kit DDX2 +

MANUAL DE OPERAÇÃO

Digital Design Eletrônica Ltda
São Paulo, Maio de 1991.

Sumário

Parte 1 - Introdução

O kit DDX2+ transforma seu micro MSX em um MSX2+ adicionando novos comandos (software) e funções (hardware).

O kit DDX2+ pode ser instalado em qualquer computador da linha HOT-BIT e Expert, excetuando-se pelo menos por enquanto o EXPERT Plus e o EXPERT DD Plus.

Este manual foi preparado com a intenção de ser um adendo ao manual original do seu microcomputador, portanto assume que você o tenha sempre à mão, e de preferência já tenha algum conhecimento sobre o MSX e certa familiaridade com o uso do micro.

IMPORTANTE: No kit para o HOT-BIT, algumas facilidades que estão disponíveis no kit para EXPERT não estão presentes (ao menos internamente). Destes, alguns poderão ser adquiridos separadamente na forma de cartuchos (como o DDXDOS2 e o SOM FM por exemplo).

Este manual é composto das seguintes partes além desta:

A parte 2 explica sobre o que é o padrão MSX2+, faz comparações com o MSX1 e MSX2, e fala sobre alguns detalhes adicionais pertinentes ao kit DDX2+.

A parte 3 explica alguns detalhes sobre o vídeo do MSX2+ que você deve saber para poder utilizar todos os recursos da parte gráfica do micro transformado.

A parte 4 descreve detalhadamente os novos comandos disponíveis no MSX2+. Também são explicados comandos que antes existiam mas foram alterados para "comportar" as novas capacidades do micro.

Na parte dos apêndices, você verá como funciona o sistema YJK, como utilizar a memory-mapper, saberá qual a configuração atual dos slots de seu m icro, poderá consultar programas exemplo, etc.

Parte 2 - O MSX2+ e o kit DDX2+

O que é MSX2+

O MSX2+ foi lançado em 1988 no Japão com a intenção de aprimorar ainda mais o padrão MSX, que já havia sido modificado com a entrada do MSX2 em 1985, também no Japão. Certas características que já estavam sendo utilizadas por meio de periféricos foram incorporadas ao design do MSX2+, tal como som "FM" e memory mapper.

Uma alteração significativa foi a mudança do processador de vídeo, que agora pode gerar 19 mil cores, fazer "scroll" de tela tanto horizontal como vertical, aumentou o número de "screens" possíveis (de 0 até 12), podendo mostrar imagens digitalizadas com perfeição (muito melhor que no MSX2 por exemplo), etc.

Outros incrementos são os mesmos existentes no MSX2, tal como relógio em tempo real, RAMDISK em BASIC (aproveitando os 32k de memória RAM geralmente não utilizados pelo BASIC), VRAM (memória de vídeo) de 128k, possibilidade de mostrar texto em 80 colunas, etc.

O kit DDX2+ vai além destas características expostas aqui e inclui também o DDXDOS2 (compatível com o MSXDOS2) embutido na placa. O DDXDOS2 é um sistema operacional em disco e em ROM e em alguns aspectos supera o próprio MS-DOS dos compatíveis com IBM PC (Note que o MSXDOS2 foi originário da própria Microsoft (japonesa).

Algumas características são: Utilização de diretórios, "batch files" mais elaborados, troca de drives lógicos, RAMDISK utilizando a "memory-mapper", e muitas outras.

Mesmo com todas estas características adicionais, o padrão MSX2+ prevê compatibilidade com o MSX2 e com o MSX. É claro que podem existir programas que apresentem problemas, mas isto com certeza será por que os programas foram mal escritos ou mal adaptados (o mercado esteve inundado por este tipo de software, mas felizmente a situação melhorou e as boas "software-houses" estão conscientes, liberando apenas programas que sigam realmente o padrão MSX).

Sobre o assunto "compatibilidade" é bom frisar que o padrão MSX foi criado de certa forma para ser bem flexível; por exemplo: os slots podem ser primários (não expandidos) ou secundários (expandidos),

a RAM pode estar em qualquer um destes slots, e podem variar uma série de outros detalhes. Como a visão de certos programadores infelizmente é limitada, estes fazem seus programas não enxergando que podem existir outros computadores que tenham características diferentes mas deveriam prever esta possibilidade. Até programas de fabricantes de micros japoneses tem este problema, talvez aí com a intenção de obrigar o uso de seu micro. O fato é que poderíamos escrever um livro sobre este problema da compatibilidade e talvez não chegaríamos a conclusão de quem é ou quem são os culpados.

Mais sobre algumas características do kit DDX2+ ainda neste capítulo.

Comparação entre MSX1,2 e 2+

Nesta parte compararemos as diferenças entre os padrões MSX1, MSX2 e MSX2+. Apesar de valer, dentro de certos limites, para qualquer micro nestes padrões, entenda MSX1 como os micros HOT-BIT e EXPERT; MSX2 e MSX2+ como estes micros com os kits DDX2 e DDX2+ da Digital Design instalados.

CPU:

Todos utilizam a mesma CPU ou seja, o Z80A rodando com clock de 3.575611 MHz. Muitos nos perguntaram antes do projeto do kit DDX2+ se não era possível "turbinar" o micro, ou seja, aumentar o clock do Z80. A resposta é SIM e NÃO !

É possível aumentar o clock do Z80! , mas existe um preço? Achamos que não vale a pena comprometer a funcionalidade do hardware básico do micro, que já trabalha perto do limite, além de comprometer possíveis programas que utilizam rotinas de tempo, sem falar nos periféricos que foram projetados para operar na frequência mais baixa. Se um acelerador de micros ("turbo") fosse um produto realmente viável e satisfatório, a Digital Design certamente o estaria produzindo e seria um dos produtos mais vendáveis. Mesmo assim a Digital Design está planejando para um futuro não muito distante lançamentos surpreendentes na área de MSX, e poderá incluir micros mais velozes.

MEMÓRIA ROM:

O MSX vem com 32k de memória ROM no chamado MSX-BASIC versão 1.0 (note que este número de versão refere-se ao padrão

original japonês e não ao número de versões da indústria nacional). No MSX2 a ROM é de 48k (MSX-BASIC versão 2.0), e no MSX2+ é de mais de 64k. No kit DDX2+ completo (incluindo o DDXDOS2) teremos 160k de ROM. No original japonês devido à inclusão de todos os caracteres KANJI e de drivers para escrita em japonês, a soma de toda ROM existente certamente ultrapassa a casa dos 256k.

MEMÓRIA RAM:

O MSX vem com 64k de RAM, o MSX2 e o MSX2+ sem memory-mapper instalada tem os mesmos 64k. Já o kit DDX2+ reconhece automaticamente quantas "mappers" existem no sistema (indicando a memória total no momento do boot) e já vem com 256k internamente. O MSX2+ não requer obrigatoriamente o uso de mappers, mas no kit DDX2+ é incorporado mesmo no modelo mais simples.

No kit DDX2+ (e no MSX2+) a medida em que se instalem cartões de memory-mapper, esta quantidade é automaticamente calculada pelo sistema. O DDXDOS2 também reconhece esta memória podendo ser facilmente utilizada como RAMDISK. Note que o cartucho de memory-mapper deverá estar em conformidade com o padrão para poder ser reconhecida e utilizada. Consulte a Digital Design para informações sobre o cartucho MEGARAM-MAPPER. Mais sobre a memory-mapper no apêndice C deste manual.

VDP (Video Processador):

O MSX utiliza o TMS9918 e tinha 16k de RAM de vídeo e pode mostrar textos de no máximo 40 colunas e apenas 16 cores e o número de "screens" é de apenas quatro. A resolução máxima é de 256 por 192 "pixels", com limitações quanto ao uso de cor, isto é, as 16 cores não estão disponíveis para cada um destes pontos individualmente.

No MSX2 a RAM passa para 128k, mostra textos de 80 colunas, pode mostrar 512 cores com no máximo 256 cores simultaneamente, pode ter 16 cores por sprite e o número de screens é de 9. A resolução é de 512 por 212 pontos e no modo "entrelaçado" a resolução vertical é dobrada (212 para 424 pontos). Tem sprites expandidos ,pode se utilizar até 8 sprites numa mesma linha, pode se definir cores para cada linha dos sprites, e muito mais.O modo "entrelaçado" requer para um melhor funcionamento

um monitor de alta persistência, caso contrário, a imagem fica "piscando" rapidamente e o resultado é visualmente desastroso. Para utilizar o entrelace, o usuário deve escrever o próprio programa ou utilizar algum que faça uso desta característica pois este modo não é diretamente manipulado pelo BASIC. Nos micros japoneses, o entrelace é utilizado para "tentar" mostrar os caracteres japoneses (KANJI) na tela em quantidade razoável.

No MSX2+ a RAM ainda é de 128k, mostra textos de 80 colunas, pode mostrar 19268 (dezenove mil e duzentas e sessenta e oito) cores utilizando-se o sistema YJK ao invés do RGB. Utilizando-se o RGB pode se usar uma palette de 512 cores. A resolução é a mesma do MSX2, ou seja, 512 por 212 (ou 424 no modo entrelaçado). O número de "screens" é de 12. Os sprites também são os mesmos do MSX2. Conheca sobre o sistema YJK lendo o apêndice B deste manual.

RELOGIO EM TEMPO REAL

No MSX inexiste, no MSX2 é alimentado por 2 pilhas pequenas e no Kit DDX2+ por baterias recarregáveis, isto é, não é necessário trocar as pilhas do relógio de tempos em tempos. Neste caso fica óbvio que a bateria se recarrega enquanto o micro estiver sendo utilizado, portanto, se o micro ficar desligado por longos períodos (meses) , a bateria poderá se descarregar por completo, perdendo as informações por ela garantidas.

O relógio, além da data e horário , por intermédio da bateria, também mantem arquivadas certas informações do sistema, como por exemplo tipo de "beep", tamanho da tela, "password", e outros.

RAMDISK em BASIC

Passou a existir no BASIC a partir do MSX2. Foi criado para utilizar a área de memória normalmente não utilizada pelo BASIC (de 0000H até 7FFFH). Note que como a memória do micro é volátil, as informações guardadas neste RAMDISK também o serão. Também deve ser utilizado com precaução quando forem utilizados programas que façam uso desta área, como boa parte dos jogos no mercado.

Note também que no caso da utilização do DDXDOS2, a própria memory-mapper pode ser utilizada como RAMDISK com resultados notadamente melhores.

SOM "FM"

Este talvez, além das novas características do Video Processador, seja uma das melhores coisas que foram incorporadas ao MSX2 +. Quem conhece um pouco de sintetizadores já sabe do que estamos falando. Mesmo quem não conhece, com certeza já ouviu muito deste tipo de som em músicas de sucesso na ultima década (e até hoje). A sintetização em FM (modulação em frequência) foi um marco na história dos sintetizadores digitais (quem não ouviu falar no famoso YAMAHA DX7).

Para que você possa utilizar este verdadeiro sintetizador embutido no seu micro, basta você utilizar as novas instruções do BASIC criadas para este fim, ou então, deliciar-se com os novos jogos (a maioria para MSX2 + utiliza SOM FM).

A Digital Design também proporcionará um cartucho de SOM FM à parte para ser instalado no MSX2 ou mesmo no MSX1 (opção também para quem tem o HOTBIT, já que não vem incorporado ao kit neste caso).

A qualidade de som nem pode ser comparada com o PSG original do micro, embora este continue ativo. Podem ser tocadas 9 vozes (ou canais) simultaneamente e até 3 canais de ritmo (bateria).

Mais detalhes sobre o kit DDX2 +

Aqui falaremos um pouco diretamente sobre algumas características pertinentes ao kit DDX2 + podendo ou não ter relação direta com o padrão MSX2 +.

O kit DDX2 + completo inclui todas as características definidas até agora. Existe também a opção do kit básico, obviamente de menor custo, que só não apresenta: SOM FM e DDXDOS2. Estas características especiais poderão ser incorporadas no momento da compra ou quando o usuário sentir necessidade. Para tanto é só procurar seu revendedor DDX e pedir para incluir um destes dispositivos no seu kit.

Quando o DDXDOS2 vem instalado, existirá uma chave no painel do micro que servirá para habilitar ou não o uso do DOS2. Esta opção permite que você rode certos programas que necessitam de muita

memória ou rodem só com o DOS antigo (por deficiência do próprio software).

Tanto o SOM FM como o DDXDOS2 têm manual explicativo separado, isto é, este manual não vai tratar detalhadamente sobre estes dispositivos.

O kit DDX2 + incorpora também um novo circuito de RESET (com botão para o painel também disponível) que substitue o circuito original do micro. O usuário que já tem incorporado um botão de RESET poderá perguntar: Para que um circuito de RESET se eu tenho o meu que é feito de apenas um botão e um par de fios? A resposta é a seguinte: O novo RESET não é apenas um botão, mas um circuito de POWER-ON RESET, isto é, proporciona um RESET confiável e longo o bastante logo que é percebida uma falha na tensão (isto inclui o liga e desliga do botão da fonte).

Quantas vezes você teve que ligar seu Expert e ele não "entrou na primeira", principalmente depois de quente?

Isto nunca mais vai acontecer (pelo menos por falha de RESET)! Embora não seja prática muito saudável ficar desligando e ligando a fonte rapidamente, com o novo circuito isto é até possível.

O kit DDX2 + também tem agora uma nova saída de vídeo monocromático. Certos monitores de baixo nível não mostravam uma tela bem contrastada e com brilho no kit DDX2, portanto, atendendo o usuário, a DDX aumentou o ganho da saída monocromática. Note que a saída é monocromática pura e não a velha saída de NTSC do vídeo processador utilizada em outros kits que por motivos óbvios "parece" monocromática em monitores PAL-M mas incorpora o sinal de *burst* e toda a modulação do sinal de vídeo colorido que na prática aparece como pequenas barrinhas verticais no vídeo monocromático.

O kit DDX2 + mantem total compatibilidade com os micros MSX2 + japoneses exceto pelo seguinte detalhe: O kit DDX2 + não possui os caracteres japoneses destes micros (do mesmo modo que estes não possuem os caracteres da língua portuguesa) e a chamada KANJI-ROM e os comandos adicionais no BASIC para seu gerenciamento, portanto, em certos jogos de origem japonesa certas mensagens não aparecerão e em certos casos até será gerado algum erro do tipo "SYNTAX ERROR" pois devem ter sido chamados

comandos não existentes (comandos como CALL KANJI, PUT KANJI e outros relativos ao uso da língua japonesa).

IMPORTANTE: Como explicado logo no início deste manual, no kit para o HOT-BIT, algumas facilidades não estão disponíveis na placa.): o DDXDOS2, SOM FM, Memory-mapper, novo circuito de RESET, nova saída de vídeo e bateria recarregável. No caso do SOM FM, DDXDOS2 e memory-mapper poderão ser adquiridos em forma de cartucho opcionalmente (consulte a Digital Design).

Parte 3 - O vídeo no MSX

Passaremos agora a explicar como funciona o vídeo no MSX2 +. É importante que você "fique por dentro" deste assunto para poder utilizar todos os novos recursos de vídeo.

O que é PALETTE

O MSX2 e o MSX2 + utilizam o sistema de palette para definir as cores utilizadas no micro pelo VDP. Palette nada mais é do que uma cor (e nas funções do VDP realmente atua como tal) que pode ser criada pelo usuário dentro de 512 possíveis escolhas.

O sistema trabalha com 16 palettes do mesmo modo que o MSX1 trabalhava com cores. Assim que o micro é inicializado as palettes são construidas automaticamente para que fiquem iguais às cores utilizadas anteriormente pelo MSX1. Por exemplo, você pode mexer com as palettes para mudar as cores mostradas na tela à vontade. Se você quer que algum programa trabalhe com várias tonalidades de verde, por exemplo, você pode redefinir as palettes para obter estas cores como se estivesse usando as cores originais.

Portanto, qualquer uma das palettes (0 a 15) pode ser definida com uma entre 512 cores existentes. Para definir a cor da palette use o comando COLOR =. Todas as cores do MSX podem ser obtidas a partir da combinação dos sinais VERMELHO,VERDE e AZUL. Você define uma palette como a combinação destas três cores em termos de níveis. Estas podem ser combinadas com valores que vão de 0 até 7 cada uma totalizando assim 512 cores possíveis.

Todos os comandos aonde o MSX1 utilizava "número da cor" são agora equivalentes a dizer "número da palette". Exceto quando ressaltado em contrário, todos os comandos de tela (gráficos ou não) utilizam palettes ao invés de cores.

O uso de palettes no MSX2 + é basicamente igual ao do MSX2. O MSX2 + tem um modo adicional de se mostrar cores na tela que é o sistema YJK. Este sistema é complexo e permite a mostra de até 19268 cores simultaneamente na tela. Como dito anteriormente as cores iniciais (ou "default") das palettes são automaticamente selecionadas no momento em que o micro é ativado. Para que você tenha uma idéia de como foram compostas estas cores, daremos a seguir os valores

de cada nível (R,G e B - ou vermelho, verde e azul) que compõe cada cor:

palette	cor obtida	nível R	nível G	nível B
0	transparente	0	0	0
1	preto	0	0	0
2	verde	1	6	1
3	verde claro	3	7	3
4	azul escuro	1	1	7
5	azul claro	2	3	7
6	vermelho escuro	5	1	1
7	ciano	2	6	7
8	vermelho	7	1	1
9	vermelho claro	7	3	3
10	ouro	6	6	1
11	amarelo	6	6	3
12	verde escuro	1	4	1
13	magenta	6	2	5
14	cinza	5	5	5
15	branco	7	7	7

Os tipos de screen

Os tipos de screen (modos) podem variar de 0 até 12. A screen 9 é reservada ao MSX padrão coreano, portanto não é utilizado pelo DDX2+.

As screens 0 a 3 mantém compatibilidade com o MSX1 com algumas ressalvas. As screens 4 a 8 são totalmente compatíveis com o MSX2 e as screens 10 a 12 são únicas ao MSX2+.

Os sprites para as screens 1,2 e 3 são iguais ao MSX1. Para as screens 4,5,6,7,8,10,11 e 12 são sprites modo 2 onde podem ser utilizadas uma cor para cada linha horizontal do sprite além do controle de outras funções para unir sprites; além disto, nestes modos, o VDP

proporciona acesso a comandos gráficos por hardware, tornando o software gráfico muito mais rápido e eficaz.

É importante que você leia sobre o sistema YJK no apêndice B deste manual.

Os tipos de screen são:

SCREEN 0 em 40 colunas

É o modo chamado "TEXT 1" no vocabulário do VDP. Permite 40 colunas de texto por 24 linhas, uma cor para todos os caracteres e uma cor de fundo. Funciona da mesma maneira que no MSX1 com a diferença que as cores possíveis de 0 até 15 são palettes que podem ser escolhidas dentre 512 cores.

SCREEN 0 em 80 colunas

É o chamado modo "TEXT 2". Permite 80 colunas de texto por 24 linhas (ou ainda por 26.5 linhas - se programado diretamente no VDP pois este modo não é suportado pelo BASIC). Funciona como no modo de 40 colunas e pode ter caracteres piscantes (este também é um recurso não suportado diretamente pelo BASIC, portanto não será comentado). Consulte a parte sobre "programas exemplo".

SCREEN 1

É também chamado modo "GRAPHIC 1". Continua o mesmo do MSX1 novamente com a diferença que as cores são palettes escolhidas entre as 512 cores possíveis.

SCREEN 2

É chamado modo "GRAPHIC 2". O mesmo do MSX1, só que utilizando palettes.

SCREEN 3

É chamado modo "MULTICOLOR", que é um modo pseudo-gráfico, onde um caractere é dividido em 4 blocos. É igual ao modo do MSX1 só que utilizando palettes.

SCREEN 4

É chamado modo "GRAPHIC 3". Idêntico à screen 2, com a única diferença que usa sprites no modo 2.

SCREEN 5

É chamado modo "GRAPHIC 4". Resolução de 256 x 212 pixels ou 256 x 192 pixels, cada pixel pode ter uma entre 16 cores diferentes referentes a palettes escolhidas dentre 512 cores. Permite o uso de até 4 páginas distintas (numeradas de 0 até 3). Verifique o comando SET PAGE.

SCREEN 6

É chamado modo "GRAPHIC 5". Resolução de 512 x 212 ou 512 x 192 pixels. Cada pixel pode ter uma entre 4 cores diferentes referentes a palettes escolhidas dentre 512 cores. Pode ser utilizadas também 4 páginas distintas.

SCREEN 7

É chamado modo "GRAPHIC 6". Resolução de 512 x 212 ou 512 x 192 pixels. Cada pixel pode assumir uma entre 16 palettes (dentre 512 possíveis). Podem ser utilizas apenas 2 páginas gráficas.

SCREEN 8

É chamado modo "GRAPHIC 7". Resolução de 256 x 212 ou 256 x 192 pixels. Cada ponto pode assumir uma entre 256 cores possíveis. Neste modo não são utilizadas as palettes. Para os sprites neste modo também são utilizadas 16 cores fixas (ao invés de palettes). As cores possíveis para os sprites são: 0 = preto, 1 = azul escuro, 2 = vermelho escuro, 3 = Púrpura escuro, 4 = Verde escuro, 5 = turquesa, 6 = verde oliva, 7 = cinza, 8 = laranja claro, 9 = azul, 10 = vermelho, 11 = magenta, 12 = verde, 13 = azul claro, 14 = amarelo, 15 = branco. Podem ser utilizadas 2 páginas gráficas distintas. Este modo , apesar de não ser o de maior resolução, era o mais visualmente impressionante no MSX2 por mesclar até 256 cores simultaneamente. Imagine então mesclar mais de 10.000 cores ao mesmo tempo (como nos modos seguintes) !

SCREEN 9

Não utilizado na versão brasileira nem na japonesa. Reservado para uso do MSX coreano.

SCREEN 10

É o mesmo da screen 5, com a diferença de ser utilizado o sistema YJK "mixado" ao invés do sistema convencional para mostra de cores. Isto faz subir o número de cores simultâneas para 12499 ou utilizar 16 palettes entre 512 cores. O consumo de memória neste modo é o dobro, portanto podem ser utilizadas apenas 2 páginas. Note que os comandos gráficos não manipulam diretamente esta quantidade de cores, mas sim utilizam as 16 palettes. Portanto, para usufruir de todas estas cores, o usuário deverá trabalhar diretamente com a VRAM (consulte explicação sobre o YJK no final deste manual).

SCREEN 11

É o mesmo modo da screen 8, mas neste modo utiliza-se o sistema YJK "mixado". Como no caso anterior, podem ser utilizadas 12499 cores distintas ou mixar o uso com 16 palettes. Podem ser utilizadas 2 páginas distintas. Este modo é tratado pelos comandos gráficos como se fosse o modo do screen 8, isto é, utilizando o sistema de cores convencional e não YJK. Então na verdade você não especifica cores da palette nos comandos gráficos, mas sim um número de cor da screen 8 que ia de 0 até 255 (é importante que você consulte a explicação sobre o sistema YJK no final deste manual).

SCREEN 12

É o mesmo modo da screen 8, mas neste modo utiliza-se o sistema YJK "não mixado" ou "sem atribuição", elevando o número de cores simultâneas para 19268 cores, neste caso sem a utilização mixada da palette de cores. Também podem ser utilizadas 2 páginas distintas. Este modo também é tratado pelos comandos gráficos como se fosse a screen 8 (veja explicação sobre a SCREEN 11).

Os sprites

Como mencionado anteriormente, os sprites no MSX2+ podem ser do tipo 1 ou tipo 2. O tipo 1 é o compatível com o MSX1 e é usado nas screens 1 até 3. Nas screens seguintes é utilizado os sprites tipo 2, que podem ter uma cor distinta para cada linha horizontal . Neste modo, conjuntos de sprites podem ser movidos "juntos" como se fossem um único sprite, além de outras propriedades. Consulte os comandos COLOR SPRITE.

Parte 4 - Novos comandos

Introdução

Nesta seção, serão descritas as instruções, funções e variáveis do sistema que foram criadas ou modificadas para dar suporte às novas características do MSX2+. Todas pertencem ao modelo básico do kit DDX2+. As instruções sobre o DDXDOS2 e o SOM FM bem como características e detalhes terão lugar em manual separado, uma vez que são ítens opcionais.

As instruções que não sofreram alterações não serão listadas, portanto, como mencionado anteriormente, este manual é apenas um complemento ao manual original de seu microcomputador

Utilizaremos a seguinte notação: ítens entre chaves ([e]) são opcionais, seguidos de reticências (...) indicam que podem ser usados mais ítens da mesma forma, ítens entre as chaves { e } e separados internamente por uma barra vertical "|" devem ser escolhidos e apenas um deles utilizado e ítens dentro dos sinais ‹ e › mostram o que deve ser colocado naquela posição não devendo ser tomado literalmente como texto ou nome de comando. Escritos em letras maiúsculas devem ser tomados literalmente. Em letras minúsculas indicam que devem ser substituídos. Eis alguns exemplos:

[A] ‹expressão›[,‹expressão› ...]	"A" é um ítem opcional ‹expressão› pode ser usada mais de uma vez.
{ A | B }	deve ser escolhido A ou B
SCREEN	deve se escrever SCREEN
PRESET (x,y)[,‹cor›]	deve se escrever PRESET e colocar o valor da coordenada x e y incluindo os parênteses. O ítem ‹cor› é opcional e se usado deve ser substituído por um valor da cor (sem os caracteres ‹ e ›).

Funções do VDP

☐ SCREEN ‹modo›[,‹tamanho do sprite›[,‹chave do "click"›[,‹baudrate do cassete›[,‹tipo de impressora› [,‹modo de entrelace›]]]]]

Apenas ‹modo› e ‹entrelace› têm novo significado. ‹modo› pode ser especificado de 0 até 12. Modos de 0 até 3 são os mesmos usados pelo MSX 1, modos 4 até 8 são os mesmos utilizados pelo MSX2 e os modos 9,10,11 e 12 são únicos ao MSX2+. ‹modo de entrelace› permite que se utilize as funções de entrelace do VDP.
0 : display normal sem entrelace (default)
1 : display com entrelace
2 : display sem entrelace, mas chaveando telas par/ímpar
3 : display com entrelace e chaveamento de telas.

Se for necessário o chaveamento de telas, deverá ser utilizado um número de screen ímpar (pelo comando SET PAGE) para que ao ser executado este comando, a tela seja chaveada com a tela de número menor.
O modo de entrelace junto com a possibilidade de se chaver entre duas páginas permite que se dobre a resolução vertical de 212 pontos para 424, mas como mencionado no início do manual, seria necessário um monitor de alta persistência para que o resultado fique razoável.

☐ SET PAGE ‹página mostrada›,‹página ativa›

Este comando é novo. Permite que você selecione qual página que deve ser mostrada na tela e qual página será usada para ler ou escrever, isto é, você pode manter uma página na tela enquanto trabalha com outra página para poder mostra-la depois já completamente modificada. Este comando funciona só com os modos de screen de 5 até 12. O número de páginas depende no número da screen: para screen 5 e 6 as páginas podem ser 0,1,2 ou 3 enquanto que nos screens restantes pode ser 0 ou 1.

☐ SET SCROLL [‹direçãoX›[,‹direçãoY›[,‹máscara›[,página]]]]

Este comando também é novo e é único ao MSX2+. Agora é possível mesmo em BASIC fazer scroll horizontal e vertical de telas gráficas. No MSX2 só era possível fazer scroll vertical programando diretamente no VDP. O scroll horizontal era simulado em alguns programas com extensiva cópia de áreas da tela.

Note que neste comando a tela e seus dados não são afetados ; o que muda é a posição que enxergamos a VRAM pelo quadro do monitor. Este comando funciona em todas as screens, mas o resultado depende da screen ativa. Na screen 0 o comando não tem muita aplicação prática, a não ser talvez a obtenção de algum efeito especial. A partir da screen 4, isto é, quando se pode trabalhar com páginas distintas, este comando permite o scroll de uma página à outra (algo como num projetor de slides) - um efeito bem interessante.

Analisemos então o efeito na screen 0. Os valores de ‹direçãoX› e ‹direçãoY› podem variar entre 0 e 7. Se algum deles for zero (ou abreviado) não haverá scroll na direção horizontal ou vertical respectivamente. Se ‹direçãoX› for diferente de zero, a tela será movida de 8-‹direçãoX› pontos horizontais para a direita (move 1 se ‹direçãoX›=7, 2 se ‹direçãoX›=6, e assim por diante). No caso vertical, se ‹direçãoY› for diferente de zero, haverá scroll de ‹direçãoY› pontos na vertical. Neste caso o scroll não se dá na tela, mas dentro de cada linha, "rodando as letras", daí a afirmação de que uma aplicação para isso seria algum efeito especial (veja a seção de programas exemplo para uma mostra desse efeito).

Em screens diferentes de 0, o comando faz realmente o "scroll". Note que o termo "scroll" nos dá idéia de "ir rolando" a tela, mas na verdade, este comando se executado uma única vez altera a posição da tela instantaneamente. Para ter o efeito de continuidade (como na tela de apresentação do MSX2+) temos que executar SCROLL diversas vezes (passo a passo). O valor de ‹direçãoX› pode variar de 0 até 511 enquanto ‹direçãoY› varia de 0 até 255. O conceito de direção varia um pouco para X e Y. Na vertical para cada valor da direção equivale a um pixel e na horizontal cada valor da direção equivale a andar 1/256 da tela inteira, o que dependendo da screen pode ser 1 ou 2 pixels.

Para a vertical, o valor de ‹direçãoY› para a posição normal da tela é 0. A medida que se aumenta o valor da ‹direçãoY› a tela começa "subir"; quando chega em 255 a tela está a um ponto de sua origem. Portanto a sequência para subir a tela seria: 0,1,2,3,...253,254,255,0 e para descer seria: 0,255,254,253,...3,2,1,0. Infelizmente o scroll vertical se feito simplesmente pelo SET SCROLL apresenta um pequeno problema: dependendo da screen, pode aparecer sujeira entre as telas visto que o ciclo completo do movimento é de 256 pontos quando os dados da tela têm no máximo 212 pontos na

vertical. Para contornar este problema, deve-se fazer um "copy" de uma área da tela para a área que fica fora da tela. Veja a parte de "programas exemplo".

Para o scroll horizontal, o valor de ‹direçãoX› também tem como 0 seu ponto de partida. Qualquer valor entre 0 e 255 equivale ao mesmo valor somado de 256, isto é, 256 equivale a 0, 257 equivale a 1 e assim por diante. A primeira vista isso parece redundante, mas tem aplicação no scroll entre duas páginas. Portanto para fazer um scroll horizontal para a direita ‹direçãoX› terá que percorrer os valores 0,255,254,253...2,1,0 ou 256,255,254,...,2,1,0 e para a esquerda os valores: 0,1,2,...253,254,255,0 ou 0,1,2,...253,254,255,256.

Quando <máscara> não é especificada ou tiver valor zero, e fazemos um scroll horizontal com valores não múltiplos de 8, a borda esquerda se desloca irregularmente tampando de 1 até 7 pontos da borda esquerda da tela. Num scroll passo a passo a impressão que se tem é a de que a borda esquerda está pulando. Para contornar este detalhe, poderemos definir ‹máscara› com o valor de 1. Neste caso a borda sempre cobre os 8 pontos da esquerda da tela fazendo com que esta borda não mais "mexa". Especificando ‹máscara› = 1, faz com que sempre se cubra os 8 pontos à esquerda da tela, mesmo que não se faça scroll horizontal. Na verdade são 8 pontos quando se usa screens com resolução de 256 pontos. No caso de resolução de 512 pontos, a máscara cobriria 16 pontos na verdade.

Tudo o que foi dito até agora vale quando utilizamos ‹página› com valor 0 (ou abreviada) não estamos em screens que permitem o uso de páginas (SET PAGE). Quando ‹página› é igual a 1 o scroll horizontal pode ser feito entre as duas páginas, isto é, poderemos passar de uma página a outra gradualmente. Importante: o scroll entre duas páginas só funciona na horizontal. O scroll vertical sempre mostrará a mesma página.

Para utilizar o scroll entre duas páginas, deveremos obedecer certas regras: A página ativa no momento do SET SCROLL deverá ser ímpar, caso contrário o scroll é feito somente com a página de número superior à em uso. Quando ‹direçãoX› é zero, a página mostrada é de número menor; quando ‹direçãoX› é 256 a página mostrada é a atual (número ímpar). Então, para um scroll na prática temos que seguir este procedimento: A página mostrada na tela deverá ser a de número ímpar (1 , 3 ou 1 somente dependendo da

screen utilizada). Para tanto utilize o comando SET PAGE. A página que irá aparecendo aos poucos será a pagina de número menor (0 ou 2). Como 256 é o valor de ‹direçãoX› para mostrar a página ímpar, a sequência para um scroll para a esquerda é: 256,257,258...,510,511,0 e um scroll para a direita é: 256,255,254,...3,2,1,0.

☐ WIDTH ‹largura da tela›

Quando o screen é 0 (modo texto) , a largura pode ser agora de até 80 dígitos.

☐ COLOR [‹cor de frente›[,‹cor de fundo›[,cor de borda›]]]

Altera a cor de frente, cor de fundo e cor de borda de uma tela. A mudança está em que as referências à cores são na verdade referências ao número da palette . A cor de borda nos modos não-texto só é alterada depois de um comando CLS. Se o modo do screen é 0, a cor de borda é ignorada.

Do modo de screen 0 até 7 e 10, cor é um número de 0 até 15 que indica o número da palette. Na screen 8 a cor varia de 0 até 255 e não indica mais uma palette mas a cor mesmo, composta diretamente pelos níveis de vermelho, verde e azul. A cor pode ser calculada como sendo: 32 x (nível de verde) + 4 * (nível de vermelho) + (nível de azul). Nível de verde e vermelho variam de 0 até 7 enquanto nível de azul varia de 0 até 3, perfazendo as 256 cores possíveis nestes modos. Nota: para as screens 11 e 12 a cor varia de 0 até 255 (como na screen 8) mas seu uso difere pois nestes modos a cor é mostrada pelo sistema YJK.

A cor da borda na screen 6 tem um significado especial: Se o bit 4 for igual a 1 ou seja , o valor da cor for maior do que 15, a cor de borda para a coordenadas pares de x pode ser especificada diferentemente da cor da coordenada y. Em resumo: cor de borda = ‹valor de 0 até 15› ou cor de borda = 16 + ‹cor das coordenadas pares› x 4 + ‹cor das coordenadas ímpares›. Neste último caso, só podem ser usadas as cores de 0 a 3.COLOR = (‹número da palette›,‹nível de vermelho›,‹nível de verde›,‹nível de azul›)

Este comando serve para especificar as cores de cada palette individualmente. Quando o computador é ligado, as cores das palettes são inicializadas de modo que sejam iguais às cores normais do MSX (como se não houvessem palettes). Como visto na descrição do comando COLOR anteriormente, as palettes são

usadas nos screens 0 até 7 e 10. Os níveis de vermelho , verde e azul podem variar de 0 até 7, portanto poderemos combinar até 8 x 8 x 8 = 512 cores. Apesar da palette número 0 ser definida como cor transparente, ela poderá se necessario ser utilizada como uma cor normal (não transparente) se for alterado um registro interno do VDP. A instrução: VDP(9) = VDP(9) OR &H20 faz com que a palette 0 seja tratada como palette normal, enquanto: VDP(9) = VDP(9) AND &HDF faz com que seu uso volte ao normal, isto é, seja considerada cor transparente.

☐ COLOR [= NEW]

Esta instrução faz com que as palettes sejam restauradas aos mesmos valores que quando o computador é energizado. É uma boa prática executar o comando COLOR = RESTORE no início e no fim de um programa.

☐ COLOR = RESTORE

Esta instrução faz com que as palettes sejam alteradas em conformidade com a tabela de palettes armazenada na VRAM. Por exemplo: se apenas os dados de uma tela com uma palette pouco usual forem salvos (com BSAVE), a imagem original não poderá ser restaurada apenas por um BLOAD, porque apenas carregando os dados da imagem não faz com que as palettes sejam alteradas. Portanto a imagem deverá ser salva inclusive com a tabela de palettes. Para obter então as cores originais desta tela é só carregá-la com BLOAD e logo após executar o comando COLOR = RESTORE. Para saber qual o endereço final dos dados de uma tela na VRAM, consulte o manual de seu micro para as screens 0 até 3 e a seguinte tabela (para as screens restantes):

screen 4 - de 0 até 3FFFH
screens 5 e 6 - de 0 até 7FFFH
screen 7,8,10,11 e 12 - de 0 até FFFFH

☐ LOCATE [‹x› [,‹y› [,‹chave do cursor›]]]

Esta função foi alterada apenas para prever o modo texto de 80 colunas na screen 0, portanto o valor de x pode variar de 0 até 79.

☐ LINE [{(x1,y1) | STEP(x1,y1)}]-{(x2,y2) | STEP(x2,y2)} [,‹cor›[,{B | BF} [,‹operação lógica›]]]

☐ PRESET {(x,y) | STEP(x,y)} [,‹cor› [,‹operação lógica›]]

☐ PSET {(x,y) | STEP(x,y)} [,‹cor› [,‹operação lógica›]]

Estes comandos foram alterados de forma a prever os novos limites para as coordenadas x e y. A parte nova nestes comandos é que foi incluída ‹operação lógica›. Quando este valor é especificado, a operação lógica é executada entre a ‹cor› especificada e a cor original que já estava "desenhada" e o resultado será efetivamente utilizado como a "cor". Operação lógica:

PSET ou TPSET	Usa somente a ‹cor› especificada
PRESET ou TPRESET	Usa "NOT ‹cor›"
XOR ou TXOR	Usa "cor existente XOR ‹cor›"
OR ou TOR	Usa "cor existente OR ‹cor›"
AND ou TAND	Usa "cor existente AND ‹cor›"

☐ CIRCLE {(x,y) | STEP(x,y)},‹raio›[,‹cor›[,‹ângulo.inicial›[,‹ângulo final›[,‹proporção›]]]]

Esta instrução foi alterada para prever os novos limites de coordenadas e de cores.

☐ PAINT {(x,y) | STEP(x,y)} [,‹cor› [,‹cor de limite›]]

Esta instrução também foi alterada para prever os novos limites de coordenadas e cores (palettes). Nas screens 2 e 4 a especificação de ‹cor de limite› é inválida.

☐ BASE ‹expressão›

Esta variável de sistema contem o endereço de início de cada tabela da VRAM. Para se obter o valor a ser utilizado em ‹expressão› em função do número da screen, utilize o seguinte artifício: multiplique o número da screen por 5; este valor será o número de expressão para acessar a "tabela de nomes de padrão"; acrescente 1 para obter a "tabela de cores" (não disponível na screen 0); acrescente 2 para obter a "tabela do gerador de padrões"; acrescente 3 para obter a "tabela de atributos dos sprites" (não disponível na screen 0); e finalmente acrescente 4 para obter a "tabela de padrão dos sprites" (também sem significado na screen 0). Todos os endereços de início das tabelas podem ser lidos mas apenas escritos quando o valor de ‹expressão› variar de 0 até 19, isto é, das screens 0 até

3. A tabela do screen 4 varia a medida que se altera a tabela do screen 2. Nos screen de 5 até 8 e 10 até 12 o valor do endereço obtido é relativo à página utilizada, isto é, para se obter o valor absoluto, deve-se somar ao valor obtido o endereço inicial da página utilizada. Na verdade os endereços absolutos para estas screens não precisam ser calculados pois as instruções VPEEK e VPOKE também utilizam valores relativos.

☐ VDP (‹número do registro›)

Esta variável do sistema permite que se leia e se escreva em registros do VDP diretamente. O valor do ‹número do registro› é ligeiramente diferente do número do registro real interno do VDP. Veja a correspondência, sendo ‹n› o ‹número do registro› especificado :

‹n›	registro do VDP	modo de acesso
0 até 7	0 até 7 (=MSX1)	leitura e escrita
8	registro de status 0	leitura
9 até 24	8 até 23	leitura e escrita
33 até 47	32 até 46	escrita
-1 até -9	registros de status 1 até 9	leitura

☐ VPEEK (‹endereço›)

☐ VPOKE ‹endereço›,‹dado›

Esta função e este comando permitem leitura e escrita de dados diretamente na VRAM. Foram alterados para permitir acessos de memória na faixa de 0 até 65535 (64k). Para screens 5 até 8 e 10 até 12 o endereço é relativo em relação à página ativa.

☐ BLOAD "‹nome de arquivo›"[{[,R] | [,S]} [,‹offset›]]

☐ BSAVE "‹nome de arquivo›",‹endereço inicial›,‹endereço final›[,{‹endereço de execução› | S}]

Estes são comandos do DISK-BASIC usados para carregar ou gravar os dados da VRAM de ou para arquivos em disco. É bom notar que somente as páginas ativas são válidas quando a screen é de 5 até 8 e 10 até 12.

☐ COPY (x1,y1)-(x2,y2) [,‹página fonte›] TO (x3,y3) [,‹página destino› [,‹operação lógica›]]
COPY (x1,y1)-(x2,y2) [,‹página fonte›] TO {‹nome de um array› | ‹"nome de arquivo"›}
COPY {‹nome de um array› | ‹"nome de arquivo"›} [,‹direção›] TO (x3,y3) [,‹página destino› [,‹operação lógica›]]
COPY ‹"nome de arquivo"› TO ‹nome de um array›
COPY ‹nome de um array› TO ‹"nome de arquivo"›

Os comandos COPY além das funções originais presentes no DISK-BASIC tem mais estas 5 possibilidades. Estes comandos transferem dados gráficos de telas e são válidos quando a screen for de 5 até 8 e 10 até 12.

(x1,y1)-(x2,y2) significa que a área retangular cujas diagonais são estas duas cordenadas é que deve ser transferida ou copiada. ‹página fonte› e ‹página destino› indicam a página "de" e a página "para" onde os dados devem ser copiados. Se estes valores são omitidos é assumida a página ativa. ‹direção› indica como esta área deverá ser escrita na tela, e pode valer de 0 até 3 com o seguinte significado: ‹direção›=0 a área é copiada normalmente; ‹direção›=1 indica que á área deve ser copiada "espelhada"; ‹direção›=2 copia a área de "ponta-cabeça" e ‹direção›=3 copia a área de "ponta- cabeça" e "espelhada".

‹nome de um array› pode ser de tipo inteiro, precisão simples ou dupla, bastando que seja inicializado (pelo comando DIM) com espaço suficiente para compreender a figura desejada. Seu tamanho pode ser calculado (em bytes) pela seguinte expressão: INT ((‹tamanho do pixel› * (ABS(x2-x1)+1) * (ABS(y2-y1)+1)+7/8+4 ‹tamanho do pixel› é o número de bits necessários para expressar um ponto único na tela (que varia dependendo da screen); seu valor é 4 quando o screen é 5 ou 7, 2 para o modo 6, e 8 para os modos 8,10 e 12. Por exemplo, se o valor obtido é 553 bytes, podemos definir o seguinte array de inteiros: DIM A% (277) , pois 277 inteiros ocupam 554 bytes e suportam tranquilamente a imagem. ‹operação lógica› especifica qual a operação que deve ser usada para calcular as novas cores. Veja as possíveis operações lógicas na descrição do comando LINE, PRESET e PSET. Note que quando os operadores forem precedidos de "T", a parte transparente da imagem "fonte" não será transferida.

☐ PUT SPRITE ‹número do plano do sprite›
[,{(x,y) | STEP(x,y)}[,‹cor›[,‹padrão do sprite›]]]

A única alteração refere-se ao fato de que nas screens 1 até 3, o valor da coordenada y era 209 para não mostrar o sprite especificado e 208 para não mostrar o sprite especificado e os sprites seguintes (de número de plano maior). Agora nas screens 4 até 8 e 10 até 12, aonde o limite da coordenada y é 211, os valores especificados para este fim são agora 217 e 216 respectivamente.

☐ COLOR SPRITE$ (‹número do plano do sprite›) = ‹expressão string›

Este comando, válido para as screens 4 até 8 e 10 até 12 especifica a cor de cada linha horizontal de um sprite. ‹expressão string› consiste de uma string de 1 até 16 caracteres. Bits 0 até 3 de cada caractere indica a cor (ou seja, a palette - menos para a screen 8, que indica a cor mesmo). Os bits 4 até 7 são utilizados para especificar funções especiais dos sprites:
bit 7: se 1, move o sprite 32 pontos para a esquerda
bit 6: se 1, move os sprites dos planos seguintes todos juntos. Prioridade e conflitos de sprites são ignorados e quando os sprites são sobrepostos, são mostrados somados.
bit 5: se 1, o conflito de sprites é ignorado.
bit 4: não utilizado.

Estas funções especiais podem parecer à primeira vista estranhas, mas constituem poderoso método para se construir figuras multicoloridas e maiores que um sprite e movimentá-los com a facilidade de um único sprite.

☐ COLOR SPRITE (‹número do plano do sprite›) = ‹cor›

Este comando define a palette (ou cor) de um sprite como um todo. O valor da ‹cor› é definido como o valor do caractere de cor da função COLOR SPRITE$ acima, excetuando-se a função do bit 7, isto é, pode se ligar os bits 6 e 5 do valor da cor para obter as funções definidas acima. Estas funções só são válidas nos screens 4 até 8 e 10 até 12.

☐ SET VIDEO ...

☐ COPY SCREEN ...

Estas funções foram definidas para dispositivos opcionais e não são usadas no kit DDX2+.

Funções do relógio em tempo real

☐ GET DATE ‹variável string› [,A]

Esta função lê a data do relógio em tempo real (ou do valor da data do alarme, quando a opção ",A" for especificada) e a coloca em ‹variável string›. O formato da string é o seguinte: "DD/MM/AA" DD = dia, MM = mes e AA = ano.

☐ SET DATE ‹expressão string› [,A]

Este comando inicializa a data do relógio ou do alarme. A ‹expressão string› pode ser uma string ou uma variável string. O formato deve ser o mesmo do comando GET DATE.

☐ GET TIME ‹variável string› [,A]

O mesmo que GET DATE, mas obviamente referindo-se ao horário ao invés da data. O formato é: "HH:MM:SS" ou seja hora, minutos e segundos. O sistema utilizado é o de 24horas, embora o chip de relógio permite a utilização no modo 12 horas AM e PM.

☐ SET TIME ‹expressão string› [,A]

O mesmo que SET TIME, referindo-se ao horário. Obedecer o formato descrito em GET TIME.

Observação a respeito do alarme.

A data e hora do alarme poderão ser utilizadas para seus próprios programas uma vez que não são utilizadas pelo sistema MSX2+ nesta versão, isto é, nada acontece quando for atingida a data e a hora do alarme. Quando for necessário especificar ambas data e hora, utilizar

SET TIME antes de SET DATE. Obs: os segundos da hora do alarme são automaticamente ignorados.

Comandos para definir as opções "não voláteis"

Os comandos a seguir referem-se a dados que são guardados na memória não volátil do micro e estarão disponíveis e ativos sempre que o micro for re-ligado. Os comandos SET TITLE, SET PROMPT e SET PASSWORD usam a mesma área de RAM, portanto, somente o comando que foi executado mais recentemente é considerado.

☐ SET ADJUST (‹x›,‹y›)

Serve para fazer um ajuste fino da tela mostrada em seu monitor. Os valores de x e y variam de -7 até 8. Os valores devem ser obtidos na prática para obter o melhor resultado. Note que esta função é utilizada para simular um scroll horizontal em certos jogos para MSX2. Os jogos feitos para MSX2+ utilizam-se do poderoso comando SET SCROLL.

☐ SET BEEP ‹timbre›,‹volume›

Especifica o tipo de som gerado pelo comando BEEP (ou quando é impresso o valor 7 (chrS(7)) pela tela. ‹volume› varia de 1 até 4 e dispensa explicações. ‹timbre› pode ser 1 para "tom agudo" (igual ao beep original do MSX), 2 para "tom grave", 3 para um beep de 2 tones (como sinos) e 4 para um beep de 3 tons.

☐ SET PASSWORD ‹string›

Serve para especiffcar uma password ou senha para permitir entrada no sistema. ‹string› é uma expressão string de até 255 caracteres. Nem todos os caracteres são guardados, mas sim um cálculo é feito nos caracteres digitados de modo que possa ser usada uma senha bem comprida, por exemplo: A senha MICROCOMPUTADOR1 é diferente de MICROCOMPUTADOR2 embora o número guardado seja de apenas 16 bits (4x4). Caso se ligue o computador segurando-se a tecla STOP e LGRA, a password não será necessária para entrada no sistema desde que não se corrompa certos valores da memória não volátil do sistema, fazendo com que tenha que se re-inicializar toda esta memória em caso de esquecimento.Para simplesmente desabilitar a

necessidade da PASSWORD, use qualquer um dos comandos SET TITLE ou SET PROMPT ou apenas SET TITLE "".

☐ SET PROMPT ‹string›

Este comando especifica o "prompt" do BASIC, isto é, o "Ok" recebido após cada comando.

☐ SET TITLE ‹string›,‹cor›

Este comando permite especificar a mensagem e a cor da tela inicial no momento da inicialização. A ‹string› pode ser de 1 até 6 caracteres. Caso seja de 6 caracteres, o sistema fica aguardando que você tecle algo para continuar o processo de inicialização interrompido logo após ser mostrada a ‹string›. Caso seja utilizado o comando
SET TITLE ""
a password e a mensagem inicial são cancelados e o prompt volta a ser "Ok".

‹cor› pode ser um valor de 1 até 4 (1 = azul, 2 = verde, 3 = vermelho, 4 = laranja).

☐ SET SCREEN

Este comando salva todos os parâmetros relativos a screen em uso. Na inicialização do sistema estes valores são automaticamente restaurados como se fosse enviado um comando SCREEN. Os seguintes valores são salvos: O número da screen do modo texto, largura da tela (WIDTH), cores de frente, fundo e borda, chave das teclas de função (ON ou OFF), chave de "CLICK" (ON/OFF), tipo de impressora (MSX ou ABNT), baud-rate do cassete e modo do diplay.

Comandos para a RAMDISK do BASIC

No MSX1 a memória RAM de 0000H até 7FFFH era apenas usada pelo DOS. No MSX2 e MSX2+ entretanto, esta porção de memória pode ser tratada como RAMDISK de até 32k bytes. O formato do nome de arquivo para a RAMDISK é descrita abaixo, onde ‹nome› é uma string que consiste de 1 até 8 caracteres e ‹extensão› é uma string que consiste de 1 até 3 caracteres. Note que ":", ".", caracteres de controle

0-31 e símbolos gráficos consistindo de 2 bytes não são considerados caracteres válidos.

formato = MEM:‹nome›[.‹extensão›]

exemplos: MEM:DDX2.1
MEM:ABC123
MEM:1.X00
MEM:COMPUTAD.OR

As seguintes operações são válidas ao se utilizar a RAMDISK:

LOAD,SAVE,RUN e MERGE de programa BASIC (sempre salvo em formato ASCII.) Quando qualquer um destes comandos é executado de dentro de um programa, o controle retorna ao nível de comandos.

OPEN, CLOSE, PRINT#, PRINT USING#, INPUT#, LINE INPUT#, INPUT$, EOF, LOC, LOF. Apenas de arquivos sequenciais.

A RAMDISK não suporta os seguintes comandos:

qualquer leitura e escrita de arquivos randômicos - BLOAD, BSAVE, COPY

Os comandos criados para o controle da RAMDISK são:

☐ CALL MEMINI [(limite superior da RAMDISK›)]

Este comando especifica o limite de memória superior da RAMDISK, inicializa a RAM deletando todos os arquivos que porventura existirem. Este comando deve ser usado sempre antes de se usar a RAMDISK.

☐ CALL MFILES

Este comando lista na tela os arquivos guardados na RAMDISK.

☐ CALL MKILL ("‹nome do arquivo›")

Este comando é utilizado para deletar um arquivo na RAMDISK.

☐ CALL MNAME ("‹nome antigo›" AS "‹nome novo do arquivo›")

Este comando troca o nome de um arquivo na RAMDISK.

Outras adições

☐ PAD ‹expressão›

Esta função retorna o valor do "touch pad", mouse ou "track-ball". Quando ‹expressão› varia de 0 até 7, a função retorna o estado do "touch-pad" como no MSX1.

Os valores de ‹expressão› de 8 até 11 eram utilizados no MSX2 para retornar o estado da "light-pen". No MSX2 + o suporte à "light-pen" foi cancelado (hardware).

Quando ‹expressão› varia de 12 até 19 esta função retorna o estado do mouse ou do "track-ball" (12 até 15 quando conectado à porta de joystick 1, e 16 até 19 quando conectado à porta de joystick 2).

Os dados das coordenadas X e Y só deverão ser lidas após executar-se PAD(12) ou PAD(16). A função STRIG deverá ser usada para ler o estado dos botões do mouse ou track-ball.

‹expressão›	valor retornado
12,16	-1 (calcula coordenadas)
13,17	coordenada X
14,18	coordenada Y
15,19	0 - não utilizado

Apêndice A

Configuração dos Slots

As próximas tabelas mostram a configuração dos slots do micro antes e depois da transformação. As 2 primeiras referem-se ao Expert.

		EXPERT (sem o kit DDX2+)			
		slot 0	slot 1	slot 2	slot 3
Pág	3		SLOT "A" (esq.)	RAM 64k	SLOT "B" (direito)
	2				
	1	ROM Expert			
	0				

		EXPERT 2+ (com o kit DDX2+)						
		slot 0	slot 1.0	slot 1.1	slot 1.2	slot 1.3	slot 2	slot 3
Pág	3		DOS2	SLOT "A" (esq.)	subrom +FM	Uso Futuro	Memory Mapper 256k	SLOT "B" (direito)
	2							
	1	ROM DDX2+						
	0							

As tabelas abaixo referem-se ao HOTBIT.

		HOTBIT (sem o kit DDX2+)			
		slot 0	slot 1	slot 2	slot 3
Pág	3		SLOT "1" (sup.)	SLOT "2" (lateral)	RAM 64k
	2				
	1	ROM HOTBIT			
	0				

		HOTBIT 2+ (com o kit DDX2+)						
		slot 0	slot 1	slot 2.0	slot 2.1	slot 2.2	slot 2.3	slot 3
Pág	3		SLOT "1" (sup.)	SLOT "2" (lateral)	subrom	reservado		RAM 64k
	2							
	1	ROM DDX2+						
	0							

Apêndice B

O sistema YJK

Esta seção poderia ser entitulada "O milagre da multiplicação das cores". Como poderemos ter até 19 mil cores utilizando a mesma quantidade de VRAM dos modos do MSX2 que permitiam "apenas" 256 cores ?

Quem mais se beneficia com todas estas cores são realmente os aplicativos que trabalham com imagens digitalizadas. Imagine uma imagem de uma foto ou de uma paisagem ou melhor ainda, a imagem que você vê na televisão. Porque ela é tão mais real do que gráficos de computador ou imagens geradas por outros micros que nem se aproximam de desenhos animados de baixo nível ? É sabido que a televisão tem mais baixa resolução do que o microcomputador "mais simplório" do mercado. Você já deve ter percebido isso quando tentou ligar um micro direto na televisão por meio da antena e tentou "enxergar" em 80 colunas. Mas por que ainda a imagem de TV é mais real do que a do micro ?

Juntando estas perguntas e afirmações chegamos a conclusão de que a "realidade" ou "a impressão que se tem" de uma imagem não se mede tanto pela resolução, mas sim pela quantidade de cores e nuances (ou brilho) de uma mesma cor. Como no MSX2+ uma tela pode ter na maioria dos screens uma resolução de 256 x 212 pixels, e esta resolução é bem melhor que a resolução da tv, foi "bolado" o sistema YJK que faz com que em cada grupo de 4 pixels seja mantida a mesma tonalidade de cor, mas podendo se variar o brilho desta cor para cada ponto. Isso equivale em termos grosseiros a dizer que a resolução continua alta para o brilho da imagem (em que cada pixel pode ter 32 ou 16 tons) e foi diminuida para 64 x 212 para as cores que ainda podem variar em milhares de tonalidades. Note que mesmo assim as cores de um pixel dentro deste grupo de 4 pixels na prática varia de cor, pois pode ter brilhos diferentes. Para casos como imagens digitalizadas este sistema é perfeito! Se for necessário mixar imagens digitalizadas com gráficos que necessitem certa resolução e variação de tonalidades pixel a pixel, existe o sistema YJK "mixado" ou seja, utiliza

a possibilidade desta multiplicação de cores mixada com o sistema de cores convencional para cada pixel.

Para quem conhece alguma coisa sobre vídeo (eletronicamente falando) e gosta de um pouco de elocubração matemática, voce pode notar que o termo YJK sugere o seguinte:

Um sinal de vídeo pode ser descrito em termos de luminância e crominância, isto é, o valor do brilho e o valor da cor. Esta separação é um resquício do tempo da televisão preto-e- branco que só entendia luminância e por este modo até hoje é compatível com as redes de televisão e videocassetes. Portanto tenha em mente que "Y" é o valor da luminância. JK sugere um sistema de coordenadas que unido ao sinal de luminância Y podem indicar uma cor no espectro de cores. Na prática é o que faz um aparelho de televisão quando converte o sinal que recebe da antena em valores de tensão aplicados nos seus 3 canhões (vermelho,verde e azul). A única diferença , embora não afete em nada pois é uma questão de definição, é que em vídeo se fala em sinal R-Y e B-Y e para o VDP é R-Y e G-Y. O VDP literalmente converte o YJK eletrônicamente para RGB e dai é obtida a saída de vídeo do micro.

As fórmulas para a conversão YJK → RGB e vice-versa são as seguintes:

de YJK para RGB :

```
R = Y + J
G = Y + K
B = 5Y/4 - J/2 - K/4
```

de RGB para YJK

```
Y = B/2 + R/4 + G/8
J = R-Y
K = G-Y
```

o valor de Y varia de 0 até 31 no modo "não mixado" e 0-15 no modo "mixado", J e K variam de -32 a 31 (64 valores possíveis).

Voce já deve estar fazendo as contas e verificou que podem ser definidas 131072 cores para o modo "não mixado" e 65536 cores para o modo "mixado". Matematicamente é correto, mas parte destas combinações se perde na transformação e no fato do conversor do VDP ter resolução de 5 bits para cada sinal (R,G e B), que também matematicamente poderia gerar 32768 cores. Além do mais , com uma

resolução de 256 x 192 você tem "apenas" 49152 pixels; se o cálculo acima se tornasse realidade, não haveria pontos na tela para mostrar todas as cores possíveis. Portanto acredite: No modo "mixado" temos 12499 cores contra 19268 no modo "não mixado".

Como o YJK é utilizado na VRAM

No sistema convencional, cada pixel (no screen 8) ocupava 1 byte de memória VRAM ou 0.5 byte (no screen 5). Estes 2 modos (5 e 8) foram utilizados para gerar os novos modos 10,11 e 12. No sistema YJK cada pixel ocupa 1 byte mas está preso em termos a um grupo de 4 pixels (ou seja, 4 bytes). Por isso o consumo de memória no screen 10 é o dobro do screen 5.

Portanto vamos trabalhar com um grupo de 4 pixels adjacentes. Vamos chamá-los de P1,P2,P3 e P4. No sistema YJK "não mixado" ou "sem atributos" temos o seguinte:

pixel	7	6	5	4	3	2	1	0
P1	$Y1_4$	$Y1_3$	$Y1_2$	$Y1_1$	$Y1_0$	KL_2	KL_1	KL_0
P2	$Y2_4$	$Y2_3$	$Y2_2$	$Y2_1$	$Y2_0$	KH_5	KH_4	KH_3
P3	$Y3_4$	$Y3_3$	$Y3_2$	$Y3_1$	$Y3_0$	JL_2	JL_1	JL_0
P4	$Y4_4$	$Y4_3$	$Y4_2$	$Y4_1$	$Y4_0$	JH_5	JH_4	JH_3

O sinal Y de cada ponto varia de 0 a 31, enquando K (composto por KH e KL) e J (composto por JH e JL) que valem para os 4 pontos variam de -32 a 31. Portanto a cor de P1 é composta por Y1,J e K ; a de P2 é composta por Y2,J e K ; e assim por diante. Então como pode se perceber o valor Y varia de ponto para ponto independentemente enquanto J e K ficam fixos para o grupo.

No sistema YJK "mixado" temos o seguinte:

pixel	7	6	5	4	3	2	1	0
P1	$Y1_3$	$Y1_2$	$Y1_1$	$Y1_0$	A	KL_2	KL_1	KL_0
P2	$Y2_3$	$Y2_2$	$Y2_1$	$Y2_0$	A	KH_5	KH_4	KH_3
P3	$Y3_3$	$Y3_2$	$Y3_1$	$Y3_0$	A	JL_2	JL_1	JL_0
P4	$Y4_3$	$Y4_2$	$Y4_1$	$Y4_0$	A	JH_5	JH_4	JH_3

Podemos então para cada grupo de 4 pixels ter o sistema YJK e o sistema convencional de cores (palettes) mixado. Vejamos: Se o A

(bit3) valer 0, então este grupo de pixels utiliza o sistema YJK com a unica diferença que o valor Y neste caso varia de 0 até 15 e não mais de 0 até 31 como no modo "não mixado". Os valores de K e J seguem o mesma formação do modo "não mixado". Mas se A for 1, então os valores de J e K serão desprezados e Y1,Y2,Y3 e Y4 que variam de 0 até 15 serão o número de uma palette do sistema convencional.

Os comandos gráficos do BASIC sabem distinguir a screen 5 da screen 10 (que consome o dobro da memória) e ligam o bit de atributo do "YJK" mixado para continuar trabalhando com palettes ; neste caso para utilizar os recursos do YJK o usuário deverá escrever diretamente na VRAM.

Entretanto, o BASIC e seus comandos gráficos tratam as screens 11 e 12 como se fossem a screen 8, isto é, fazem todas operações gráficas assumindo o modo convencional de cores, utilizando cores de 0 até 255 (compostas por R,G e B). Vamos utilizar um exemplo: Imagine que você na utilizou os seguintes comandos em um programa:

```
SCREEN 8
COLOR ,72:CLS
PSET (0,0),61
```

Então, após estes comandos, os 4 primeiros bytes da VRAM estarão com:

byte 0 =	00111101	(61 decimal)
byte 1 =	01001000	(72 decimal)
byte 2 =	01001000	
byte 3 =	01001000	

Portanto você ligou o primeiro ponto da tela com a cor 61 (que decomposta em níveis RGB dá R=7,G=1,B=1, ou seja , a cor vermelho claro. Obviamente as cores dos pontos adjacentes não serão afetadas continuando com uma cor marrom.

Agora, imagine que você utilizou SCREEN 11 ao invés de SCREEN 8. Como a screen 11 utiliza YJK "mixado", vejamos o que acontece: O byte de atributo do primeiro pixel (0,0) está ligado, portanto os 3 bits menos significativos serão ignorados e a cor deste pixel será igual a palette número 3 que deve ser "verde-claro" (se não foi modificada a palette). Note que a tela inteira, ao invés de apresentar a cor esperada

(marrom) apresenta a cor "azul escuro" ,pois com todos os bits de atributo ligados teremos como cor de tela a palette 4.

Agora, suponha que trabalhássemos com a screen 12, que usa o sistema YJK não mixado. Teremos 3 cores sendo apresentadas na tela: A cor do primeiro pixel (0,0), a cor dos 3 pixels seguintes (1,0), (2,0) e (3,0) e a cor do restante da tela. A cor do primeiro pixel será definida pelo sistema YJK com Y=7, J=0, e K=5. Os tres pixels seguintes terão Y=9,J=0,K=5. A tela restante terá Y=9,J=0,K=0. Transformando para RGB em níveis de 0 até 100% para cada componente, teremos algo próximo dos seguintes conjuntos RGB: (41%,47%,36%) para o primeiro pixel, (44%,49%,39%) para os tres pixels seguintes e (44%,44%,40%) para o restante da tela. Em termos de cores "palpáveis" teremos algo perto de verde-escuro, um verde-não-tão-escuro e um roxo (todos com pouco brilho).

Concluindo: você poderá utilizar os comandos de desenho do BASIC nos modos 11 e 12 como fazia no modo 8, mas você notou pela explicação que foi feita agora que teremos cores trocadas e algum "borramento" também. Mas é claro que conhecendo-se bem o sistema YJK, estes detalhes poderão ser contornados utilizando-se os comandos gráficos tendo em mente que a cor especificada (0 a 255) é na realidade mais complexa. Fora isso, as telas se comportam de maneira idêntica.

Apêndice C

A memory-mapper

Para que você entenda esta explicação, é necessário que você tenha algum conhecimento sobre a arquitetura do MSX (como Slots, memória, etc). Memory-mapper não deve ser confundida com MEGARAM, apesar de serem de conceito muito parecido, diferem no modo de acesso.

Como você deve saber, o Z80 (que é o processador do MSX) endereça diretamente apenas 64k bytes. No MSX isto foi contornado utilizando-se o conceito de Slots, que você já deve conhecer.

Memory-Mapper foi uma maneira bolada para se utilizar mais memória RAM no MSX além dos 64k normais, sem é claro, "gastar" mais slots. Geralmente a memory-mapper é incorporada à propria RAM do micro (no caso do Kit DDX2+ ela é a própria RAM principal do micro e tem 256k), mas nada impede que seja adicionada externamente por meio de cartucho (desde que obedeça o padrão correto de acesso).

A menor mapper tem 64k (apesar de parecer absurdo, já que um micro normal já tem 64k sem precisar de mapper, é utilizado por alguns micros japoneses que tem o circuito da mapper embutido mas não foi colocada memória adicional) e vão geralmente dobrando de tamanho (128k, 256k, 512k, e assim por diante).

O kit DDX2+ permite que você ligue um cartucho de mapper externo (novamente desde que o cartucho seja feito dentro do padrão mapper) como o DDX MEGA-MAPPER de 512k da Digital Design. Assim o micro ficará com memória total de 768k.

Note que programas feitos para MSX quando ainda não dispunham da memory-mapper nunca vão nunca utilizar esta memória adicional. O BASIC se inclui nesta categoria pela sua própria concepção.

Novos programas para MSX já "sabem que a mapper existe" e podem ou não fazer uso dela. O MSXDOS2 faz uso da mapper, ou melhor, requer a mapper para que possa funcionar. Ele pode e utiliza a mapper como RAMDISK.

Pode ocorrer que alguns programas não "rodem" com a mapper do kit DDX2+ ou quando tenham mais uma mapper externa conectada. Tais

programas pertencem à categoria "MAL ADAPTADOS" ou "MAL FEITOS".

Portanto, se você deseja utilizar a mapper em seus programas ou deseja saber mais profundamente sobre ela, saiba agora como proceder para utiliza-la corretamente (dentro do padrão).

Para entender a mapper, você deve imaginar o bloco de 64k de RAM dividido em 4 partes (ou páginas):

Página 0	0000 - 3FFFH
Página 1	4000-7FFFH
Página 2	8000-BFFFH
Página 3	C000-FFFFH

Cada página comporta 16Kbytes. Se a mapper tem 256k (no caso da mapper embutida do kit DDX2+), então ela tem 256/16 = 16 bancos de 16k. Imagine o slot em que está a mapper como sendo uma janela de 64k incorporando as 4 páginas (0,1,2 e 3). Através de escrita em determinadas portas de I/O poderemos fazer com que esta janela "enxergue", em cada uma das páginas, qualquer um dos 16 bancos possíveis (no nosso caso).

Os endereços de I/O são: FCH para a página 0, FDH para a página 1, FEH para a página 2 e FFH para a página 3. Portanto basta escrever o número do banco desejado (de 0 até o número total de bancos-1, ou seja de 0 até 15 no nosso caso) na porta de I/O desejada.

Você pode notar que podemos fazer com que o mesmo banco apareça em duas ou mais páginas. Exemplo: se for escrito o valor do banco 0 nas páginas 2 e 3, ao alterar o valor do endereço 8000H a modificação poderá ser verificada também no endereço C000H uma vez que é na realidade a mesma posição física da RAM. Portanto o acesso a estas portas de I/O deverá ser feito com muito cuidado e conhecimento.

Geralmente, estas portas de I/O podem também ser lidas, devolvendo o valor nelas gravado. Deve-se notar que o valor lido geralmente reflete apenas os bits utilizados para representar todos os bancos possíveis. Por exemplo, no caso do kit DDX2+, utiliza-se apenas os 4 bits menos significativos. Os outros bits serão lidos como 1, veja: Se for gravado o valor 0 em uma das portas, o valor retornado por ela será 240 (ou seja 11110000 em binário).

Note que se existir outra mapper conectada ao sistema e ela tiver 512k (necessitando para tanto de 5 bits para representar os 32 bancos

possíveis) o valor devolvido será 224 (ou 11100000 em binário). Note que não existe conflito de I/O na leitura pois os valores devolvidos dos bits em comum são os mesmos e os bits não usados não são acionados pela mapper "menor".

Você pode perceber então que podem exister 1, 2 ou mais sistemas de mapper no mesmo micro ocupando slots diferentes. Pode inclusive haver um slot com RAM mas que não seja mapper e outro slot com uma mapper (caso de uma mapper externa instalada em um micro sem mapper). Este é o motivo de tantos softwares que "se dizem" funcionar em mapper não trabalhar satisfatoriamente com a mesma, pior ainda quando houver mais de uma mapper conectada.

Resumindo, devem ser inspecionadas todas as áreas com RAM para saber se nelas existe uma mapper. O indicador de memória do MSX2+ quando do momento da inicialização do sistema (quando o micro é ligado) é um ótimo exemplo disto.

O MSXDOS2 também faz esta busca de memória pelo sistema e permite que se utilize como RAMDISK parte ou totalidade da memória não necessária para suas funções básicas.

Para quem já estiver familiarizado com linguagem de máquina e conhecer bem o Z80 e sua arquitetura, estas informações bastam. Verifique o programa exemplo de utilização da mapper no fim deste manual. Para quem ainda tem muitas dúvidas, sugerimos que não tente utilizar a mapper em seus próprios programas antes de se sentir bem mais à vontade na utilização de linguagem de máquina.

Apêndice D

ESTES PROGRAMAS ESTÃO NO DISQUETE DO SPACE MATROW E VÁRIOS

Programas exemplo

Neste apêndice são listados vários programas exemplo que o ajudarão a entender melhor o funcionamento do MSX2+ e do kit DDX2+.

São programas os mais variados e criados o mais simples possível. A maioria dos novos comandos são exemplificados em um ou em mais de um programa.

PALETES.BAS

```
1000 '
1100 ' EXEMPLO DE DEFINICAO DAS PALETTES
1200 '
1300 DIM R(15),G(15),B(15):' Niveis para as 16 cores
1400 RESTORE
1500 FOR I=0 TO 15:READ R(I):NEXT:' Le niveis para RED
1600 FOR I=0 TO 15:READ G(I):NEXT:' Le niveis para GREEN
1700 FOR I=0 TO 15:READ B(I):NEXT:' Le niveis para BLUE
1800 COLOR = NEW:' restaura
1900 VDP(9)=VDP(9) OR &H20:' cor 0 = nao transparente
2000 SCREEN 2
2100 COLOR15,0,0:CLS
2200 OPEN "GRP:" AS #1
2210 ' barras para as 16 cores
2300 FOR I=0 TO 15
2400   LINE (I*16,0)-(I*16+15,100),I,BF
2500   PSET (I*16+4,110),0
2600   PRINT #1,USING "#";INT (I/10)
2700   PSET (I*16+4,120),0
2800   PRINT #1,USING "#";I MOD 10
2900 NEXT
2910 ' desenho da seta no sprite
3000 SPRITE$(0)=CHR$(&H18) + CHR$(&H3C) + CHR$(&H7E)+
CHR$(&HFF) + STRING$(4,&H18)
3100 PSET (16,150),0
3200 PRINT#1,"<SETAS> SELECIONAM A PALETTE":
```



```
3300 PSET (16,160),0
3400 PRINT#1,"USE 'r,g,b' e 'R,G,B' !";
3500 I=8:' COMEÇA COM A PALETTE 8
3550 '
3600 PUTSPRITE 0,(I*64+4,130),15,0
3700 A$=INPUT$(1)
3800 IF A$=CHR$(27) THEN COLOR=NEW:END
3900 IF A$="R" THEN GOSUB 4900:' aumenta nivel R
4000 IF A$="r" THEN GOSUB 5100:' diminui
4100 IF A$="G" THEN GOSUB 5300:' aumenta nivel G
4200 IF A$="g" THEN GOSUB 5500:' diminui
4300 IF A$="B" THEN GOSUB 5700:' aumenta nivel B
4400 IF A$="b" THEN GOSUB 5900:' diminui
4500 IF A$=CHR$(28) THEN I=(I+1) MOD 16:GOTO 3600
4600 IF A$=CHR$(29) THEN I=(I+15) MOD 16:GOTO 3600
4700 COLOR = (I,R(I),G(I),B(I))
4800 GOTO 3600
4900 IF R(I)<7 THEN R(I)=R(I)+1
5000 RETURN
5100 IF R(I)>0 THEN R(I)=R(I)-1
5200 RETURN
5300 IF G(I)<7 THEN G(I)=G(I)+1
5400 RETURN
5500 IF G(I)>0 THEN G(I)=G(I)-1
5600 RETURN
5700 IF B(I)<7 THEN B(I)=B(I)+1
5800 RETURN
5900 IF B(I)>0 THEN B(I)=B(I)-1
6000 RETURN
6050  ' DATAs para imitar os valores "default"
6100  DATA 0,0,1,3,1,2,5,2,7,7,6,6,1,6,5,7
6200 DATA 0,0,6,7,1,3,1,6,1,3,6,6,4,2,5,7
6300 DATA 0,0,1,3,7,7,1,7,1,3,1,3,1,5,5,7
```

SCREEN0.BAS

```
1000 '
1100 ' EXEMPLO DA SCREEN 0
1200 ' DOS COMANDOS WIDTH,LOCATE
1300 '
```

```
1400 SCREEN 0:WIDTH40:COLOR 15,4:CLS
1500 KEY OFF
1600 FOR X=0 TO 38
1700 LOCATE X,0:PRINT"#";:LOCATE X,23:PRINT"#";
1800 NEXT
1900 FOR Y=0 TO 22
2000 LOCATE 0,Y:PRINT"#";:LOCATE 39,Y:PRINT"#";
2100 NEXT
2200 LOCATE 5,10:PRINT"SCREEN 0 40 COLUNAS, TECLE ALGO";
2300 A$=INPUT$(1)
2400 WIDTH 80:CLS
2500 FOR X=0 TO 78
2600 LOCATE X,0:PRINT"#";:LOCATE X,23:PRINT"#";
2700 NEXT
2800 FOR Y=0 TO 22
2900 LOCATE 0,Y:PRINT"#";:LOCATE 79,Y:PRINT"#";
3000 NEXT
3100 LOCATE 5,10:PRINT"SCREEN 0 80 COLUNAS, TECLE ALGO";
3200 A$=INPUT$(1):CLS
```

SPRITES2.BAS

```
1000 '
1100 ' EXEMPLO DE SPRITES TIPO 2
1200 ' OU EXTENDIDOS
1300 '
1400 SCREEN 4,3:COLOR 15,14,2:CLS
1500 SPRITE$(0)=STRING$(32,255):' bloco
1600 PUT SPRITE 0,(50,50),15,0
1640 ' Muda a cor do sprite como um todo
1700 FOR C=0 TO 15
1800  COLOR SPRITE (0)=C
1900  FOR X=1 TO 200:NEXT
2000 NEXT
2050 ' Uma cor para cada linha do sprite
2100 FOR C=0 TO 15:C$=C$+CHR$(C):NEXT
2200 COLOR SPRITE$(0)=C$
2250 ' Movimenta o sprite pela tela
2300 FOR X=50 TO 200
2400 PUTSPRITE 0,(X,50),,0
```



```
2500 NEXT
2600 FOR X=200 TO 50 STEP -1
2700 PUTSPRITE 0,(X,50),,0
2800 NEXT:GOTO 2300
2900 GOTO 2900
```

OPLOGICA.BAS

```
1000 '
1100 'EXEMPLO DE OPERACOES LOGICAS
1200 'NOS COMANDOS GRAFICOS
1300 '
1350 'A mesmo desenho sera feito 5 vezes mudando-se
1360 'apenas a "operacao logica"
1370 '
1400 SCREEN 7:COLOR 15,14,4:CLS
1500 LINE (50,50)-(310,120),4,BF
1600 LINE (100,70)-(350,150),6,BF,PSET
1700 A$=INPUT$(1):CLS
1800 LINE (50,50)-(310,120),4,BF
1900 LINE (100,70)-(350,150),6,BF,PRESET
2000 A$=INPUT$(1):CLS
2100 LINE (50,50)-(310,120),4,BF
2200 LINE (100,70)-(350,150),6,BF,XOR
2300 A$=INPUT$(1):CLS
2400 LINE (50,50)-(310,120),4,BF
2500 LINE (100,70)-(350,150),6,BF,OR
2600 A$=INPUT$(1):CLS
2700 LINE (50,50)-(310,120),4,BF
2800 LINE (100,70)-(350,150),6,BF,AND
2900 GOTO 2900
```

SETPAGE.BAS

```
1000 '
1100 ' EXEMPLO DE SET PAGE
1200 '
1300 SCREEN 5
1400 COLOR 15,14,2
```

```
1500 OPEN"GRP:" AS #1
1600 SET PAGE 0,0:CLS
1650 ' preenche as 4 paginas enquanto
1660 ' mantem a pagina 0 na tela.
1700 FOR I=0 TO 3
1800  SET PAGE 0,I
1900  CLS
2000  PSET (10,10),0
2100  PRINT#1,"PAGINA "I
2200  S=I*10
2300  LINE (S+50,S+50)-(S+200,S+150),I+5,BF
2400 NEXT
2500 I=0
2550 ' muda as paginas "mostradas"
2600 SET PAGE I,I
2700 A$=INPUT$(1):I=(I+1) MOD 4
2800 GOTO 2600
```

GETTIME.BAS

```
1000 '
1100 ' EXEMPLO DE GET TIME,DATE
1200 '
1300 SCREEN 0:COLOR 15,4
1400 LINE INPUT "HORARIO PARA ALARME (HH:MM:SS) ";AL$
1500 SET TIME AL$,A
1600 CLS
1610 ' mostra a data e hora
1700 LOCATE 10,2:PRINT"DATA ";
1800 LOCATE 10,4:PRINT"HORA ";
1900 GET DATE A$:LOCATE 15,2:PRINTA$
2000 GET TIME A$:GET TIME X$,A:LOCATE 15,4:PRINTA$
2005 ' enquanto nao chegar a hora do alarme,
2010 ' continua mostrando
2100 IF A$<>X$ THEN 1900
2200 LOCATE 0,20:PRINT"ALARME!"
2300 BEEP:FOR X=1 TO 100:NEXT:GOTO 2300
```


SETADJST.BAS

```
1000 '
1100 ' EXEMPLO DE SET ADJUST
1200 '
1300 SCREEN 0:COLOR 15,4:CLS
1400 KEY OFF
1450 ' define um quadro na tela para auxiliar
1500 FOR X=0 TO 38
1600  LOCATE X,0:PRINT"#";:LOCATE X,23:PRINT"#";
1700 NEXT
1800 FOR Y=0 TO 22
1900  LOCATE 0,Y:PRINT"#";:LOCATE 39,Y:PRINT"#";
2000 NEXT
2100 LOCATE 5,10
2200 PRINT"USE AS SETAS PARA AJUSTAR! ";
2300 X=0:Y=0
2400 SET ADJUST (X,Y)
2500 A$=INPUT$(1)
2600 IF A$=CHR$(28) THEN 3100
2700 IF A$=CHR$(29) THEN 3300
2800 IF A$=CHR$(31) THEN 3700
2900 IF A$=CHR$(30) THEN 3500
3000 GOTO 2400
3100 X=X+1:IF X= 9 THEN X= 8
3200 GOTO 2400
3300 X=X-1:IF X=-8 THEN X=-7
3400 GOTO 2400
3500 Y=Y-1:IF Y=-8 THEN Y=-7
3600 GOTO 2400
3700 Y=Y+1:IF Y= 9 THEN Y= 8
3800 GOTO 2400
```

YJK.BAS

```
1000 '
1100 ' EXEMPLO DO PROBLEMA
1200 ' DESCRITO NO APENDICE B
1300 '
```

```
1350 ' executa a mesma sequencia para a
1355 ' screen 8, 11 e 12
1360 ' O comando utilizado e' o line ao inves do PSET
1366 ' apenas para melhor visualizacao no monitor.
1368 ' E' utilizada a coordenata (40,0) para melhorar
1369 ' a visao em certos televisores.
1400 SCREEN 8
1500 COLOR ,72,0:CLS
1600 LINE (40,0)-(40,140),61
1700 A$=INPUT$(1)
1800 SCREEN 11
1900 COLOR ,72,0:CLS
2000 LINE (40,0)-(40,140),61
2100 A$=INPUT$(1)
2200 SCREEN 12
2300 COLOR ,72,0:CLS
2400 LINE (40,0)-(40,140),61
2500 A$=INPUT$(1)
2600 GOTO 1400
```

SETSCROL.BAS

```
50 '
60 ' exemplo de scroll vertical
65 ' nao utilizando o SET SCROLL
70 '
100 FOR I=1 TO 8:D(I)=VAL(MID$("00022220",I,1))-1:NEXT
200 SCREEN 5
300 COLOR 15,14,13
400 CLS
500 OPEN "GRP:" AS #1
600 PSET (0,0),0
610 ' limpa os sprites
700 FOR I=0 TO 31:SPRITE$(0)=STRING$(32,0):NEXT
800 PSET (10,10):PRINT#1,"espaco extra"
900 COPY (0,0)-(255,43) TO (0,212),,PSET
1000 CLS
1100 PRINT#1, "scroll vertical"
1200 PRINT#1, "poderia ser feito tambem
1300 PRINT#1, "pelo comando SET SCROLL
```



```
1400 PRINT#1, "
1500 PRINT#1, "Use as teclas de cursor
1600 LINE (0,0)-(255,211),15,B
1700 P=0
1710 ' movimenta a tela pelas setas
1800 P=P-D(STICK(0))
1900 P=(P+256) MOD 256
2000 VDP(24)=P
2100 GOTO 1800
```

BLINK.BAS

```
50 '
55 ' exemplo de "blink" no modo de 80 colunas
60 '
100 SCREEN 0:WIDTH 80
200 COLOR 15,4
300 ADR=BASE(1)
310 ' limpa os atributos piscantes
400 FOR I=0 TO 2048/8
500   VPOKE ADR+I,0
600 NEXT
610 C1=1 :'cor alternada para caracter (frente)
615 C2=15:'cor alternada para caracter (fundo)
700 VDP(13)=C1*16 + C2
710 A=1:'(1/6 segundos) com cor "alternada"
720 B=3:'(3/6 segundos) com cor "normal"
730 'A e B de 0 a 15
731 'se A=0 entao caracter fica sempre com cor normal
732 'se B=0 e A nao for 0, caracter fica com cor alternada
733 'podendo simular atributo inverso ,por exemplo.
735 '
800 VDP(14)=A*16+B
900 PRINT"tecle caracteres a vontade... ":PRINT:PRINT
1000 K$=INPUT$(1)
1100 IF K$<CHR$(28) THEN 1500
1200 IF K$>" " THEN GOSUB 1600
1300 PRINTK$;
1400 GOTO 1000
1500 VDP(14)=0:END
```

```
1600 X=POS(0):Y=CSRLIN
1700 A=(Y*80+X)\8
1800 B=X MOD 8
1900 M=VAL("&b"+MID$("0000000010000000",8-B,8))
2000 VPOKE ADR+A,VPEEK (ADR+A) XOR M
2100 RETURN
```

SCROLHOR.BAS

```
45 '
50 ' exemplo de scroll horizontal
55 ' entre duas paginas
60 '
100 SCREEN 5
200 OPEN "GRP:" AS #1
300 COLOR 15,14,13
400 SET PAGE 0,1
500 CLS
600 SET PAGE 1,1
700 LINE (120,20)-(210,180),4
800 LINE -(30,150),4
900 LINE -(120,20),4
1000 PAINT (128,100),4,4
1100 LINE (0,0)-(255,211),4,B
1200 PSET (10,10),0
1300 PRINT#1,"ESTA E A PAGINA 1"
1400 SET PAGE 1,0
1500 LINE (50,50)-(210,120),10,BF
1600 LINE (0,0)-(255,211),10,B
1700 PSET (10,10),0
1800 PRINT#1,"ESTA E A PAGINA 0"
1900 SET PAGE 1,1
2000 PSET (10,185),0
2100 PRINT#1,"TECLE ALGO P/SCROLL"
2200 A$=INPUT$(1)
2300 FOR I=256 TO 0 STEP -1
2400 SET SCROLL I,0,1,1
2500 NEXT
2600 FOR I=0 TO 511
2700 SET SCROLL I,0,1,1
```



```
2800 NEXT
2900 FOR I=511 TO 256 STEP -1
3000 SET SCROLL I,0,1,1
3100 NEXT
3200 GOTO 3200
```

MATRIZES.BAS

```
100 '
101 ' EXEMPLO DE COPIA DE PARTES DA TELA PARA MATRIZES
102 '
150 ' calcula numero de bytes necessarios para
155 ' uma imagem de 50 por 50 pontos
200 X=((4 * 50 * (50+7))/8)+4
250 ' como um inteiro usa 2 bytes cada:
260 ' define uma matriz inteira com folga
300 DIM A%(INT(X/2))
400 SCREEN 7
500 COLOR 15,4,13:CLS
600 LINE (40,10)-(89,59),14,BF
700 LINE (40,10)-(89,59),1,B
800 LINE (45,15)-(84,38),15,BF
900 LINE (44,15)-(45,54),1,BF
1000 CIRCLE (65,25),10,6
1100 PAINT (65,25),6,6
1200 COPY (40,10)-(89,59) TO A%
1300 D=0
1400 I=RND(1)*(511-50):J=RND(1)*(212-50)
1500 COPY A%,D TO (I,J)
1600 D=(D+1) MOD 4
1700 GOTO 1400
```

BRILHO.BAS

```
100 '
200 ' exemplo de uso dos 32 valores de brilho (Y) possiveis
300 ' para uma cor escolhida (definindo-se J e K)
400 '
500 INPUT "Escolha o valor de j (-32 ate 31) ";J
```

```
600 INPUT "Escolha o valor de k (-32 ate 31) ";K
700 IF J<0 THEN J=63+J
800 IF K<0 THEN K=63+K
900 SCREEN 12
1000 COLOR 0,0,0:CLS
1100 FOR Y=0 TO 31
1200 P(0)=(Y*8)+(K AND 7)
1300 P(1)=(Y*8)+((K/8) AND 7)
1400 P(2)=(Y*8)+(J AND 7)
1500 P(3)=(Y*8)+((J/8) AND 7)
1600 FOR X=0 TO 3
1700 LINE (Y*8+X+4,0)-(Y*8+X+4,211),P(X)
1800 LINE (Y*8+X,0)-(Y*8+X,211),P(X)
1900 NEXT
2000 NEXT
2010 A$=INPUT$(1)
2020 SCREEN 0:COLOR 15,4:CLS
2030 GOTO 500
```

ENTRELAC.BAS

```
50 '
55 ' exemplo de telas alternadas e
60 ' modo de entrelace
65 '
100 SCREEN 5
200 OPEN "GRP:" AS #1
300 COLOR 15,14,13
400 SET PAGE 0,1
500 CLS
600 SET PAGE 1,1
700 LINE (120,20)-(210,180),4
800 LINE -( 30,150),4
900 LINE -(120,20),4
1000 PAINT (128,100),4,4
1100 LINE (0,0)-(255,211),4,B
1200 PSET (10,8),0
1300 PRINT#1,"ESTA E A PAGINA 1"
1400 SET PAGE 1,0
1500 LINE (50,50)-(210,120),10,BF
```



```
1600 LINE (0,0)-(255,211),10,B
1700 PSET (10,20),0
1800 PRINT#1,"ESTA E A PAGINA 0"
1900 SET PAGE 1,1
2000 PSET (10,185),0
2100 VDP(10)=VDP(10) AND &HF3
2200 VDP(14)=&H11
2300 A$=INPUT$(1)
2400 VDP(10)=VDP(10) OR &HC
2500 VDP(14)=0
2600 GOTO 2600
```

SCROLL.BAS

```
100 SCREEN 0:WIDTH 40
110 COLOR 15,4:CLS
120 PRINT"Exemplo de efeito gerado pelo
130 PRINT"scroll horizontal e vertical
140 PRINT"no screen 0
150 PRINT
160 FOR I=1 TO 100
170  PRINTUSING "##-";RND(1)*99;
180 NEXT
190 PRINT:PRINT
200 PRINT"tecle algo para o scroll
210 PRINT"horizontal ";
220 A$=INPUT$(1)
230 FOR I=1 TO 100
240  X=INT(RND(1)*7.99)
250  SET SCROLL X,0
260 NEXT
270 SET SCROLL 0,0
280 PRINT:PRINT
290 PRINT"tecle algo para o scroll
300 PRINT"vertical ";
310 A$=INPUT$(1)
320 FOR I=1 TO 100
330  Y=INT(RND(1)*7.99)
340  SET SCROLL 0,Y
350 NEXT
```

```
360 SET SCROLL 0,0
370 END
```

```
100 '
110 ' exemplo de uso da mapper
120 '
130 ' este programa permite que se
140 ' visualilze qualquer area de 256
150 ' bytes da
160 ' mapper interna do micro (256k)
170 '
180 ' O programa e apenas um exemplo
190 ' e assume que a mapper nao esta
200 ' sendo utilizada por outro
210 ' programa como o DDXDOS2 por ex.
220 '
230 '
240 CLEAR 100,&HD000
250 DEFINT A-Z
260 RESTORE:ADR = &HD000
270 READ AS:IF AS< > "FIM" THEN POKE
ADR,VAL("&H" + AS):ADR = ADR + 1:GOTO 270
280 DEFUSR0 = &HD001
285 ' calcula o slot da mapper
290 SL = INT(INP(&HA8)/64)
300 POKE &HD000,SL
310 IF PEEK(&HFCC1 + SL) < 128 THEN 360
320 X = USR0(-1)
330 J = INT(X/64) XOR 3
335 ' Mapper em slot expandido
340 SL = 128 + SL + J*4
350 POKE &HD000,SL.
360 SCREEN 0:WIDTH 80
370 COLOR 15,1:CLS
380 INPUT "Endereco alto (000-400) ";AS
390 A = VAL("&h" + AS)
400 H = A MOD 64
410 BANK = INT (A/64)
```



```
420 OUT &HFC,BANK
430 ADR = H*256
440 FOR J = 0 TO 15
450  PRINTRIGHT$("0000" + HEX$(ADR),4)" - ";
460  FOR X = 0 TO 15
470   PRINT RIGHT$(HEX$(USR0(ADR + X) + 256),2)" ";
480  NEXT
490  PRINT
500  ADR = ADR + 16
510 NEXT
520 PRINT:GOTO 380
530 DATA 00
540 DATA 2A,F8,F7,3A,00,D0,CD,0C,00
550 DATA 32,F8,F7,AF,32,F9,F7,C9
560 DATA FIM
```
