INTERFACE DE SAÍDA COM DISPOSITIVOS GRÁFICOS

por

Sílvia Delgado Olabarriaga *
Márcio Serolli Pinho **
João Luiz Dihl Comba **

RP no. 79 SETEMBRO 1987

Nota técnica do projeto "Banco de Dados e Ferramentas para CAD de Sistemas Digitais"

Orientação : Profa. Carla M. Dal Sasso-Freitas **

* trabalho realizado com apoio da FINEP
** trabalho realizado com apoio do CNPQ

UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
PÓS GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO
Av. Osvaldo Aranha, 99
90001 - Porto Alegre - RS - BRASIL
Telefone : (0512) 21-8499
Telex : (051) 2680 - CCUF BR

Correspondência : UFRGS - CPGCC
Caixa Postal 1501
90001 - Porto Alegre - RS - Brasil

RESUMO

O relatório apresenta uma interface de saída com dispositivos gráficos constituída por um conjunto de rotinas dependentes de hardware. O objetivo principal deste conjunto de rotinas é servir de base para o pacote gráfico (independente de dispositivo) do AMPLO. A interface implementa funções para geração de pontos, linhas, retângulos, áreas e texto, com atributos específicos de cada ente, bem como funções de controle e interrogação de características dos dispositivos.

PALAVRAS-CHAVE : computação gráfica, interface de saída, funções gráficas.

ABSTRACT

This report presents a device-dependent interface constituted by a set of graphical output functions. The main goal of these functions is to implement the lower layer of the graphical software used in AMPLO. On the top of this layer there is a graphical, device-independent package. The functions allow the exhibition of points, lines, rectangles, areas and text, each one with its specific attributes. There are also control and device interrogation functions.

KEYWORDS : computer graphics, output interface, graphical functions.

SUMARIO

LISTA DE FIGURAS

1. INTRODUÇAO

Um dos aspectos principais do AMPLO (Ambiente para Projeto Lógico de Sistemas Digitais) [WAG 86b,WAG 87b] é a preocupação com a interface homem-máquina. Todas as ferramentas do ambiente serão ativadas a partir de uma interface de alto-nível, denominada LAGO (Linguagem de Acesso Global ao AMPLO), onde recursos gráficos deverão ser utilizados para se atingir maior eficiência. Desta interface poderão ser ativados os compiladores e editores (gráficos) das linguagens de descrição de hardware [WAG 87a], os simuladores associados a cada nível de descrição e realizadas consultas á base de dados. Cada uma destas ferramentas interage com o projetista segundo uma linguagem de comandos. No caso dos compiladores e das consultas á base de dados, esta linguagem é textual, mas implementada sob a forma de cardápios. No caso dos simuladores, está prevista a utilização de representação gráfica para certos parâmetros de entrada e saída. No caso dos editores gráficos, é óbvia a utilização de entes gráficos, já que estes editores implementam as ferramentas de síntese dos sistemas digitais através de suas representações gráficas, equivalentes ás textuais [WAG 86a], analisadas pelos diferentes compiladores.

A necessidade de recursos gráficos por parte das diversas ferramentas no ambiente e o fato da linguagem escolhida para sua implementação (MicroSoft C, versão 4.0) não contar com biblioteca de rotinas gráficas originaram a definição de um pacote gráfico, independente de dispositivo, contendo muitas funções que deveriam ser implementadas dentro das ferramentas. Para adaptar este pacote ao hardware existente atualmente, optou-se por construir a "interface de saída" (objeto deste relatório) como uma camada dependente de dispositivo, que pode ser usada isoladamente, quando não é necessária a independência de dispositivo.

As finalidades desta interface são múltiplas :

a) fornecer um conjunto de funções básicas de saída em dispositivos gráficos, a serem usadas por todas as ferramentas de AMPLO;

b) fornecer um ambiente padronizado para realização de saída em dispositivos gráficos, sejam eles vídeos ou traçadores;

c) permitir que um programa utilize diversos dispositivos gráficos, sem que isto signifique alterações em seu código-fonte;

d) facilitar o transporte de software gráfico entre instalações.

2. O QUE E' "IS"

"IS" e' um conjunto de funções que possibilitam o acesso á superfície de exibição de um dispositivo gráfico. Estas funções trabalham com uma caneta virtual que, ao ser deslocada, pode desenhar linhas e pontos, preencher áreas e escrever caracteres sobre a superfície de exibição.

A execução destas funções é regida pelo "estado corrente dos atributos", que pode ser configurado conforme as necessidades do programa aplicativo. Este estado è composto de variáveis globais que indicam cor, posição, estilo de linha, etc. Inclui também um indicativo de erro que permite ao aplicativo opcionalmente verificar a condição de têrmino da função chamada.

As funções da IS são classificadas em :

a) controle : permitem a ativação e desativação do dispositivo gráfico;

b) alteração de atributos : permitem a configuração do estado da IS conforme necessidades do programador;

c) saída : realizam a saída propriamente dita na superfície de exibição;

d) interrogação : permitem a obtenção de informações dependentes do dispositivo gráfico conectado.

As funções de saída utilizam o ambiente configurado pelas demais. As coordenadas das posições da caneta virtual são especificadas em "coordenadas de dispositivo".

3. ESTADO DA IS

O estado da IS é determinado pelos atributos correntes de linha, área, texto, etc, que podem ser alterados pelas funções descritas no item 4.2.

Os atributos são armazenados em variáveis globais, descritas a seguir :

a) para linhas, retângulos, pontos :
   cor
   estilo (tipo do traçado, p. ex: pontilhado, tracejado, etc.)

b) para áreas :
   cor
   padrão (forma de preenchimento, p. ex: liso, quadriculádo, etc.)

c) para texto :
   cor
   tipo de letra (ex: itálico, gótico, etc.)
   fator de escala (tamanho do caractere)
   conjunto de caracteres ativo (pode-se trabalhar alternadamente com dois conjuntos de caracteres)

d) posição corrente :
   (x,y) - em coordenadas de dispositivo, indica a posição da caneta virtual na superfície de exibição.

e) dispositivo de saída ativo;
   vídeo ou traçador

Os tipos destas variáveis, assim como seus valores válidos em determinada implantação são definidos no arquivo "ISTIPOS.H", descrito na seção 5.1.

4. FUNÇÕES DA IS

O nome de todas as funções da IS inicia por "isnn_", seguido de um string que indica o procedimento executado. Normalmente recebem parâmetros, cujos tipos são especificados no capítulo 6.

4.1 Controle

Têm por objetivo ativar ou desativar o hardware do dispositivo de saída gráfica.

a) is20_inic - inicializa hardware do dispositivo de saída ativo e seta os atributos com valores pré-definidos;

b) is28_fim - desativa dispositivo de saída gráfica.

4.2 Alteração de atributos

Permitem ao aplicativo controlar o valor corrente dos atributos que definem o estado da IS. Sua execução pode ter têrmino :

- normal : o atributo é alterado para o valor indicado;
- por erro : o atributo permanece inalterado, e a variável global "iserro" contêm o código do erro detectado.

As funções de alteração de atributos são as seguintes :

a) is01_setcorlin - altera cor usada para linhas, pontos e retângulos;

b) is02_setestlin - altera estilo usado para desenhar linhas e retângulos;

c) is03_setcorarea - altera cor usada para preenchimento de áreas;

d) is04_setpadarea - altera padrão usado para preenchimento de áreas;

e) is05_setcortx - altera cor usada para escrita de caracteres;

f) is06_setipotx - altera tipo dos caracteres;

g) is07_setfattx - altera fator de escala dos caracteres usados para escrita de um texto;

h) is08_setcjtx - altera conjunto de caracteres ativo;

i) is09_posic - altera posição corrente para (x,y);

j) is10_ativadisp - ativa um dispositivo de saída (tela ou papel);

4.3 Saída

Permitem a exibição de linhas, pontos, textos e áreas na superfície de exibição do dispositivo gráfico ativo. Seu funcionamento é baseado nos parâmetros que recebem e nos atributos correntes da IS. Em alguns casos, sua execução pode alterar o estado da IS (mudança da posição corrente, retorno de erro, etc.)

São as seguintes as funções de saída :

a) is21_linha (x, y) - desenha uma linha da posição corrente até (x,y), usando cor e estilo de linha correntes. Altera posição corrente para (x,y);

b) is22_linhah (x) - traça uma linha horizontal da posição corrente até (x,y corrente), usando cor e estilo de linha correntes. Altera posição corrente para (x, y corrente);

c) is23_linhav (y) - traça uma linha vertical da posição corrente até (x corrente, y), segundo cor e estilo correntes. Altera posição corrente para (x corrente, y);

d) is24_retang (x, y) - traça o contorno do retângulo cuja diagonal vai da posição corrente até (x,y). Usa cor e estilo de linha correntes. Não altera posição corrente;

e) is25_ponto - desenha um ponto na cor e posição correntes. Não altera posição corrente;

f) is26_area (x,y) - preenche a área retangular que tem diagonal da posição corrente até (x,y), usando cor e padrão de área correntes. Não altera posição corrente;

g) is27_texto (string) - escreve um string na posição corrente, utilizando cor e conjunto de caracteres ativo (tipo e fator de texto correntes). Não altera posição corrente;

4.4 Interrogação

Permitem ao aplicativo a aquisição de informações sobre o dispositivo ativo.de saída gráfica.

a) is30_coordisp - retorna as coordenadas máximas e mínimas do dispositivo. O ponto de x e y mínimos corresponde ao canto superior esquerdo do monitor;

b) is31_aspecto - devolve a máxima coordenada normalizada de dispositivo nos eixos x e y;

c) is32_tamcarac - retorna o tamanho da matriz que define os caracteres para o fator de texto corrente (em coordenadas de dispositivo).

5. COMO USAR A IS EM UM PROGRAMA APLICATIVO

O aplicativo deve ser codificado na linguagem C e usar o compilador da Microsoft (versão 4.0 ou mais recente). Se for necessário utilizar outra linguagem, fica a cargo do programador a implantação dos procedimentos adequados á compatibilização da chamadas de rotinas e compartilhamento de dados.

Alguns procedimentos devem ser sempre adotados pelo aplicativo :

a) inclusão de arquivos-cabeçalho que contêm definições de tipos e constantes usadas na passagem de parâmetros para rotinas da IS. São eles "ISTIPOS.H" (ver seção 5.1) e "ISERROS.H" (ver seção 5.2);

b) inicialização da IS, através da função "is20_inic";

c) programação dos atributos de texto, linha e área através das funções descritas na seção 4.2;

d) exibição propriamente dita, através das funções descritas na seção 4.3;

e) desativação da IS, através da função "is28_fim".

Além disto, é importante certificar-se de que os tipos dos parâmetros, bem como seus valores, estão corretos. Para maior segurança, testar sempre o código de erro retornado pelas funcoes; o aplicativo só deve prosseguir se o indicativo de erro valer zero (ver seção 5.2).

Por fim, é necessário gerar um programa executável, ligando-o á biblioteca que contêm as funções da IS. De maneira geral, pode ser obtido através do seguinte comando :

LINK aplicativo, aplicativo,,ISGRAFS;

-> "aplicativo.obj" é o nome do arquivo que contêm o programa-objeto correspondente ao aplicativo.
-> "aplicativo.exe" é o nome do arquivo que contêm o programa executável.

Deve-se prestar atenção, também, ao modelo de dados e código (grande ou pequeno) usado pelo aplicativo e IS; os modelos devem ser compatíveis.

Para executar o aplicativo, faz-se necessária a presença de arquivos no diretório corrente. Estes arquivos contêm as matrizes que definem os caracteres que serão exibidos pela rotina de texto da IS. São eles :

- "COM4X6.TXT" (necessário apenas se fator 0 for usado);
- "COM8X8.TXT" (presença obrigatória)

5.1 Tipos de dados usados pela IS

Os tipos de dados usados pela IS são definidos no arquivo "ISTIPOS.H", assim como os valores que podem ser recebidos como parâmetro pelas funções da interface.

- COR = cor de linha, texto e área. São válidos apenas os seguintes valores :
  - BRANCO
  - PRETO
  - ~~EXCLUSIVO~~ INVERSO (ou exclusivo com o conteúdo anteriormente exibido na tela)

- ESTILO = estilo de linha. São válidos apenas os seguintes valores :
  - CHESIMPL = linha cheia, simples
  - TRACOS = linha tracejada
  - PONTOS = linha pontilhada
  - PTOTRACO = linha feita de traços e pontos
  - CHEDUP = linha cheia, dupla
  - ESPESSA = linha cheia, grossa

Os estilos TRACOS, PONTOS, PTOTRACO, CHEDUP e ESPESSA podem ser usados apenas em linhas horizontais ou verticais; linhas de orientação qualquer só podem ser do tipo CHESIMPL.

- PADRAO = padrão para preenchimento de área. Valores válidos : apenas LISO (preenchimento uniforme).

- TIPO = tipo de letra. Valores válidos : apenas COMUM, que corresponde ao tipo normal apresentado pelos monitores de vídeo.

- FATOR = fator de escala para caracteres. Valores válidos :
  - 0 (zero) = caracteres 4 x 6
  - 1 (um) = caracteres 8 x 8

- DISP = tipo de dispositivo de saída. Valores válidos são apenas TELA ou PAPEL.

E' recomendado o uso de operadores "cast" para passagem de parâmetros para as funções da IS.

5.2 Erros detectados pela IS

Os erros detectados pela IS dizem respeito basicamente a valores invàlidos recebidos como parâmetro ou problemas de acesso a disco.

A ocorrência de um erro è detectada pelo aplicativo com a presença de um valor diferente de zero na variável "iserro". Esta variável è global e pode ser acessada pelo aplicativo se esse incluir o cabeçalho "ISERROS.H". O còdigo do erro detectado è nela armazenado e pode ser um dos seguintes :

- CORINV = cor invàlida ;
  (diferente de PRETO, BRANCO e EXCLUSIVO)

- TIPOINV = tipo de letra invàlido;
  (diferente de COMUM).

- FATINV = fator de textó invàlido;
  (diferente de 0 e de 1)

- CONJINV = conjunto de caracteres invàlido;
  (diferente de 0 e de 1)

- ERRDISCO = erro de acesso a disco durante a carga de um arquivo contendo caracteres;

- ESTINV = estilo de linha invàlido;
  (diferente dos mencionados na seçào 5.1)

- PADINV = padràO de àrea invàlido;
  (diferente de LISO)

- COORDINV = coordenada de dispositivo invàlida;
  (em x ou y)

- DISPINV = dispositivo de saída invàlido;
  (atualmente, apenas TELA)

- LININV = tentativa de desenhar uma linha de orientação qualquer com estilo diferente de CHESIMPL.

A variável "iserro" sempre è alterada pelas funçòes da IS : em caso de erro, recebe o còdigo apropriado; em caso contràrio, recebe zero.

5.3 Considerações gerais sobre o uso das funções da IS

As rotinas da IS funcionam conforme o estado corrente dos atributos. Uma vez que estes tenham sido programados, basta utilizar as funções de saída desejadas. As funções de alteração de atributos (seção 4.2) devem ser chamadas apenas quando for necessária trocar o valor dos mesmos.

A seguir serão feitos breves comentários sobre o ambiente de funcionamento da IS.

5.3.1 Exibição de linhas, retângulos e áreas

E' necessário programar os valores dos atributos cor (de linha e de área), estilo de linha e padrão de área. Depois disto, basta chamar as funções correspondentes á exibição destes elementos.

5.3.2 Exibição de texto

A cada tipo e fator de caractere corresponde um arquivo em disco que deve ser carregado na memória sempre que o aplicativo executar a função que altera tais atributos. A fim de reduzir a quantidade de acessos a disco decorrente desta característica, a IS permite que o aplicativo utilize dois conjuntos de caracteres alternadamente. Ambos estão residentes na memória simultaneamente e podem ser programados e selecionados conforme as necessidades da aplicação. Apenas os atributos do conjunto de caracteres ativo podem ser alterados, sendo mantidos mesmo durante o período em que este não estiver ativo. E' importante salientar que a cor não é um atributo associado ao conjunto ativo, devendo ser reprogramada sempre que necessário.

Outra observação importante deve ser feita a respeito da forma de exibição dos caracteres : apenas os pixels que devem ser "ligados" são efetivamente desenhados na cor corrente; os demais pixels da área em que o texto é exibido permanecem inalterados. Sendo assim, caracteres pretos aparecem apenas se exibidos sobre um fundo previamente preenchido com outra cor. Além disto, é importante ter em mente o significado do caractere "espaço" (0x20) neste ambiente : corresponde á ausência de pixels ligados. Desta forma, espaços não podem ser usados para "apagar" um string que está na tela, pois este caractere sempre deixa o fundo como está.

```
/*
        NOME : EXEMPLO.C
        PROGRAMA COM EXEMPLO DE USO DA IS
*/

#include "istipos.h"
#include "iserros.h"

main()
  {
  is20_inic();                          /* inicializa pacote */
  if (iserro)
     {
     printf("Erro na inicializacao da interface de saida (%d)\n",
             iserro);
     getch();
     is28_fim();
     }
  else
     {
                                /* retangulo de linhas simples, cheias */
     is09_posic(10,10);
     is24_retang(620,190);
                                 /* retangulo de linhas tracejadas */
     is02_setestlin(TRACOS);
     is09_posic(40,180);
     is24_retang(590,20);
                                /* retangulo de linhas pontilhadas */
     is02_setestlin(PONTOS);
     is09_posic(550,160);
     is24_retang(80,40);
                                 /* retangulo de linhas "traco-ponto" */
     is02_setestlin(PTOTRACO);
     is09_posic(500,50);
     is24_retang(130,150);
                                /* retangulo de linhas cheias, duplas */
     is02_setestlin(CHEDUP);
     is09_posic(150,60);
     is24_retang(480,140);
                                   /* area preenchida com branco */
     is03_setcorarea(BRANCO);
     is09_posic(200,70);
     is26_area(430,130);
                                /* texto no centro, em preto, 8 X 8 */
     is05_setcortx(PRETO);
     is09_posic(290,90);
     is27_texto("ISGRAF");
                                     /* texto em preto, 4 X 6 */
     is08_setcjtx(1);
     is07_setfattx(0);
     is09_posic(290,110);
     is27_texto("EXEMPLO");
```

```
            /* duas linhas verticais, simples, com cor INVERSO */
    is01_setcorlin(INVERSO);
    is02_setestlin(CHESIMPL);
    is09_posic(220,10);
    is23_linhav(190);
    is09_posic(410,10);
    is23_linhav(190);
                                    /* mensagem em branco, 8 X 8 */
    is08_setcjtx(0);
    is05_setcortx(BRANCO);
    is09_posic(400,198);
    is27_texto("Return para terminar");

    getch();

    is28_fim();                              /* encerra uso da IS */
    }
}
```

6. MANUAL DE REFERENCIA DA IS

A seguir são apresentadas as informações relevantes para utilização das rotinas da IS, organizadas sob os seguintes tópicos :

a) arquivos que devem ser incluídos no aplicativo para acessar constantes e tipos usados na chamada da função;

b) protótipo da função, indicando o tipo do valor retornado, seu nome e parâmetros recebidos;

c) breve descrição contendo funcionamento e eventuais informações retornadas;

d) observações gerais.

is01_setcorlin

RESUMO

```
#include "istipos.h"
#include "iserros.h"

is01_setcorlin (cor_lin)
COR cor_lin ;
```

DESCRIÇAO

Altera o atributo de cor de linha para o valor recebido como parâmetro .

Atribui o código de erro á variável global "iserro" :

- ZERO : execução OK , cor de linha alterada .
- CORINV : cor invâlida , atributo inalterado .

OBSERVAÇAO

Para informações sobre os atributos de cor válidos, dirija-se ao item 5.1.

is02_setestlin

RESUMO

```
#include "istipos.h"
#include "iserros.h"

is02_setestlin (est_lin)
ESTILO est_lin ;
```

DESCRIÇAO

Altera o atributo de estilo de linha para o valor recebido como parâmetro .

Atribui o código de erro á variável global "iserro" :

- ZERO : execução OK , estilo de linha alterado .
- ESTINV : estilo invâlido , atributo inalterado .

OBSERVAÇAO

Para informações sobre os atributos de estilo vâlidos, dirija-se ao item 5.1.

is03_setcorarea

RESUMO

```
#include "istipos.h"
#include "iserros.h"

is03_setcorarea (cor_area)
COR cor_area ;
```

DESCRIÇAO

Altera o atributo de cor de àreà para o valor recebido como parâmetro .

Atribui o còdigo de erro à variàvel global "iserro" :

- ZERO : execução OK , cor de àrea alterada .
- CORINV : cor invàlida , atributo inalterado .

OBSERVAÇAO

Para informaçðes sobre os atributos de cor vàlidos, dirija-se ao item 5.1.

is04_setpadarea

RESUMO

```
#include "istipos.h"
#include "iserros.h"

is04_setpadarea (pad_area)
PADRAO pad_area ;
```

DESCRIÇÃO

Altera o atributo de padrão de área para o valor recebido como parâmetro .

Atribui o código de erro à variável global "iserro" :

- ZERO : execução OK , padrão alterado .
- PADINV : padrão invàlido , atributo inalterado .

OBSERVAÇÃO

Para informações sobre os atributos de padrão válidos, dirija-se ao item 5.1.

is05_setcortx

RESUMO

```
#include "istipos.h"
#include "iserros.h"

is05_setcortx ( cor_tx )
COR cor_tx;
```

DESCRIÇÃO

Altera o atributo "cor de textò" para o valor recebido como parâmetro. São permitidas apenas as cores BRANCO e PRETO.

Atribui o côdigo de erro à variável global "iserro" :

- ZERO : execução OK, cor de texto alterada;
- CORINV : cor invàlida, atributo inalterado.

is06_setipotx

RESUMO

```
#include "istipos.h"
#include "iserros.h"

is06_setipotx ( tipo_tx )
TIPO tipo_tx;
```

DESCRIÇAO

Altera o atributo "tipo de letra" do conjunto de caracteres ativo para o valor recebido como parâmetro. Automaticamente carrega na memória o arquivo contendo a descrição dos caracteres para o novo tipo selecionado, no tamanho indicado pelo atributo "fator de texto".

Atribui o código de erro à variavel global "iserro" :

- ZERO : execução OK, tipo de letra do conjunto ativo foi alterado;
- TIPOINV : tipo invàlido, atributo inalterado.
- ERRDISCO : erro de acesso a disco durante a carga do arquivo de caracteres. Neste caso, as matrizes que definem os caracteres podem ser destruídas.

OBSERVAÇAO

Para informações sobre os tipos de letra vàlidos, dirija-se ao item 5.1.

is07_setfattx

RESUMO

```
#include "istipos.h"
#include "iserros.h"

is07_setfattx ( fator_tx )
FATOR fator_tx;
```

DESCRIÇAO

Altera o atributo "fator de texto" do conjunto de caracteres ativo para o valor recebido como parâmetro. Automaticamente carrega na memória o arquivo contendo a descrição dos caracteres correspondente ao tipo de letra corrente, no novo tamanho selecionado.

Atribui o código de erro á variável global "iserro" :

- ZERO : execução OK, fator de texto do conjunto ativo foi alterado;
- FATINV : fator invâlido, atributo inalterado;
- ERRDISCO : erro de acesso a disco durante a carga do arquivo de caracteres. Neste caso, as matrizes que definem os caracteres podem ser destruídas.

OBSERVAÇAO

O máximo fator de texto permitido pode variar conforme o conjunto de caracteres que está ativo :

- conjunto 0 : máximo fator = 1 (matriz de 8 X 8);

- conjunto 1 : por enquanto, idem conjunto 0.

is08_setcjtx

RESUMO

```
#include "istipos.h"
#include "iserros.h"

is08_setcjtx ( conj_tx )
CONJ conj_tx;
```

DESCRIÇÃO

Troca o conjunto ativo de caracteres para escrita de texto. Existem apenas dois conjuntos de texto, 0 e 1, que podem ser programados e usados alternadamente. Os atributos do novo conjunto de caracteres são aqueles programados anteriormente, na última vez em que tal conjunto foi utilizado. Não há carga de arquivos em disco quando esta função é executada.

Atribui o código de erro à variável global "iserro" :

- ZERO : execução OK, conjunto ativo de caracteres foi alterado;
- CONJINV : identificador de conjunto invâlido, conjunto inalterado.

is09_posic

RESUMO

```
#include "istipos.h"
#include "iserros.h"

is09_posic (x, y)
int  x, y ;
```

DESCRIÇÃO

Altera a variável global de posicão corrente para (x, y). Faz a verificação dos valores de x e y .conforme os valores permitidos pelo dispositivo e atribui o còdigo de erro á variável global "iserro" :

- ZERO : execução OK , posição corrente alterada.
- COORDINV : coordenada passada como parâmetro é invàlida (não satisfaz os valores màximos ou mínimos).

OBSERVAÇÃO

Para informações sobre valores de coordenadas vàlidos, ver função "is30_coordisp".

is10_ativadisp

RESUMO

```
#include "istipos.h"
#include "iserros.h"

is10_ativadisp (disp_saida)
DISP disp_saida ;
```

DESCRIÇAO

Altera o atributo de dispositivo de saída ativo para o valor recebido como parâmetro.

Atribui o código de erro à variável global "iserro" :

- ZERO : execução OK.
- DISPINV : dispositivo de saída invâlido , atributo inalterado .

OBSERVAÇAO

Para verificar os dispositivos de saída válidos, dirija-se ao item 5.1.

is20_inic

RESUMO

```
#include "istipos.h"

is20_inic ()
```

DESCRIÇAO

Inicializa e ativa os atributos da interface com os valores "default" . Inicializa os atributos de texto e passa a tela para o modo gràfico.

OBSERVAÇAO

Os valores "default" sào os seguintes :

- cor de linha = BRANCO .
- cor de area = BRANCO .
- cor de texto = BRANCO .
- estilo de linha = CHESIMPL .
- padràO de area = LISO .
- dispositivo ativo = TELA .
- posiçàO corrente = (0,0) .
- conjunto 0 e 1 = tipo de texto = COMUM , fator de texto = 1 .
- conjunto ativo = 0

is21_linha

RESUMO

```
#include "istipos.h"
#include "iserros.h"

is21_linha (x,y)
int  x,y ;
```

DESCRIÇAO

Desenha uma linha da posição corrente atê o ponto (x,y) passado como parâmetro , segundo a cor e estilo de linha correntes. Chama uma rotina escrita êm linguagem de montagem que realmente faz o traçado da linha . Indica-se o seu uso para o traçado de linhas nào horizontais ou verticais, pois essas podem ser desenhadas de forma mais eficiente pelas rotinas is22_linhah e is23_linhav. Ao final da execução, as variâveis globais pc.x e pc.y conterào x e y respectivamente .

Atribui o côdigo de erro â variâvel global "iserro" :

- ZERO : execução OK, linha traçada.
- COORDINV : coordenada passada como parâmetro è invâlida (nào satisfaz os valores màximos ou mînimos).

OBSERVAÇAO

Para informações sobre valores vâlidos para coordenadas em x e y, consulte a função "is30_coordisp".

is22_linhah

RESUMO

```
#include "istipos.h"
#include "iserros.h"

is22_linhah (x)
int  x ;
```

DESCRIÇAO

Traça uma linha horizontal da posição corrente até (x, pc.y) segundo a cor e estilo de linha correntes . No final a variável global pc.x assume o valor de x.

Atribui o código de erro á variável global "iserro" :

- ZERO : execução OK , linha desenhada.
- COORDINV : coordenada passada como parâmetro é invâlida (não satisfaz os valores máximos ou minimos).

OBSERVAÇAO

Para informações sobre valores válidos para coordenadas em x e y, consulte a função "is30_coordisp".

is23_linhav

RESUMO

```
#include "istipos.h"
#include "iserros.h"

is23_linhav (y)
int  y ;
```

DESCRIÇÃO

Traça uma linha vertical da posição corrente até (pc.x, y) segundo a cor e estilo de linha correntes . No final a variável global pc.y assume o valor de y.

Atribui o código de erro á variável global "iserro" :

- ZERO : execução OK, linha desenhada.
- COORDINV : coordenada passada como parâmetro é invàlida (não satisfaz os valores màximos ou mínimos).

OBSERVAÇÃO

Para informações sobre valores válidos para coordenadas em x e y, consulte a função "is30_coordisp".

is24_retang

RESUMO

```
#include "istipos.h"
#include "iserros.h"

is24_retang (x, y)
int  x, y ;
```

DESCRIÇÃO

Traça o contorno do retângulo cuja diagonal é dada pela posição corrente e o ponto (x, y) passado como parâmetro , na cor e estilo correntes. Os valores das variáveis globais pc.x e pc.y permanecem inalterados apos a execução da rotina .

Atribui o código de erro à variável global "iserro" :

- ZERO : execução OK, retângulo traçado.
- COORDINV : coordenada passada como parâmetro é invàlida (não satisfaz os valores màximos ou mínimos).

OBSERVAÇÃO

Para informações sobre valores válidos para coordenadas em x e y, consulte a função "is30_coordisp".

is25_ponto

RESUMO

is25_ponto ()

DESCRIÇAO

Desenha um ponto na posição corrente, na cor de linha corrente.

is26_area

RESUMO

```
#include "istipos.h"
#include "iserros.h"

is26_area (x, y)
int  x, y ;
```

DESCRIÇAO

Preenche a área formada entre a posição corrente e o ponto (x,y) recebido como parâmetro , na cor e padrão de área correntes. Ao final da execução, as variáveis globais pc.x e pc.y conterão os valores de x e y respectivamente.

Atribui o código de erro á variável global "iserro" :

- ZERO : execução OK, área preenchida.
- COORDINV : coordenada passada como parâmetro é invâlida (não satisfaz os valores máximos ou mínimos).

OBSERVAÇAO

Para informações sobre valores válidos para coordenadas em x e y, consulte a função "is30_coordisp".

is27_texto

RESUMO

```
is27_texto ( s )
char *s;
```

DESCRIÇAO

Exibe o string s na posição e cor de texto correntes, usando tipo e fator definido para o conjunto de caracteres ativo. Os caracteres contidos em s devem ter código entre "espaço" (0x20) e "~" (0x7e); o string deve ser sempre terminado por um caractere nulo ('\0').

O posicionamento do string em relação á posição corrente da caneta virtual é feito da seguinte forma : a base do string é colocada um pixel abaixo da posição corrente.

Esta função não altera as variáveis globais "iserro", nem a posição corrente da caneta virtual.

is28_fim

RESUMO

```
is28_fim ()
```

DESCRIÇAO

Modifica a tela para o modo alfanumérico.

is30_coordisp

RESUMO

```
is30_coordisp ( xmin, xmax, ymin, ymax )
int * xmin;
int * xmax;
int * ymin;
int * ymax;
```

DESCRIÇÃO

Retorna nas variáveis recebidas como parâmetro os valores máximos e mínimos das coordenadas do dispositivo de exibição ativo. O ponto (xmin, ymin) corresponde ao canto superior esquerdo da superfície de exibição.

Na placa CGA : xmin = 0, xmax = 639
ymin = 0, ymax = 199

Esta função não altera o estado da IS.

is31_aspecto

RESUMO

```
is31_aspecto ( xmax, ymax )
float * xmax;
float * ymax;
```

DESCRIÇÃO

Retorna nas variáveis recebidas como parâmetro a relação de aspecto da superfície de exibição ativa. A maior dimensão (normalmente x) é fixada como tendo comprimento 1.0 (um), sendo o comprimento da outra determinado em relação á primeira.

Na placa CGA : xmax = 1.0
ymax = 0.8

is32_tamcarac

RESUMO

```
is32_tamcarac ( tx, ty )
int *tx;
int *ty;
```

DESCRIÇAO

Retorna nas variâveis recebidas como parâmetro o tamanho da matriz que define os caracteres para o fator de texto corrente :

- tx = nûmero de pixels em x (nro. de colunas);
- ty = nûmero de pixels em y (nro. de linhas);

7. IMPLANTAÇÃO DA IS

As funções que implementam as primitivas de exibição foram escritas parcialmente na linguagem C e na linguagem de montagem do microprocessador INTEL 8086. Todas as rotinas que podem ser chamadas diretamente do aplicativo são escritas em C e têm número inferior a 40 (isnn_... , onde nn<40). As rotinas de número superior a 40 foram desenvolvidas diretamente em linguagem de montagem para obtenção de máximo desempenho.

A seguir, apresentaremos apenas os tópicos mais relevantes sobre a implantação de tais rotinas, enfatizando algoritmos e estruturas de dados empregadas. Para maiores detalhes, consultar diretamente os programas-fonte.

7.1 Funções de ponto e linhas

7.1.1 is41_initgraph

Coloca o vídeo em modo gráfico de alta resolução, com 640 pontos horizontais por 200 pontos verticais. A inicialização é feita através da interrupção 16 do BIOS, usando os seguintes parâmetros :

AH = 0 (rotinas de vídeo)
AL = 06 (seleção do modo de vídeo)

7.1.2 is42_modotexto

Coloca o vídeo em modo texto com 24 linhas por 80 colunas. A inicialização é feita através da interrupção 16 do BIOS usando os seguintes parâmetros :

AH = 0 (rotinas de vídeo)
AL = 2 (seleção do modo de vídeo)

7.1.3 Cálculo de endereço do ponto a ser traçado

Para obtermos o endereço de um ponto na página gráfica da placa CGA, ou seja, seu deslocamento a partir do início desta página, devemos considerar a organização da memória gráfica.

A memória gráfica inicia em B800H e cada byte armazena oito pontos - bit em 1 corresponde a ponto aceso ; bit em 0, a ponto apagado. Os bytes de uma mesma linha estão armazenados em posições contíguas de memória.

A memória gráfica está dividida em dois blocos : linhas pares e linhas ímpares. O primeiro bloco inicia no deslocamento 0 e contém as linhas pares na ordem 0,2,...,196, 198. O segundo contém as linhas ímpares 1,3,5,...,197,199, e inicia no deslocamento 2000H a partir do início da página gráfica.

A fórmula de cálculo do endereço do ponto de coordenadas (x,y) é a seguinte:

END := 8192 * (x mod 2) + 80 * (y div 2) + (x div 8)

De posse do endereço do ponto, é simples obter o endereço dos pontos adjacentes a ele:

- ponto á direita : se ambos estiverem no mesmo byte, o endereço é o mesmo; senão, o novo endereço é END + 1;

- ponto á esquerda : se ambos estiverem no mesmo byte, então o endereço é o mesmo; senão, o novo endereço é END-1;

- ponto acima : devemos considerar dois casos :
  * linha original é par : o novo endereço é END + 8112 (8192-80)
  * linha original é ímpar : o novo endereço é END - 8192

- ponto abaixo : devemos considerar dois casos:
  * linha original é par : o novo endereço é END + 8192
  * linha original é ímpar : o novo endereço é END - 8112

Para traçar linhas verticais e quaisquer são usados dois incrementos : o primeiro em função do primeiro ponto da linha e o segundo em função do próximo ponto da linha, como por exemplo:

- numa linha em que YI é par e YI < YF (está descendo na tela), o primeiro incremento será 8192 (y1 = y + 1) e o segundo incremento será -8112 (y2 = y1 + 1); durante o traçado da linha os dois incrementos são trocados entre si e a cada passo um deles é somado ao valor de END do último ponto traçado.

A tabela completa dos incrementos è a seguinte :

```
para YI < YF (linha descendo na tela)
     se YI è par, então:
       1o. incremento : +8192
       2o. incremento : -8112
     se YI è ímpar então:
       1o. incremento : -8112
       2o. incremento : +8192

para YI > YF.
     se YI êpar então:
       1o. incremento : +8112
       2o. incremento : -8192
     se YI è ímpar então:
       1o. incremento : -8192
       2o. incremento : +8112
```

7.1.4 is46_ponto_asm

Coloca um ponto numa das posições da tela gráfica.

Os parâmetros passados para esta rotina são x, y e cor, onde:

x - coordenada x do ponto;
y - coordenada y do ponto;
cor - cor em que o ponto deve ser traçado . As cores válidas são 0, 1 e 2 :
0 = faz com que o ponto seja traçado com a cor de fundo;
1 = faz com que o ponto seja aceso;
2 = faz com que seja feito um XOR do conteúdo da tela no ponto indicado.

Cada byte contêm oito pontos dipostos da seguinte forma: o bit mais significativo corresponde ao ponto com o menor valor de x dentre os que compõem o byte. Por exemplo, para acender o segundo ponto de um dos bytes devemos colocar em 1 o segundo bit de mais alta ordem : 01000000. A figura 7.1 ilustra a organização dos pontos na tela gráfica.

Desta forma a "máscara" 01000000 corresponde tanto aos pontos com coordenada x = 1, quanto aos de x = 9,17,25,..., ou seja, todos os pontos que situam-se no bit 6 (resto da divisão de x por 8 è 1).

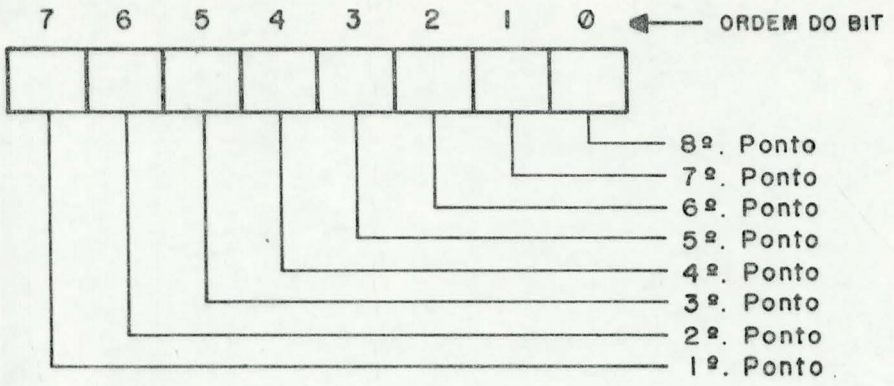

Figura 7.1 - Byte da Tela Gráfica.

O algoritmo usado é o seguinte :

1. Calcular o endereço do ponto (x,y).

2. Calcular a máscara do ponto a ser traçado:
   - calcula-se "r", o resto da divisão de x por 8
   - desloca-se "r" vezes para direita a "mascara" 10000000, o que fará com que fique em 1 apenas o bit correspondente á posição do ponto dentro do byte.

3. Se cor = 1
   então operar um "OR" da posição de memória onde deve ser traçado o ponto com a máscara obtida, acendendo desta forma apenas o bit correspondente ao ponto e mantendo os pontos adjacentes inalterados. O exemplo abaixo ilustra o funcionamento da rotina :

Para X = 10 temos
10 mod 8 = 2 (devemos deslocar a máscara dois bits para a direita)

1 0 0 0 0 0 0 0 --> 0 0 1 0 0 0 0 0

Supondo que o byte onde o ponto deverá ser traçado contenha "1 0 0 0 1 1 1 0 ", após a operação "OR" teremos:

```
  0 0 1 0 0 0 0 0
OR 1 0 0 0 1 1 1 0
   ================
  1 0 1 0 1 1 1 0
      !
      !
      ----------> apenas este bit foi modificado
```

4. Se cor = 0
   então negar a máscara;
   operar um "AND" da posição de memória onde deve ser traçado o ponto com a máscara obtida, apagando desta forma apenas o bit correspondente ao ponto e mantendo os pontos adjacentes inalterados. O exemplo abaixo ilustra o funcionamento da rotina:

Para X = 10 temos 10 mod 8 = 2

1 0 0 0 0 0 0 0 --> 0 0 1 0 0 0 0 0

Negando a máscara obtida teremos
1 1 0 1 1 1 1 1

Supondo que o byte onde o ponto deverá ser traçado contenha "1 0 0 0 1 1 1 0", após a operação "AND" teremos:

```
    1 1 0 1 1 1 1 1
AND 1 0 0 0 1 1 1 0
    ===============
    1 0 0 0 1 1 1 0
      !
      !
      ----------> apenas este bit foi modificado
```

5. Se cor = 2
   então operar um "XOR" da posição de memória onde deve ser traçado o ponto com a máscara obtida, invertendo desta forma apenas o bit correspondente ao ponto e mantendo os pontos adjacentes inalterados. O exemplo abaixo ilustra o funcionamento da rotina :

Para X = 10 temos 10 mod 8 = 2
             então devemos deslocar a máscara dois bits para a direita

1 0 0 0 0 0 0 0 --> 0 0 1 0 0 0 0 0

Supondo que o byte onde o ponto deverá ser traçado contenha "1 0 0 0 1 1 1 0", após a operação "XOR" teremos:

```
    0 0 1 0 0 0 0 0
XOR 1 0 0 0 1 1 1 0
    ===============
    1 0 1 0 1 1 1 0
      !
      !
      ----------> apenas este bit foi
                  modificado(trocado de 0 para 1)
```

7.1.5 Estilos de linhas

Os vários estilos de linhas que as rotinas descritas a seguir permitem são obtidos a partir de máscaras de 8 bits onde os bits em 1 correspondem aos pontos que devem ser traçados, e os em 0 aos que não devem ser alterados.

Para linhas horizontais temos:

```
estilo CHEIO          : 11111111
estilo PONTO-PONTO    : 10101010
estilo TRAÇO-TRACO    : 11100111
estilo TRACO_PONTO    : 11100100
```

Para linhas verticais temos:

```
estilo CHEIO          : 11111111
estilo PONTO-PONTO    : 10101010
estilo TRAÇO-TRAÇO    : 11101110
estilo TRAÇO_PONTO    : 11100100
```

O teste para verificar se um ponto deve ou não ser traçado é feito operando-se um AND da máscara do ponto com a máscara do estilo; se o resultado for 0 então o ponto não deve ser alterado; em caso contrário, deve ser ligado.

Exemplo : supondo que a máscara para um dos pontos de uma determinada linha seja "0 1 0 0 0 0 0 0" e o estilo seja TRACO-PONTO, teremos :

```
máscara do ponto  -->  0 1 0 0 0 0 0 0
máscara do estilo -->  1 1 1 0 0 1 0 0

resultado do AND      -->  0 1 0 0 0 0 0 0
```

Como o resultado da operação AND foi diferente de zero, o ponto deve ser traçado;

Para passar ao próximo ponto da linha basta deslocar a máscara um bit para a direita; caso a máscara torne-se 0 estaremos partindo para o traçado do primeiro ponto do byte seguinte (á direita). O exemplo abaixo ilustra o algoritmo:

Supondo que inicialmente a máscara seja
0 0 0 0 1 0 0 0
deslocando a máscara teremos:

```
0 0 0 0 1 0 0 0  --> 0 0 0 0 0 1 0 0  -->
0 0 0 0 0 0 1 0  --> 0 0 0 0 0 0 0 1  -->
0 0 0 0 0 0 0 0  --> (recriar a máscara) -->
1 0 0 0 0 0 0 0
```

7.1.6 is43_linha_asm

O traçado de linhas quaisquer é feito em duas etapas:

1) inicializações;
2) traçado da linha;

A FASE DE INICIALIZAÇÕES compõe-se dos seguintes passos :

1. Se XI > XF
   então trocar XI e XF entre si e trocar YI e YF entre si.

2. Calcular o número de pontos da linha em X
   DIF_X := XF - XI

3. Calcular o número de pontos da linha em Y
   DIF_Y := ABS(YF - YI)

4. Calcular os valores dos dois incrementos conforme explicado na seção 7.1.3.

5. Calcular o valor do ERRO que será usado no algoritmo de traçado da linha
   Se DIF_X > DIF_Y
   então ERRO := (DIF_X div 2)* -1
   senão ERRO := (DIF_Y div 2)

6. Calcular o endereço do ponto XI,YI conforme explicado na seção 7.1.3.

7. Calcular para que lado "cresce" a linha em X
   Se XI = XF
   então PASSO_X := 0 (* não cresce*)
   senão PASSO_X := 1 (*cresce para direita*)

8. Calcular para que lado "cresce" a linha em Y
   Se YI > YF
   então PASSO_Y := +1 (* cresce para baixo *)
   senão se YI < YF
   então PASSO_Y := -1 (*cresce para cima*)
   senão PASSO_Y := 0 (* não cresce *)

9. Criar "màscara" para o primeiro ponto da linha conforme rotina de pontos item 7.1.4.

A FASE DE TRAÇADO DE LINHA e' composta dos seguintes procedimentos :

Repetir os passos seguintes até que XI=XF e YI=YF

1. Traçar o ponto (XI,YI) conforme rotina de ponto (item 7.1.4) .

2. Se ERRO < 0
   então ERRO := ERRO + DIF_X
         Se PASSO_Y <> 0
         então somar á END o valor do primeiro incremento.
               trocar os incrementos entre si.
                  YI := YI + PASSO_Y
         senão ERRO := ERRO + DIF_Y
         Se PASSO_X <> 0
         então XI := XI + PASSO_X
               Deslocar máscara para direita 1 bit
               Se resultar 0
               então recriar máscara com primeiro bit em 1 e os demais em 0.
                     incrementar END.

3. Se XI = XF
   então PASSO_X := 0

4. Se YI = YF
   então PASSO_Y := 0

OBS.: o valor de END só é incrementado quando a máscara se torna 0, pois neste caso estaremos prestes a traçar um ponto que fica no próximo byte.

7.1.7 is_44linhah_asm

Para traçar linhas horizontais com maior velocidade, o algoritmo apresentado a seguir vale-se do fato de que os pontos que compõem tais linhas estão em posições contíguas de memória, o que possibilita a obtenção do endereço do ponto seguinte a partir do anterior.

O traçado das linhas horizontais divide-se em quatro partes :

- fase preliminar;
- traçado da máscara inicial : traçar os pontos que compõem o byte onde se encontra o primeiro ponto da linha;
- traçado dos pontos que compõem a parte central da linha, ou seja, do segundo ao penúltimo byte que compõe a linha;
- traçado da máscara final : traçar os pontos que compõe o byte onde se encontra o último ponto da linha .

FASE PRELIMINAR

1. Carregar os parâmetros
   XI = início da linha em X
   XF = fim da linha em x
   YI = coordenada y dos pontos da linha
   COR = cor da linha
   ESTILO = estilo em que a linha deve ser traçada

2. Calcular o número de pontos da linha
   Se XI > XF
   então trocar XI e XF entre si.
   DIF_X := XF - XI + 1.

3. Calcular o endereço do ponto XI, YI
   END := endereço do ponto

MASCARA INICIAL

1. Calcular a "máscara" para o primeiro ponto da linha conforme item 7.1.4.

2. Deslocar para a direita a máscara correspondente ao estilo o mesmo número de vezes que foi deslocada "máscara", no item anterior.

3. Testar se o ponto corrente deve ou não ser traçado conforme algoritmo apresentado na seção 7.1.5

4. Caso deva, traçá-lo conforme o item 7.1.4.

5. Decrementar o tamanho da linha; se resultar em zero, o traçado está terminado.

6. Deslocar "máscara" 1 bit para a direita; se "máscara" tornar-se 0, passar ao traçado da parte central da linha; senão, voltar aos três passos anteriores.

PARTE CENTRAL DA LINHA

1. Incrementar o endereço obtido anteriormente (passar ao próximo byte)

2. Dividir o tamanho atual da linha por 8, para obter o número de bytes que compõem a parte central da linha. Caso resulte em zero, passar ao traçado da máscara final da linha (quarta fase) pois a linha compunha-se de apenas dois bytes.

3. Traçar a parte central da linha usando o seguinte algoritmo :

```
- Se COR = 1
  então Enquanto DIF_X > 0
        faça
          decrementa DIF_X;
          operar um "OR" da posição de memória
             dada pelo endereço atual (END) com a
             máscara de estilo da linha;
          Incrementrar END;
- Se COR = 0
  então Negar a máscara de estilo da linha;
        Enquanto DIF_X > 0
        faça
          decrementar DIF_X;
          operar um "AND" da posição de memória
             dada pelo endereço atual (END) com a
             máscara de estilo da linha;
          Incrementrar END;
```

MASCARA FINAL

1. Se o último ponto da linha já foi traçado, então terminou o traçado da linha. Para verificar isto divide-se (XF + 1) por 8; se o resto for 0, a linha já está pronta.

2. Criar uma máscara para o último byte da linha:
   - Calcular a máscara para o ponto XF da mesma forma que se faz quando do traçado de pontos, item 7.1.4 .
   - Colocar em 1 todos os bits de ordem mais alta que o bit que está aceso na máscara obtida de XF.

Exemplo : para XF = 20 teremos ->
20 mod 8 = 4

10000000 ... 00001000 ... 11111000

- Operar um "AND" da máscara obtida no item anterior com a máscara correspondente ao estilo atual. Supondo estilo PONTO-PONTO, temos:

```
          1 1 1 1 1 0 0 0   - máscara obtida no item anterior
   AND    1 0 1 0 1 0 1 0   - máscara do estilo Ponto-Ponto
          ===============
          1 0 1 0 1 0 0 0   - MASCARA DO ÚLTIMO BYTE DA LINHA
```

- Com a máscara obtida no item anterior fazer :

```
Se COR = 1
então  operar um "OR"da máscara com  a posição
          de memória apontada por END
Se COR = 0
entao negar a "mascara"
      operar  um  AND  dela com  a  posição  de
      memória apontada por END.
```

OBS:

O traçado de linhas duplas ou espessas compõe-se, de fato, pelo traçado de duas linhas horizontais paralelas, usando para cada uma delas o estilo de linha CHEIA :

- Somar "dist" ao valor de YI
- Traçar uma linha usando o estilo CHEIA
- Subtrair "dist" de YI
- Traçar uma linha usando o estilo CHEIA

A variável "dist" indica a distância que deve ser deixada entre as duas linhas. No traçado de linhas do tipo DUPLA, a distância é 2; para linhas do tipo ESPESSA, a distância é zero (traça-se uma linha em YI e outra em YI-1).

7.1.8 is45_linhav_asm

Para traçar linhas verticais com maior velocidade, o algoritmo apresentado a seguir vale-se do fato de que os pontos que compõem estas linhas estão em posições de memória cujos endereços podem ser obtidos a partir do endereço do ponto anterior (conforme item 7.1.3). O algoritmo usado e' explicado a seguir :

1. Carregar os parâmetros
   YI = inicio da linha em Y
   YF = fim da linha em Y
   XI = coordenada X dos pontos da linha
   COR = cor da linha
   ESTILO = estilo em que a linha deve ser traçada

2. Se YI > YF
   então troca YI e YF entre si
   Calcular o numero de pontos da linha.
   DIF_Y := YF - YI + 1

3. Calcular os (2) incrementos conforme explicado na seção 7.1.3.

4. Calcular o endereço do ponto (XI,YI) conforme explicado na seção 7.1.3 --> END := endereço do ponto (XI,YI)

5. Calcular "máscara" para o ponto XI,YI conforme explicado no item 7.1.4 .

6. Criar uma segunda máscara "mascara_2" com apenas o bit de mais alta ordem em 1, e os demais em 0.

7. Repetir os passos seguintes enquanto DIF_Y > 0

- Verificar se o ponto corrente deve ou não ser traçado. "máscara_2" será usada para verificar se o ponto corrente deve ou não ser traçado. Esta verificação é feita operando-se um "AND" entre "máscara_2" e a máscara do estilo corrente; se a operação resultar em valor diferente de zero, o ponto corrente deve ser traçado; caso contrário não:

- Se (máscara_2 AND máscara) <> 0
  então traçar o ponto usando "máscara" da mesma forma que no item 7.1.4.
- Adicionar a END o valor do primeiro incremento.
- Trocar os incrementos entre si.
- Deslocar "mascara_2" um bit para a direita; caso torne-se 0, refazer "máscara_2" colocando em 1 apenas o bit de mais alta ordem.
- Decrementar DIF_Y.

OBS:

O traçado de linhas duplas e espessas compõe-se, de fato, do traçado de duas linhas verticais paralelas, usando para cada uma delas o estilo de linha CHEIA:

- Somar "dist" ao valor de XI
- Traçar uma linha usando o estilo CHEIA
- Subtrair "dist" de XI
- Traçar uma linha usando o estilo CHEIA

A variável "dist" indica a distância que deve ser deixada entre as duas linhas. No traçado de linhas do tipo DUPLA, a distância é 2; para linhas do tipo ESPESSA, a distância é zero (traça-se uma linha em XI e outra em XI+1).

7.2 Funções de texto

As funções para escrita de texto executam no seguinte ambiente :

a) caracteres representados através de matrizes de bits. Ex : matriz de 4 X 6, em que cada bit ligado (=1) corresponde a um pixel da tela que deve ser desenhado na cor de texto corrente (ver figura 7.2);

b) estão previstos diversos tipos e tamanhos de caracteres. O conjunto de matrizes que definem os caracteres de determinado tamanho e tipo é armazenado num arquivo, carregado na memória apenas mediante seleção do programador;

c) a exibição de cada caractere é feita por uma rotina

escrita em linguagem de montagem, ligada à função "is27_texto".

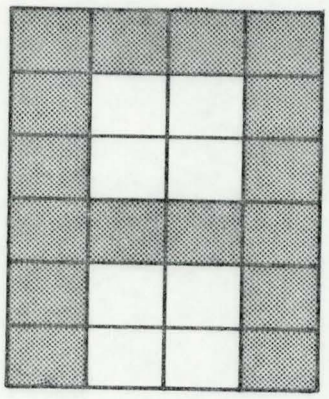

Figura 7.2 - Matriz que Define um Caractere.

7.2.1 Armazenamento das matrizes

Estão definidos apenas os caracteres ASCII cujo código esteja entre 33 e 126 (inclusive); caracteres fora desse intervalo são traduzidos como "espaços em branco" dentro do string.

Cada caractere é representado através de uma matriz de tamanho dependente do "fator de texto" corrente. As linhas da matriz são armazenadas em endereços contíguos de memória, na seguinte ordem : em primeiro lugar, as linhas da base do caractere; depois as superiores (ver figura 7.3).

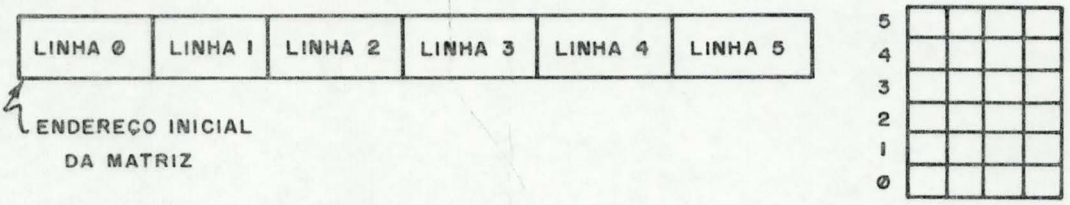

Figura 7.3 - Armazenamento das Matrizes.

A primeira linha da matriz sempre está armazenada em um endereço alinhado a byte. As demais linhas são alinhadas a nybble (de 4 em 4 bits). Desta forma, se o número de colunas não for múltiplo de 4, haverá desperdício de memória.

A área alocada para armazenamento do conjunto de matrizes ativo tem tamanho fixo, mas seu formato pode variar de acordo com o fator de texto corrente, podendo inclusive ser ocupado apenas parcialmente. Na presente versão, são dois os fatores de texto implantados :

- fator 0 : 4 X 6
  nro. de linhas = 6
  nro. de colunas = 4
  nro. de bytes/caractere = 3

- fator 1 : 8 X 8
  nro. de linhas = 8
  nro. de colunas = 8
  nro. de bytes/caractere = 8

Para calcular o endereço inicial da matriz que define determinado caractere, o seguinte cálculo deve ser feito :

Ei = Ea + ((c - 33) * NB)

onde :
Ei = endereço inicial da matriz
Ea = endereço da área de matrizes
c = código ASCII do caractere
NB = nro. de bytes/carac. para fator de texto corrente.

7.2.2 Estruturas de dados

a) tabela de tipos de texto (t_narqs) : associa os tipos de texto aos nomes dos arquivos que contêm as matrizes correspondentes a cada um dos fatores possíveis (ver figura 7.4). A linha é indexada com o identificador do tipo de letra e a coluna, com o identificador do fator de texto.

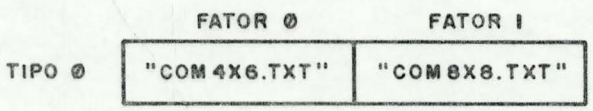

Figura 7.4 - Tabela de Nomes de Arquivos de Texto.

b) tabela de fatores de texto (t_fat) : associa os fatores de texto a seus tamanhos. Para cada fator, contêm :

- nro. de linhas da matriz.
- nro. de colunas da matriz.
- nro. de bits de espaçamento horizontal entre caracteres.
- nro. de bytes ocupados por cada matriz.

c) áreas para armazenamento de matrizes (t_cjmat0 e t_cjmat1) : são alocadas de acordo com o máximo fator de texto permitido :

- t_cjmat0 : no máximo 8 X 8
- t_cjmat1 : máximo fator possível (atualmente =1)

Estas áreas têm tamanho fixado em tempo de compilação e permanecem alocadas durante toda a execução do programa.

d) tabela de descritores dos conjuntos de caracteres : associa um identificador de conjunto de texto aos seus respectivos atributos :

- endereço da área para armazenamento das matrizes
- identificador de tipo de texto.
- identificador de fator de texto.
- nro. máximo de fator de texto permitido (definido em função do tamanho da área alocada).

7.2.3 is47_carac_asm

Esta função e'escrita em linguagem de montagem e exibe apenas um caractere de cada vez através do algoritmo explicado a seguir.

VARIAVEIS :

- end : endereço do byte na página gráfica em que o canto inferior esquerdo da matriz deve ser exibido.
- desloc : nro. de bits de deslocamento dentro do byte na página gráfica.
- nlin : nro. de linhas da matriz.
- ncol : nro. de colunas da matriz.
- nbits : nro. de bits da linha atual que ainda devem ser exibidos.
- M : máscara para escrita na página gráfica.
- M1aux : máscara auxiliar, armazena bits que sobraram da iteração anterior.
- M2aux : máscara auxiliar, armazena bits que sobraram da máscara atual devido ao deslocamento dentro do byte (desloc).

ALGORITMO

```
1. Inicializa variáveis :
     end, desloc - de acordo com x,y correntes.
     nlin, ncol  - de acordo com fator corrente.

2. Faça nlin vezes
     {
     nbits = ncol
     M1aux = zeros
     Enquanto nbits > 0
       {
       Se nbits < 8
       então M = próximos nbits da matriz.
       senão M = próximos 8 bits da matriz.

       nbits = nbits - 8
       M2aux = zeros

       desloca M "desloc" bits 'a direita, colocando os
          bits que saem de M na porção mais  significa-
          tiva de M2aux.

       M = M OR M1aux (junta M com resíduo anterior).

       Se M <> 0
       então escreve M no byte indicado por "end".
       senão deixa fundo como está.

       end = end + 1 (próximo byte na página gráfica).
       M1aux = M2aux
       }

     Se M1aux <> 0
     então  escreve  M1aux no byte indicado  por  "end"
            (resíduo).

     atualiza end para acessar linha de cima.
     }

3. Fim
```

OBSERVAÇÕES

a) os "próximos bits" da linha são sempre copiados nos bits mais significativos da variável M, sendo os demais preenchidos com zero. Existe ainda uma variável que indica o endereço dos próximos bits da matriz que devem ser lidos, assim como a informação de que tais bits começam no nybble superior ou no inferior;

b) a variável end é atualizada a cada nova linha para apontar a linha de varredura imediatamente superior (a exibição do caractere é feita de baixo para cima);

c) a escrita da máscara M na posição indicada por end pode ser feita de duas formas, conforme a cor de texto corrente :

```
cor = BRANCO :  liga os pixels
                MEM[end] = MEM[end] OR M
cor = PRETO  :  desliga pixels
                MEM[end] = MEM[end] AND (NOT M)
```

Este algoritmo foi implantado em linguagem de montagem através de uma função que recebe todos os parâmetros pela pilha do sistema :

- EMAT = dois bytes, contêm o endereço da matriz que descreve o caractere (offset relativo ao DS);
- EGRAF = dois bytes, contêm o endereço na página gráfica onde deve ser iniciada a exibição do caractere (offset relativo a B8000H);
- DESLOC = um byte, contêm o número de bits de deslocamento do caractere dentro do byte da página gráfica;
- NLIN = um byte, contêm o número de linhas da matriz que define o caractere;
- NCOL = um byte, contêm o número de colunas da matriz que define o caractere;
- COR = um byte, contêm a cor em que o caractere deve ser desenhado.

As variáveis usadas pela rotina estão, sempre que possível, armazenadas em registrador. A seguir serão comentadas as funções dos registradores na estrutura de dados da rotina :

- CS, DS, SS : não são alterados, apenas compartilhados com o programa em C.
- ES : base para acesso á página gráfica.
- SI : variável "end" (relativo a ES).
- DI : offset em DS para acesso ao próximo byte da matriz que define o caractere.
- AH : variável "M".
- AL : variável "M2aux".
- BH : variável "nbits".
- BL : indica os próximos bits da matriz que devem ser recuperados : BL = 0 : todo o byte indicado por DI.
  BL = 1 : apenas 4 bits inferiores.
- DL : variável "M1aux".

Os demais registradores não têm função específica.

7.2.4 is27_texto

A exibição de um string é feita pela rotina is27_texto, através de chamadas á função is47_carac_asm.

A seguir descreveremos sucintamente o funcionamento dessa rotina :

1. Calcula posição do string (canto inferior esquerdo do envelope do texto) : x = x corrente
   y = y corrente + 1

2. Calcula endereço da página gráfica em que o string deve começar a ser copiado :
   end (do byte) = ver seção 7.1.3
   desloc (no byte) = x módulo 8

3. Para cada caractere do string (até' nulo)
   {
   Se caractere está entre 33 e 126
   então {
   Calcula endereço da matriz do carac.
   Chama rotina em assembler
   }
   Calcula novo endereço (end) e deslocamento (desloc) do próximo caractere, considerando o número de colunas e espaço horizontal do fator de texto ativo.
   }

4. Fim

7.3 is25_ponto

A rotina de ponto foi implementada chamando uma rotina em assembler que desenha o ponto, passando como parâmetro as coordenadas (x, y) da posição corrente e a cor em que se deseja que o ponto seja desenhado.

7.4 is24_retang

A rotina de retângulo foi implementada utilizando-se duas rotinas em assembler de traçado de linha : linha horizontal e linha vertical. São traçados duas linhas horizontais, uma sobre a coordenada y indicada como parâmetro e outra sobre a coordenada pc.y, de x a pc.x. Posteriormente são traçadas duas linhas verticais , uma sobre a coordenada x recebida como parâmetro e outra sobre a coordenada pc.x , de y a pc.y .

7.5 is26_area

A rotina de área foi implementada utilizando-se a rotina em assembler de traçado de linhas horizontais. Com uma variável auxiliar assumindo valores entre pc.y (posição corrente) e y (parâmetro), traça-se linhas horizontais de pc.x (posição corrente) a x (parâmetro recebido) na cor de área corrente. Desta forma, esta rotina permite apenas o preenchimento de áreas com o padrão LISO.

BIBLIOGRAFIA

[ART 84] ARTWICK, B. **Applied Concepts in Microcomputer Graphics.** Englewood Cliffs, Prentice-Hall, 1984.

[MIC 83] MICROSOFT. **C Compiler; Language Reference.** 1983.

[MIC 85a] MICROSOFT. **C Compiler; Run-Time Library Reference.** 1985.

[MIC 85b] MICROSOFT. **C Compiler; User's Guide.** 1985.

[NOR 85] NORTON, P. **Programmer's Guide to the IBM PC.** Belleuvne, Washington, Microsoft Press, 1985.

[WAG 86a] WAGNER, F.R., C.M.D.S.-FREITAS e L.G.GOLENDZINER. Equivalência de Descrições Textuais e Gráficas de Sistemas Digitais num Ambiente de CAD. In: Seminário Integrado de Software e Hardware, XIII. Recife, PE, 19-25 Julho 1986. **Anais.** UFPe, Recife, 1986. pp 486-493.

[WAG 86b] WAGNER,F.R. et al. Ambiente integrado para Projeto de Sistemas Digitais Auxiliado por Computador. In: Congresso Nacional de Informática, XIX. Rio de Janeiro, 18-25 Agosto 1986. **Anais.** SUCESU, Rio de Janeiro, 1986. pp 111-116, vol.2.

[WAG 87a] WAGNER, F.R., C.M.D.S.-FREITAS e L.G.GOLENDZINER. **Linguagens de Descrição de Hardware para Suporte á Integração do Processo de Projeto em AMPLO.** Porto Alegre, PGCC da UFRGS, março 1987. (RP no. 065)

[WAG 87b] WAGNER, F.R.,C.M.D.S.-FREITAS e L.G.GOLENDZINER. A Digital Systems Design Methodology based on Nets of Agencies. In: BARBACCI,M. e C.J.KOOMEN (eds.) **CHDL'87.** Amsterdam, North-Holland, 1987. pp. 213-224.

APENDICE: Indice remissivo das rotinas

1. Rotinas de controle

2. Rotinas de alteração de atributos

3. Rotinas de saída

4. Rotinas de interrogação