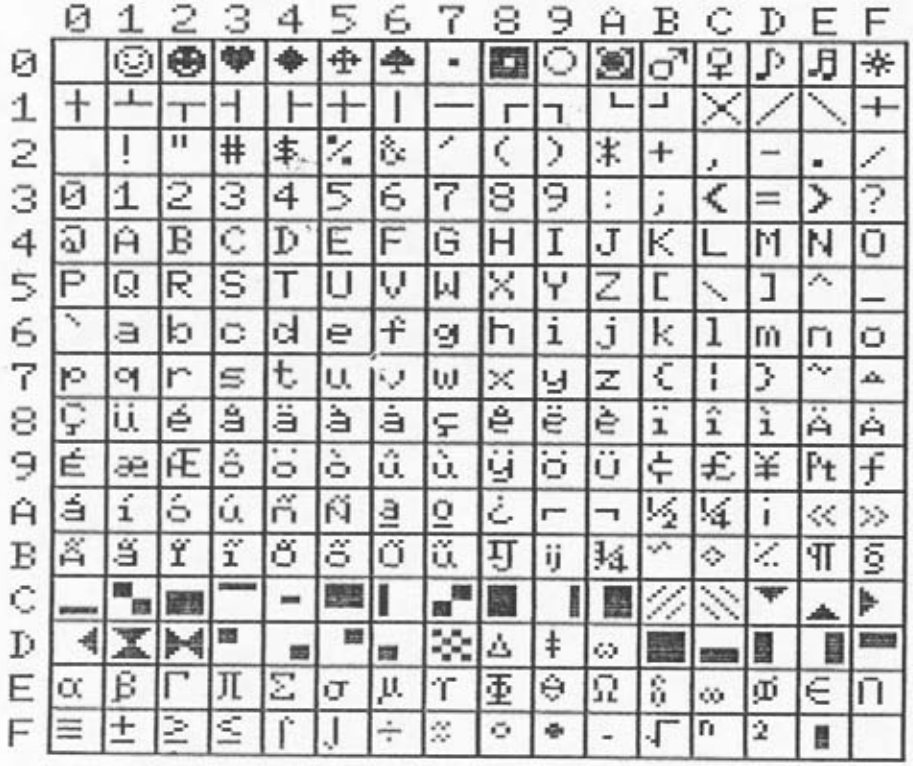
1 - TABELAS DE CARACTERES

1.1 - TABELA DE CARACTERES JAPONESA

A tabela abaixo é a que vem nos micros japoneses.

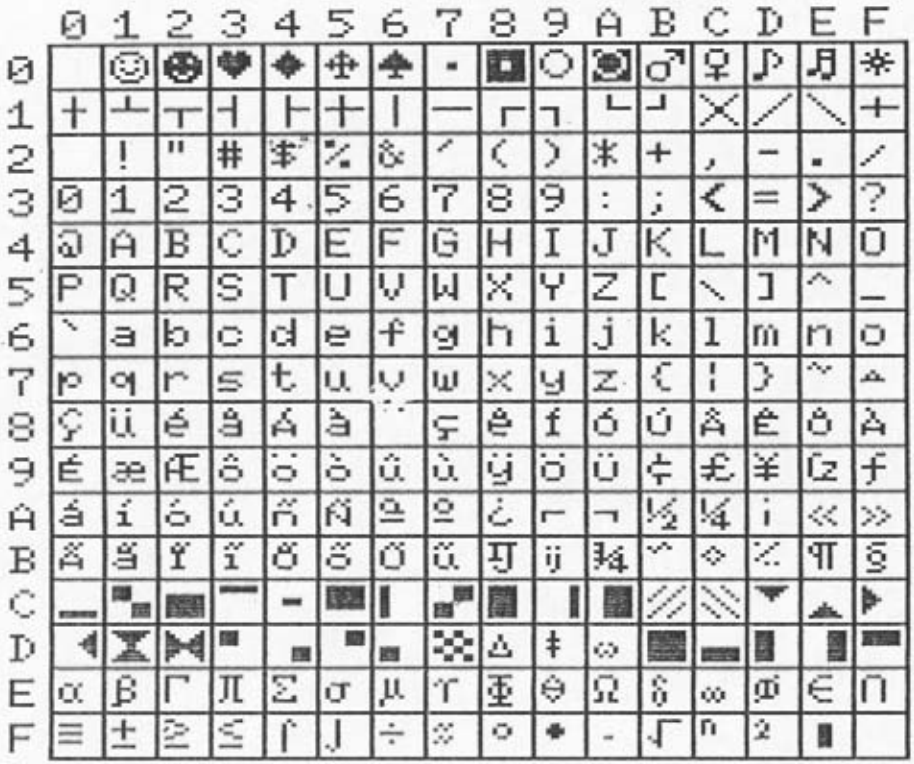
1.2 - TABELA DE CARACTERES INTERNACIONAL

A tabela de caracteres internacional é a adotada por todos os países da Europa (Reino Unido, França, Alemanha, etc.) e também pelos Estados Unidos.



1.3 - TABELA DE CARACTERES BRASILEIRA

No Brasil, optou-se por uma tabela de caracteres ligeiramente diferenciada da Internacional. Isto foi necessário para adaptar a tabela de caracteres à língua portuguesa.



2 - CÓDIGOS DE CONTROLE

TECLADO	DEC	HEX	FUNÇÃO
Ctrl+A	001	01H	Determina caractere gráfico
Ctrl+B	002	02H	Desvia cursor para início da palavra anterior
Ctrl+C	003	03H	Encerra a condição de entrada
Ctrl+D	004	04H	
Ctrl+E	005	05H	Cancela caracteres do cursor até fim da linha
Ctrl+F	006	06H	Desvia cursor ao início da palavra seguinte
Ctrl+G	007	07H	Aciona o beep
Ctrl+H	008	08H	Apaga letra anterior ao cursor (BS)
Ctrl+I	009	09H	Move cursor p/ pos. tabulação seguinte (TAB)
Ctrl+J	010	0AH	Muda de linha (Linefeed)
Ctrl+K	011	0BH	Volta cursor para a posição 1,1 (HOME)
Ctrl+L	012	0CH	Limpa a tela e volta cursor para a posição 1,1
Ctrl+M	013	0DH	Retorno do Carro (RETURN)
Ctrl+N	014	0EH	Move o cursor para o fim da linha
Ctrl+O	015	0FH	
Ctrl+P	016	10H	
Ctrl+Q	017	11H	
Ctrl+R	018	12H	Liga/desliga modo de inserção (INS)
Ctrl+S	019	13H	
Ctrl+T	020	14H	
Ctrl+U	021	15H	Apaga toda a linha na qual está o cursor
Ctrl+V	022	16H	
Ctrl+W	023	17H	
Ctrl+X	024	18H	(SELECT)
Ctrl+Y	025	19H	
Ctrl+Z	026	1AH	
Ctrl+[	027	1BH	(ESC)
Ctrl+\	028	1CH	Move o cursor para a direita
Ctrl+]	029	1DH	Move o cursor para a esquerda
Ctrl+^	030	1EH	Move o cursor para cima
Ctrl+_	031	1FH	Move o cursor para baixo
Delete	127	7FH	Apaga a letra que está sob o cursor (DEL)

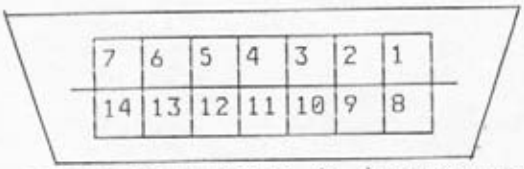
3 - MAPA DAS PORTAS DE I/O DO Z80

```
00H a 2FH  -  Liberadas para uso pelo programador
30h a 38H  -  Interface SCSI
39H a 5FH  -  Reservadas - não utilizar
60H a 6FH  -  Portas para o VDP V9990
70H a 7BH  -  Reservadas - não utilizar
7CH a 7DH  -  FM OPLL (YM2413)
7EH a 7FH  -  Reservadas - não utilizar
80H a 87H  -  Interface Serial RS232C (8251/8253)
88H a 8BH  -  Portas do VDP para adaptação com o MSX1
8CH a 8DH  -  Para o Modem
8EH a 8FH  -  Megaram (8EH) e Megaram Disk (8FH)
90H a 91H  -  Portas para impressora
92H a 97H  -  Reservadas - não utilizar
98H a 9BH  -  Portas para os VDPs V9938 e V9958
9CH a 9FH  -  Reservadas - não utilizar
A0H a A3H  -  PSG (AY-3-8910)
A4H a A5H  -  PCM - porta de comando (A5H) e de dados (A4H)
A6H        -  Reservada - não utilizar
A7H        -  Controle da tecla PAUSE e indicador modo Turbo
A8H a ABH  -  Porta paralela (8255)
ACH a AFH  -  MSX-Engine (chip MSX I/O)
B0H a B3H  -  Expansão de memória (especificação SONY 8255)
B4H a B5H  -  IC do relógio (RP-5C01)
B6H a B7H  -  Reservadas - não utilizar
B8H a BBH  -  Controle de lightpen (especificação SANYO)
BCH a BFH  -  Controle de VHD (especificação JVC 8255)
C0H a C1H  -  MSX-AUDIO
C2H a C7H  -  Reservadas - não utilizar
C8H a CFH  -  MSX-Interface
D0H a D7H  -  Interface para Disk Drive e HD
D8H a D9H  -  Kanji-ROM (especificação TOSHIBA)
DAH a DBH  -  Para futuras expansões dos Kanjis
DCH a DFH  -  Reservadas - não utilizar
E0H a E1H  -  Acesso à MSX-MIDI (externa)
E2H a E3H  -  Reservadas - não utilizar
E4H a E5H  -  Controle de troca do processador (Z80 <-> R800)
E6H a E7H  -  Relógio do sistema para o turbo R
E8H a EBH  -  Acesso à MSX-MIDI (8251)
ECH a EFH  -  Acesso à MSX-MIDI (8253)
F0H a F3H  -  Reservadas - não utilizar
F4H        -  Estado do RESET para o turbo R
F5H        -  Controle do sistema (setanto bit em 1 habilita):
              b0 - Kanji-ROM          b4 - MSX-Interface
              b1 - Reservado Kanji    b5 - Serial RS232C
              b2 - MSX-AUDIO          b6 - Lightpen
              b3 - Superimpose        b7 - IC do relógio
F6H        -  Barramento I/O de cores
F7H        -  Controle AV (setando o bit em 1 habilita):
              b0 - Audio R (direito)
              b1 - Audio L (esquerdo)
              b2 - Seleciona entrada de video
              b3 - Detecta entrada de video
              b4 - Controle AV
              b5 - Controle Ym
              b6 - Inverso b4 (VDP registrador R#9 escrita)
              b7 - Inverso b5 (VDP registrador R#9 leitura)
F8H a FBH  -  Reservadas - não utilizar
FCH a FFH  -  Memória Mapeada
```

4 - INTERFACE DE IMPRESSORA

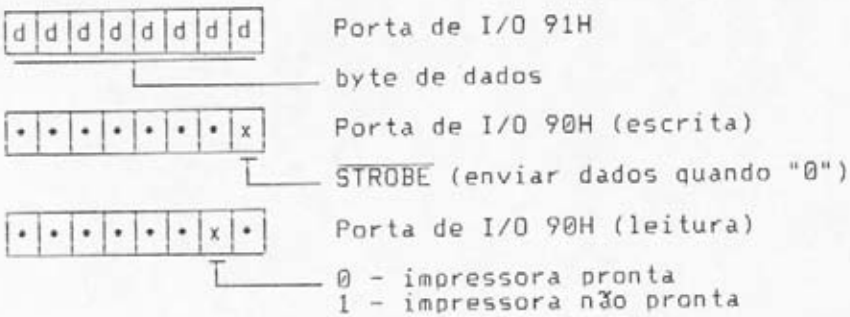
Esta seção descreve como acessar a impressora através de programas em Assembly. A interface de impressora é suportada pelo BIOS, pelo BASIC e pelo DOS. O MSX usa duas portas paralelas de 8 bits para acesso à impressora. O padrão adotado é o Centronics. O conector standard também é definido (Amphenol 14 contatos com conector fêmea no micro).

Veja a configuração abaixo:



conector standard de impressora

1 - $\overline{\text{STROBE}}$
2 a 9 - dados (b0 a b7)
11 - BUSY
14 - GND



Os dados enviados para a impressora dependem se esta foi especialmente desenvolvida para o padrão MSX ou não.

Numa impressora MSX standard podem ser impressos todos os caracteres que saem no vídeo. Os caracteres gráficos especiais de código 01H a 1FH também podem ser impressos enviando o cabeçalho gráfico 01H seguido do código do caractere + 40H.

A mudança de linha numa impressora MSX standard é feita enviando-se os caracteres de controle 0DH e 0AH. Para imprimir uma imagem em bits, envie "nnnn" significando quatro códigos decimais, depois da seqüência ESC + "Snnnn".

O MSX tem uma função para transformar o código TAB (09H) para o número adequado de espaços em impressoras que não tem a função TAB. Isto é feito através de uma flag na área de variáveis de sistema:

RAWPRT (F418H,1) - Substitui TAB por espaços quando o conteúdo for 0; caso contrário, não substitui.

4.1 - CÓDIGOS DE CONTROLE PARA A IMPRESSORA

```
0AH          - Avanço de linha
0CH          - Avanço de formulário
0DH          - Retorno do carro
ESC+"A"      - Espaçamento entre linhas normal
ESC+"B"      - Sem espaçamento entre linhas
ESC+"Snnnn"  - Impressão de imagens de bits
```

Existem muitos outros caracteres de controle. Via de regra, quanto mais sofisticada a impressora, mais caracteres de controle haverão. Entretanto, tais caracteres variam de impressora para impressora, sendo conveniente consultar o manual respectivo.

O acesso à impressora deve ser feito preferencialmente através das rotinas do BIOS, caso contrário podem haver problemas de incompatibilidade. As rotinas do BIOS dedicadas à impressora são as seguintes:

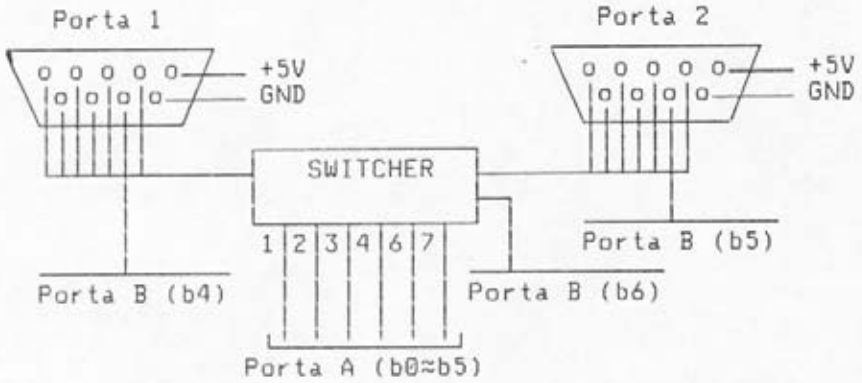
```
LPTOUT (00A5H/Main) - Envia um caractere para a impressora
LPTSTT (00A8H/Main) - Obtém o status da impressora
OUTDLP (014DH/Main) - Envia um caractere para a impressora,
                      com algumas diferenças da LPTOUT
```

Para saber como usar as rotinas citadas, leia a seção "BIOS em ROM" no capítulo 2.

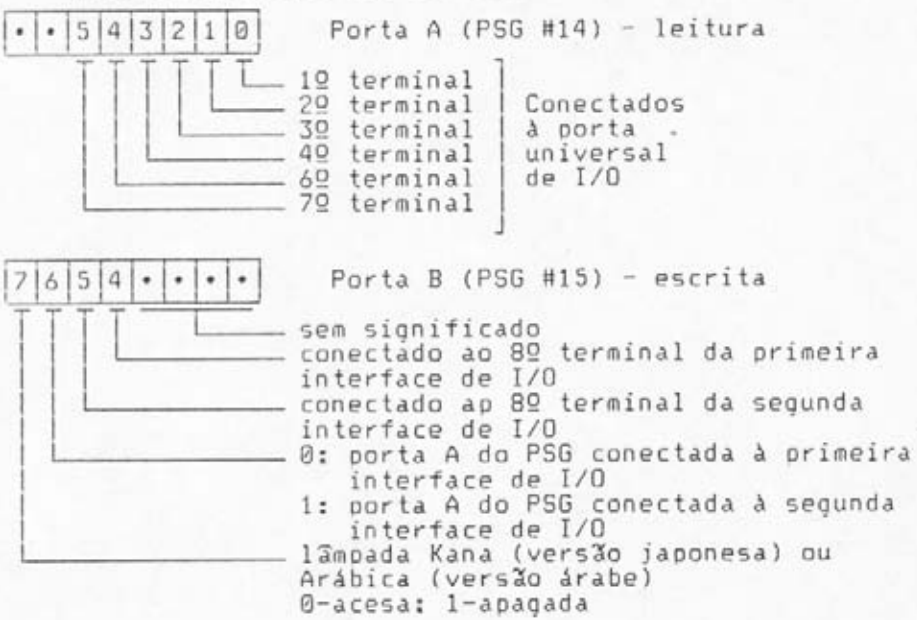
5 - INTERFACE UNIVERSAL PARA I/O

Como descrito no capítulo 5, o PSG tem duas portas de I/O para uso geral. Estas portas são conectadas à interface universal de I/O (portas do joystick). Vários dispositivos podem ser conectados à esta porta, que tem várias rotinas de suporte no BIOS, o que facilita o acesso.

Observe abaixo como as interfaces são conectadas.



As duas portas do PSG são usadas como descrito abaixo:



O acesso à interface universal de I/O deve ser feito preferencialmente pelas rotinas do BIOS descritas abaixo:

GTSTCK (00D5H/Main) - lê status do joystick
GTTRIG (00D8H/Main) - lê status do botão de disparo
GTPDL (00DEH/Main) - lê informação do paddle
GTPAD (00DBH/Main) - acessa vários dispositivos de I/O

6 - CÓDIGOS DE ERRO DO MSX-BASIC

Nº	Original Inglês	Português
01	NEXT without FOR	NEXT sem FOR
02	Syntax error	Erro de sintaxe
03	RETURN without GOSUB	RETURN sem GOSUG
04	Out of DATA	Sem 'DATA'
05	Illegal function call	Chamada ilegal de função
06	Overflow	Overflow
07	Out of memory	Falta memória
08	Undefined line number	Número de linha não definido
09	Subscript out of range	índice fora do limite
10	Redimensioned array	Array redimensionado
11	Division by zero	Divisão por zero
12	Illegal direct	Direto ilegal
13	Type mismatch	Tipo desigual
14	Out of string space	Falta área para string
15	String too long	String muito longa
16	String formula too complex	String muito complexa
17	Can't CONTINUE	Não pode continuar
18	Undefined user function	Função de usuário não definida
19	Device I/O error	Erro de dispositivo I/O
20	Verify error	Verificar erro
21	No RESUME	Sem RESUME
22	RESUME without error	RESUME sem erro
23	Unprintable error	Erro indefinido
24	Missing operand	Falta operando
25	Line buffer overflow	Linha muito longa
26≈49	Unprintable error	Erro indefinido
50	FIELD overflow	Campo maior
51	Internal error	Erro interno
52	Bad file number	Número de arquivo ruim
53	File not found	Arquivo não encontrado
54	File already open	Arquivo já aberto
55	Input past end	Fim de arquivo
56	Bad file name	Nome de arquivo ruim
57	Direct statement in file	Comando direto no arquivo
58	Sequential I/O only	Acesso seqüencial somente
59	File not OPEN	Arquivo não aberto
60	Bad FAT	Erro na FAT
61	Bad file mode	Modo de arquivo errado
62	Bad drive name	Nome de drive errado
63	Bad sector	Setor com erro
64	File still open	Arquivo já aberto
65	File already exists	Arquivo já existe
66	Disk full	Disco cheio
67	Too many files	Diretório cheio
68	Disk write protected	Disco protegido contra escrita
69	Disk I/O error	Erro de I/O de disco
70	Disk offline	Sem disco
71	RENAME across disk	RENAME em discos diferentes
72	File write protected	Arq. protegido contra escrita
73	Directory already exists	Diretório já existe
74	Directory not found	Diretório não encontrado
75	RAM disk already exists	RAMDISK já existe
76≈255	Unprintable error	Erro indefinido

7 - CÓDIGOS DE ERRO DO MSXDOS1

Nº	Original inglês	Português
50	FIELD overflow	Campo maior
51	Internal error	Erro interno
52	Bad file number	Número de arquivo ruim
53	File not found	Arquivo não encontrado
54	File open	Arquivo aberto
55	End of file	Fim de arquivo
56	Bad file name	Nome de arquivo ruim
57	Direct statement in file	Comando direto no arquivo
58	Sequential I/O only	Acesso seqüencial somente
59	File not OPEN	Arquivo não aberto
60	Disk error	Erro de disco
61	Bad file mode	Modo de arquivo errado
62	Bad drive name	Nome de drive errado
63	Bad sector	Setor com erro
64	File still open	Arquivo já aberto
65	File already exists	Arquivo já existe
66	Disk full	Disco cheio
67	Too many files	Diretório cheio
68	Write protected disk	Disco protegido contra escrita
69	Disk I/O error	Erro de I/O em disco
70	Disk offline	Sem disco
71	RENAME across disk	RENAME em discos diferentes

8 - CÓDIGOS DE ERRO DO MSXDOS2

8.1 - ERROS DE DISCO

Nº	Descrição
FFH	Incompatible disk (Disco incompatível)
FEH	Write error (Erro de escrita)
FDH	Disk error (Erro de disco)
FCH	Not ready (Não pronto)
FBH	Verify error (Verificar erro)
FAH	Data error (Erro de dados)
F9H	Sector not found (Setor não encontrado)
F8H	Write protected disk (Disco protegido contra escrita)
F7H	Unfomatted disk (Disco não formatado)
F6H	Not a DOS disk (Disco não DOS)
F5H	Wrong disk (Disco errado)
F4H	Wrong disk for file (Disco errado para arquivo)
F3H	Seek error (Erro de procura)
F2H	Bad file allocation table (Tabela de alocação de arquivos ruim)
F1H	No message (Sem mensagem)
F0H	Cannot format this drive (Este drive não pode ser formatado)

8.2 - ERROS DAS FUNÇÕES DO MSXDOS

DFH	Internal error (Erro interno)
DEH	Not enough memory (Memória insuficiente)
DDH	-
DCH	Invalid MSX-DOS call (Chamada inválida do MSXDOS)
DBH	Invalid drive (Especificação de drive inválida)
DAH	Invalid filename (Nome de arquivo inválido)
D9H	Invalid pathname (Nome da path inválido)
D8H	Pathname too long (Nome da path muito longo)
D7H	File not found (Arquivo não encontrado)
D6H	Directory not found (Diretório não encontrado)
D5H	Root directory full (Diretório raiz cheio)
D4H	Disk full (Disco cheio)
D3H	Duplicate filename (Nome de arquivo em duplicata)
D2H	Invalid directory move (Movimentação de diretório inválida)
D1H	Read only file (Arquivo somente de leitura)
D0H	Directory not empty (Diretório não vazio)
CFH	Invalid attributes (Atributos inválidos)
CEH	Invalid . or .. operation(Operação com . ou .. inválida)
CDH	System file exists (Arquivo de sistema existe)
CCH	Directory exists (Diretório existe)
CBH	File exists (Arquivo existe)
CAH	File already in use (Arquivo já em uso)
C9H	Cannot transfer above 64K (Não pode tranferir mais de 64K)
C8H	File allocation error (Erro de alocação de arquivo)
C7H	End of file (Fim de arquivo)
C6H	File access violation (Violação de acesso ao arquivo)
C5H	Invalid process id (Processo ID inválido)
C4H	No spare file handles (Não há arquivos handle disponiveis)
C3H	Invalid file handle (Arquivo handle inválido)
C2H	File handle not open (Arquivo handle não aberto)

C1H Invalid device operation
(Operação de dispositivo inválida)
C0H Invalid environment string (String inválida)
BFH Environment string too long (String muito longa)
BEH Invalid date (Data inválida)
BDH Invalid time (Hora inválida)
BCH RAM disk already exists (RAMDISK já existe)
BBH RAM disk does not exist (RAMDISK não existe)
BAH File handle has been deleted
(Arquivo handle foi deletado)
B9H Internal error (Erro interno)
B8H Invalid sub-function number
(Número de subfunção inválido)

8.3 - ERROS DE TÉRMINO DE PROGRAMAS

9FH Ctrl-STOP pressed (CTRL+STOP pressionadas)
9EH Ctrl-C pressed (CTRL+C pressionadas)
9DH Disk operarion aborted (Operação de disco abortada)
9CH Error on standard output (Erro na saída standard)
9BH Error on standard input (Erro na entrada standard)

8.4 - ERROS DE COMANDO

8FH Wrong version off COMMAND (Versão errada do COMMAND.COM)
8EH Unrecognized command (Comando não reconhecido)
8DH Command too long (Comando muito longo)
8CH Internal error (Erro interno)
8BH Invalid parameter (Parâmetro inválido)
8AH Too many parameters (Excesso de parâmetros)
89H Missing parameter (Falta parâmetro)
88H Invalid option (Opção inválida)
87H Invalid number (Número inválido)
86H File for HELP not found
(Arquivo para HELP não encontrado)
85H Wrong version of MSX-DOS (Versão errada do MSXDOS)
84H Cannot concatenate destination file
(Arquivo de destino não pode ser concatenado)
83H Cannot create destination file
(Arquivo de destino não pode ser criado)
82H File cannot be copied onto itself
(Arquivo não pode ser copiado nele mesmo)
81H Cannot overwrite previous destination file
(Arquivo de destino não pode ser previamente escrito)