Linguagem C Voltada ao Controle de Hardware

Roberto Francisco Ligeiro Marques
robertoligeiro@yahoo.com.br

Universidade Católica de Petrópolis – Faculdade de Informática
Petrópolis, Janeiro de 2001
Curso de Extensão

Parte 1 – Uma Visão Geral da Linguagem C

1.1 – As Origens da Linguagem C

A linguagem C foi inventada e implementada primeiramente por Denuis Ritchie em um DEC PDP-4 que utilizava o sistema operacional UNIX. C é o resultado de um processo de desenvolvimento que começou com uma linguagem mais antiga, chamada BCPL, que ainda está em uso, em sua forma original, na Europa. BCPL foi desenvolvida por Martin Richards e influenciou uma linguagem chamada B, inventada por Ken Thompson. Na década de 1970, B levou ao desenvolvimento de C.

Por muitos anos, de fato, o padrão para C foi a versão fornecida com o sistema operacional UNIX versão 5. Ele é descrito em *T/ie C Progrnmminig Language,* de Brian Kernighan e Dennis Ritchie (Englewood Cliffs, N.J.: Prentice Haíl, 4978). Com a popularidade dos microcomputadores, um grande número de implementações de C foi criado. Quase que por milagre, os códigos-fontes aceitos por essas implementações eram altamente compatíveis. (Isto é, um programa escrito com um deles podia normalmente ser compilado com sucesso usando-se um outro.) Porém, por não existir nenhum padrão, havia discrepâncias. Para remediar essa situação, o ANSI (*American Nacional Standards Institute*) estabeleceu, no verão de 1983, um comitê para criar um padrão que definiria de uma vez por todas a linguagem C.

1.2 – C - uma Linguagem de Médio Nível

C é freqüentemente chamada de linguagem de médio nível para computadores. Isso não significa que C seja menos poderosa, difícil de usar ou menos desenvolvida que uma linguagem de alto nível como BASIC e Pascal, tampouco implica que C seja similar à linguagem assembly e seus problemas correlatos aos usuários. C é tratada como uma linguagem de médio nível porque combina elementos de linguagens de alto nível com a funcionalidade da linguagem assembly.

Como uma linguagem de médio nível, C permite a manipulação de bits, bytes e endereços - os elementos básicos com os quais o computador funciona. Um código escrito em C é muito portável. Portabilidade significa que é possível adaptar um software escrito para um tipo de computador a outro. Por exemplo, se você pode facilmente converter um programa escrito para DOS de tal forma a executar sob Windows, então esse programa é portável.

Todas as linguagens de programação de alto nível suportam o conceito de tipos de dados. Um tipo de dado define um conjunto de valores que uma variável pode armazenar e o conjunto de operações que pode ser executado com essa variável. Tipos de dados comuns são inteiro, caractere e real. Embora C tenha cinco tipos de dados internos, ela não é uma linguagem rica em tipos de dados como Pascal e Ada. C permite quase todas conversões de tipos. Por exemplo, os tipos caractere e inteiro podem ser

livremente misturados na maioria das expressões. C não efetua nenhuma verificação no tempo de execução, como a validação dos limites das matrizes. Esses tipos de verificações são de responsabilidade do programador.

As versões originais de C não realizavam muitos (se é que realizavam algum) testes de compatibilidade entre um parâmetro de uma função e o argumento usado para chamar a função. Por exemplo, na versão original de C, você poderia chamar uma função, usando um ponteiro, sem gerar uma mensagem de erro, mesmo que essa função tivesse sido definida, na realidade, como recebendo um argumento em ponto flutuante. No entanto, o padrão ANSI introduziu o conceito de *protótipos de funções,* que permite que alguns desses erros em potencial sejam mostrados, conforme a intenção do programador.

Outro aspecto importante de C é que ele tem apenas 32 palavras-chaves (27 do padrão de fato estabelecido por Kernighan e Ritchie, mais 5 adicionadas pelo comitê ANSI de padronização), que são os comandos que compõem a linguagem C. As linguagens de alto nível tipicamente têm várias vezes esse número de palavras reservadas. Como comparação, considere que a maioria das versões de BASIC possuem bem mais de 100 palavras reservadas!

1.3 – C - uma Linguagem para Programadores

Surpreendentemente, nem todas as linguagens de computador são para programadores. Considere os exemplos clássicos de linguagens para não-programadores: COBOL e BASIC. COBOL não foi destinada para facilitar a vida do programador, aumentar a segurança do código produzido ou a velocidade em que o código pode ser escrito Ao contrário, COBOL foi concebida, em parte, para permitir que não-programadores leiam e presumivelmente (embora isso seja improvável) entendam o programa. BASIC foi criada essencialmente para permitir que não-programadores programem um computador para resolver problemas relativamente simples

Em contraposição, C foi criada, influenciada e testada em campo por programadores profissionais O resultado final é que C dá ao programador o que ele quer: poucas restrições, poucas reclamações, estruturas de bloco, funções isoladas e um conjunto compacto de palavras-chave. Usando C, um programador pode conseguir aproximadamente a eficiência de código assembly combinada com a estrutura de ALGOL ou Modula-2. Não é de admirar que C seja tranqüilamente a linguagem mais popular entre excelentes programadores profissionais.

O fato de C freqüentemente ser usada em lugar da linguagem assembly é o fator mais importante para a sua popularidade entre os programadores. A linguagem assenbly usa uma representação simbólica do código binário real que o computador executa diretamente. Cada operação em linguagem assembly leva a uma tarefa simples a ser executada pelo computador. Embora a linguagem assembly dê aos programadores o potencial de realizar tarefas com máxima flexibilidade e eficiência, é notoriamente difícil de trabalhar quando se está desenvolvendo ou depurando um programa. Além

disso, como assembly não é uma linguagem estruturada, o programa final tende a ser um código "espaguete" - um emaranhado de *jumps*, *calls* e índices. Essa falta de estrutura torna os programas em linguagem assembly difíceis de ler, aperfeiçoar e manter. Talvez mais importante: as rotinas em linguagem assembly não são portáveis entre máquinas com unidades centrais de processamento (CPUs) diferentes.

Inicialmente, C era usada na programação de sistema. Um *programa de sistema* forma uma porção do sistema operacional do computador ou de seus utilitários de suporte. Por exemplo, os programas que seguem são freqüentemente chamados de programas de sistema:

- Sistemas operacionais
- Interpretadores
- Editores
- Programas de planilhas eletrônicas
- Compiladores
- Gerenciadores de banco de dados

Em virtude da sua portabilidade e eficiência, à medida que C cresceu em popularidade, muitos programadores começaram a usá-la para programar todas as tarefas. Por haver compiladores C para quase todos os computadores, é possível tomar um código escrito para uma máquina, compilá-lo e rodá-lo em outra com pouca ou nenhuma modificação. Esta portabilidade economiza tempo e dinheiro. Os compiladores C também tendem a produzir um código-objeto muito compacto e rápido - menor e mais rápido que aquele da maioria dos compiladores BASIC, por exemplo.

Além disso, ela oferece a velocidade da linguagem assembly e a extensibilidade de FORTH, mas poucas das restrições de Pascal ou Modula-2. Cada programador C pode, de acordo com sua própria personalidade, criar e manter uma biblioteca única de funções customizadas, para ser usada em muitos programas diferentes. Por admitir - na verdade encorajar - a compilação separada, C permite que os programadores gerenciem facilmente grandes projetos com mínima duplicação de esforço.

Parte 2 – Conceitos Básicos para Controle de Hardware

2.1 – Operadores Bit a Bit

Ao contrário de muitas outras linguagens, C suporta um completo conjunto de operadores bit a bit. Uma vez que C foi projetada para substituir a linguagem assembly na maioria das tarefas de programação, era importante que ela tivesse a habilidade de suportar muitas das operações que podem ser feitas em linguagem assembly. *Operação bit a bit* refere-se a testar, atribuir ou deslocar os bits efetivos em um byte ou uma palavra, que correspondem aos tipos de dados char e int e variantes do padrão C. Operações bit não podem ser usadas em **float, double, long double, void** ou outros tipos mais complexos. A Tabela 2.1 lista os operadores que se aplicam às operações bit a bit. Essas operações são aplicadas aos bits individuais dos operandos.

Operador	Ação
&	AND
|	OR
^	XOR
~	Complemento de um
>>	Deslocamento à esquerda
<<	Deslocamento à direita

Tabela 2.1.1

Operações bit a bit encontram aplicações mais freqüentemente em "drivers" de dispositivos - como em programas de modems, rotinas de arquivos em disco e rotinas de impressoras - porque as operações bit a bit mascaram certos bits, como o bit de paridade. (O bit de paridade confirma se o restante dos bits em um byte não se modificaram. É geralmente o bit mais significativo em cada byte.)

Imagine o operador AND como uma maneira de desligar bits. Isto é, qualquer bit que é zero, em qualquer operando, faz com que o bit correspondente no resultado seja desligado. No exemplo abaixo deseja-se levar a zero apenas o bit mais significativo do *byte* 11000001.

```
11000001    193 em binário
01111111    127 em binário
            AND bit a bit
01000001    resultado
```

O operador OR, ao contrário de AND, pode ser usado para ligar um bit. Qualquer bit que é 1, em qualquer operando, faz com que o bit correspondente no resultado seja

ligado. For exemplo, o seguinte mostra a operação 128 | 3:

```
10000000    128 em binário
00000011    3 em binário
            OR bit a bit
10000011    resultado
```

Um OR exclusivo, normalmente abreviado por XOR, ativa um bit se, e somente se, os bits comparados forem diferentes. Por exemplo, 127 ^120 é:

```
01111111    127 em binário
01111000    120 em binário
            XOR bit a bit
00000111    resultado
```

Os operadores de deslocamento, >> e << movem todos os bits de uma variável para a direita ou para a esquerda, como especificado. A forma geral do comando de deslocamento à direita é:

Variável >> número de posições de bits

A forma geral do comando de deslocamento à esquerda é
Variável << número de posições de bits

Conforme os bits são deslocados para uma extremidade, zeros são colocados na outra. Lembre-se de que um deslocamento *não* é uma rotação. Ou seja, os bits que saem por uma extremidade não voltam para a outra. Os bits deslocados são perdidos e zeros são colocados.

Operações de deslocamento de bits podem ser úteis quando se decodifica a entrada de um dispositivo externo, como um conversor D/A, e quando se lêem informações de estado. Os operadores de deslocamento em nível de bits também podem multiplicar e dividir inteiros rapidamente. Um deslocamento à direita efetivamente multiplica um número por 2 e um deslocamento à esquerda divide-o por 2.

```
/* Um exemplo de deslocamento de bits. */
      #include <stdio.h>
        void main(void)
          {
          unsigned int i, j;
          i = 1;
        1* deslocamentos à esquerda */
         for( j=0 ;  j<4 ;  j++)
          {j  =  i << 1; /*desloca i de 1 à esquerda,que é o mesmo que
                           multiplicar por 2 */
          printf("deslocamento à esquerda %d : %d \n", j, i); }

         /* deslocamentos à direita */
         for(j=0; j<4; j++)
          {i =  i >> 1; /* desloca i de 1 à direita, que é o     mesmo
                           que dividir por 2 */
           printf("deslocamento à direita %d : %d \n", j, i); }
      }
```

O operador de complemento a um, inverte o estado de cada bit da variável especificada. Ou seja, todos os 1s são colocados em 0 e todos os 0s são colocados em 1.

2.2 Funções de C para leitura e escrita em Hardware

O controle de certos dispositivos de Hardware freqüentemente e feito através da leitura e escrita de alguns endereços de controle do dispositivo em questão. Alguns compiladores C oferecem algumas funções para auxiliar neste trabalho.

As funções presentes a seguir estão presentes no compilador Borland Turbo C:

int biosequip(void)

Declarada em <bios.h>.

Executa a INT 11h e retorna as portas e os periféricos conectados.

Bits	Indicação
15 e 14	Número de portas paralelas
13	Modem interno instalado
12	Número de portas de jogos
11 a 9	Número de portas seriais
8	Número de drives de discos flexiveis + 1
7 e 6	DMA presente
5 e 4	Resolução no modo texto(1 - 40x25 colorido, 2 - 80x25 colorido, 3 - 80x25 mono)
3 e 2	Número de blocos de 16 KB instalado na placa mãe
1	Co-processador instalado
0	Boot do disquete

Tabela 2.2.1

void harderr(void far *rotina)

Declarada em <dos.h>.

Associa uma rotina de manipulação de erro critico. A rotina deve conter quatro parâmetros inteiros de entrada: o número do erro, e os valores de Ax, BP e SI.

Exemplo: *int far rotina (int num_erro int ax, int bp, int si);*

Esta rotina deve retornar 0 *(ignore),* 1 *(retray)* ou 2 *(abort)*.

int inport(int porta)

Declarada em <dos.h>.

Retorna a *word* lida a partir do endereço de 110 especificado por porta.

int inportb(int porta)

Declarada em <dos.h>.

Retorna o byte lido no endereço de 1/0 especificado por porta.

void outport(int porta, int word)

Declarada em <dos.h>.

Escreve uma *word* a partir do endereço de I/O especificado por porta.

void outportb(int porta, int byte)

Declarada em <dos.h>.

Escreve um byte no endereço de I/0 especificado por porta.

Parte 3 – Controlando o Mouse

3.1 – O Mouse

3.1.1 – Principio eletrônico

Com o objetivo de suprir algumas deficiências históricas da entrada de dados, via teclado (como a lentidão dos não-datilógrafos), idealizou-se um pequeno dispositivo que também se comunica serialmente com a placa mãe: o mouse. Sua idéia principal é fazer um cursor varrer uma tela gráfica ou texto, posicionando-se a partir de um determinado pixel, provavelmente associado a alguma opção, ícone ou item de menu.

No *Iayout* original do mouse, previram-se dois ou três botões (ou chaves) e uma esfera de borracha (alto coeficiente de atrito), apoiada na superfície de um plano e em contato com pelo menos, duas roldanas, funcionando como engrenagens, rolando sem deslizar: a primeira acompanha o movimento da esfera no eixo-X e a segunda no eixo-Y

Ou seja, o projeto eletrônico \ mecânico de um mouse deve contar com os seguintes componentes ou equivalentes:

- dois ou três botões,
- pelo menos duas roldanas,
- dois transdutores de posição angular para a conversão dos deslocamentos das roldanas em dados digitais,
- interface serial para comunicação com o PC.

Quanto à forma de conexão no PC, um mouse pode: ser conectado numa das portas seriais padrões (COM 1,COM2, COM3 ou COM4); ou ser conectado diretamente no barramento do PC, através de uma entrada especial. Neste caso, poderá ou não haver interrupção (IRQ12, se houver).

O protocolo pelo qual se comunicam mouse e placa-mãe consiste numa especificação de como são passadas as informações referentes aos diques e movimentos da esfera. Teoricamente, os protocolos poderiam variar entre fabricantes, mas dois deles são os mais utilizados:

- protocolo de dois botões em três bytes;
- protocolo de três botões em cinco bytes.

3.1.2 – A INT33h

A INT 33h constitui-se numa gama de serviços que facilitam a programação com o mouse. Ela está associada a um driver, instalável, a quem estão icumbidas as funções de:

- detectar a conexão do mouse;
- interpretar as interrupções da porta serial, convertendo o protocolo lido em dados mais acessíveis;
- apagar e redesenhar o cursor na tela toda vez que o mouse for movimentado;
- dispor aos programas funções diversas, como a leitura das coordenadas do cursor, do estado dos botões, etc.

Porém, para se evitar o uso direto da INT33, utiliza-se a biblioteca "Mouse.h", onde já se encontram implementadas todas as funções para controle do mouse.

Observação:
As Funções da INT33 se encontram no Apêndice B

3.1.3 – A Biblioteca "Mouse.h"

A biblioteca <mouse.h> apresentada a seguir foi originalmente apresentada no livro PC e Periféricos, Um guia Completo de Programação, ZELENOVISK Ricardo, porém sofreu algumas alterações pelo autor desta apostila. Esta biblioteca se apresenta como uma forma prática e rápida de se utilizar o mouse em ambientes DOS em programas desenvolvidos em C.

```
                                    <mouse.h>

/* funcao para inicializacao do mouse, retorna  -1 se mouse instalado
*/
int inicializa_mouse(void)
{  regs.x.ax=0x0000;
  int86(0x33,&regs,&regs);
  return(regs.x.ax);}

void mostra_cursor(void)
{  regs.x.ax=0x0001;
  int86(0x33,&regs,&regs);}

void esconde_cursor(void)
{  regs.x.ax=0x0002;
  int86(0x33,&regs,&regs);}

/* define numero de interrupcoes do mouse */
void mouse_interrupt(int n)
```

```
{  regs.x.ax=0x001c;
  regs.x.bx=n;
  int86(0x33,&regs,&regs);}

/* funcao retorna o estado dos botoes e posicao do mouse */
struct mouse_set get_mouse_set(void)
{  struct mouse_set status;
  regs.x.ax=0x0003;
  int86(0x33,&regs,&regs);
  status.x=regs.x.cx;
  status.y=regs.x.dx;
  status.esquerdo=regs.x.bx & 1;
  status.direito=regs.x.bx & 2;
  status.meio=regs.x.bx & 4;
  return(status);}

void ajusta_posicao_cursor(int x, int y)
{  regs.x.ax=0x0004;
  regs.x.cx=x;
  regs.x.dx=y;
  int86(0x33,&regs,&regs);}

/* retorna informacoes sobre os cliques desde a ultima chamada */
struct mouse_set get_mouse_cliques(int botao)
{  struct mouse_set status;
  regs.x.ax=0x0005;
  regs.x.bx=botao;
  int86(0x33,&regs,&regs);
  status.x=regs.x.cx;
  status.y=regs.x.dx;
  status.press=regs.x.bx;
  return(status);}

/* retorna o estado dos botoes */
struct mouse_set mouse_clique()
{  struct mouse_set status;
  regs.x.ax=0x0003;
  int86(0x33,&regs,&regs);
  status.esquerdo=regs.x.bx;
  status.x=regs.x.cx;
  status.y=regs.x.dx;
  status.press=regs.x.bx;
  return(status);}

/* define a velocidade do mouse */
void mouse_velocidade(int n)
{  regs.x.ax=0x0013;
   regs.x.dx=n;
   int86(0x33,&regs,&regs);}

/* obter informacao sobre quantas vezes o botao foi solto desde a
ultima chamada*/
struct mouse_set get_mouse_solto(int botao)
{  struct mouse_set status;
  regs.x.ax=0x0006;
  regs.x.bx=botao;
  int86(0x33,&regs,&regs);
  status.x=regs.x.cx;
  status.y=regs.x.dx;
  status.press=regs.x.bx;
  return(status);}
```

```
/* define area de cobertura do cursor */
void cursor_area(int esquerda,int direita, int cima, int baixo)
{  regs.x.ax=0x0007;
  regs.x.cx=esquerda;
  regs.x.dx=direita;
  int86(0x33,&regs,&regs);
  regs.x.ax=0x0008;
  regs.x.cx=cima;
  regs.x.dx=baixo;
  int86(0x33,&regs,&regs);}

/* define cursor grafico e pixel de referencia */
void define_cursor_grafico(int x, int y, char far *cursor)
{  regs.x.ax=0x0009;
  regs.x.bx=x;
  regs.x.cx=y;
  segregs.es=FP_SEG(cursor);
  segregs.cs=_CS;
  segregs.ss=_SS;
  segregs.ds=_DS;
  regs.x.dx=FP_OFF(cursor);
  int86x(0x33,&regs,&regs,&segregs);}

void define_sensibilidade(int taxax,int taxay)
{  regs.x.ax=0x000F;
  regs.x.cx=taxax;
  regs.x.dx=taxay;
  int86(0x33,&regs,&regs);}
```

3.2 – Programa Exemplo para escrever na tela as coordenadas do Mouse.

```
#include <mouse.h>
#include <graphics.h>
#include <stdio.h>
#include <conio.h>
int gdriver = VGA, gmode = VGAHI;
struct mouse_set teste;

main()
{   int op=0;
     if (!(inicializa_mouse()))
      {
        printf("\n\rMouse nao instalado. Programa abortado.");
        exit(0);
      }
   initgraph(&gdriver,&gmode,"");
   mostra_cursor();
   while((op=kbhit())==0)
        {
           teste=mouse_clique();
            gotoxy(1,1);
            printf("x:%d , y:%d",teste.x,teste.y);
        }
   }
```

Exercício Proposto 1:

Desenvolver um programa de desenho simplificado em C que utilize a biblioteca *mouse.h* .
Dica: Utilizar biblioteca graphics.h (ver Apêndice C).

Parte 4 – Programando a Porta Paralela

4.1 – A Porta Paralela

A porta paralela, também chamada de porta de impressora, está projetada de forma a permitir a conexão do PC com impressoras paralelas, mas também pode ser usada como uma porta de entrada/sadia genérica. Ela é constituída por 3 registradores que possuem as seguintes características:

Endereço	Nome	n º bits	Entrada	Saída	Leitura	Escrita
378h	dados	8		X	X	X
379h	status	5	X		X	
37Ah	controle	4	X	X	X	X

Tabela 4.1.1 : Composição da Porta Paralela

No endereço 378h, está o registrador de dados, com 8 bits, usado para sadia, mas que também pode ser lido. No endereço 379h, está o registrador de status, que trabalha como uma entrada de 5 bits. Já no endereço 37Ah, está o registrador de controle, que opera com 4 bits, no modo quase bidirecional, podendo ser configurado para entrada ou saída (interface a coletor aberto). Uma das entradas do endereço 379h pode provocar interrupção, cuja habilitação é controlada por software. Todos os pinos de entrada/saída estão disponíveis em um conector tipo DB-25, geralmente localizado na placa multi-I/O.

4.2 - Registrador de dados (378h - saída com 8 bits)

A tabela 4.2.1 mostra como os bits do registrador de dados estão ligados aos pinos do conector.

	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
Pinos	9	8	7	6	5	4	3	2

Tabela 4.2.1: Pinagem da porta de dados.

Esses pinos deverão ser capazes de fornecer/drenar *(saurce/sink)* 2,6/24 mA. Uma instrução OUT escreve dados diretamente nos pinos do conector. Ao escrever 1 em um bit, resulta-se um nível TTL alto na saída. Esse registrador também pode ser lido com uma instrução *IN,* o que permite verificar se os dados foram corretamente transferidos.

4.3 - Registrador de status (379h - entrada com 5 bits)

A tabela 4.3.1 indica como os bits dessa porta estão ligados aos pinos do conector. Uma leitura no endereço 379h, com uma instrução *IN,* reflete o estado imediato desses

pinos. Notar que o bit 7 (pino 11) é lido invertido. O pino 10 pode ser usado para provocar a interrupção IRQ7, isto quando, neste pino, houver uma transição de nível baixo para nível alto, condicionado ao bit 4 da porta de controle (registrador 37Ah) estar em nível alto.

	bit 7	**bit 6**	**bit 5**	**bit 4**	**bit 3**	**bit 2**	**bit 1**	**bit 0**
Pinos	11(L)	10	12	13	15	-	-	-

(L) indica que a entrada é invertida
Tabela 4.3.1 : Pinagem do registrador de status.

4.4 - Registrador de controle (37Ah - bidirecional com 4 bits)

A tabela 4.4.1 indica como os bits desse registrador estão ligados aos pinos do conector. O bit4 é usado para controlar a habilitação da IRQ7. Quando este bit é colocado em nível alto (1 lógico), a interrupção está habilitada. Os bits deste registrador estão a coletor aberto, isto lhes permite trabalhar tanto como entrada quanto como saída.

Para trabalhar como saída, basta, simplesmente, usar a instrução *OUT* para escrever nos pinos. Para ser usada como entrada, é necessário que antes se programe as saídas em 1 (porta quase bidirecional); isto permitirá que elementos externos "puxem esse 1 para 0". As leituras, usando a instrução IN, vão obter o valor instantâneo de cada pino. Ao configurar-se essa porta como entrada, ou seja, ao escrever-se 1 em todos os pinos, é necessário tomar cuidado, pois alguns bits estão invertidos. Ao escrever-se no endereço 37Ah o dado 4 programa-se a porta como entrada e desabilita-se a interrupção.

	bit 7	**bit 6**	**bit 5**	**bit 4**	**bit 3**	**bit 2**	**bit 1**	**bit 0**
Pinos	9	8	7	habIRQ7	17(L)	16(H)	14(L)	1(L)
Reset	-	-	-	0	1	0	1	1

(L) indica que a entrada é invertida
Tabela 4.4.1 : Pinagem do Registrador de Controle bidirecional

4.5 – Programa exemplo para controle da Porta Paralela

```
#include <dos.h>
#include <stdio.h>
#include <conio.h>
#define apagado     0
#define porta     0x378

main()
{
  int op;

  clrscr();
  outportb(porta,0x0);
  while((op=getch())!=27)
     {
     if(op==77)
```

```
            {outportb(porta,0x01);
             printf(" endereco 378h, Bit0 -> Pino 2\n");}

          if(op==75)
            {outportb(porta,0x02);
             printf(" endereço 378h, Bit1 -> Pino 3\n");}

          if(op==72)
            {outportb(porta,0x03);
            printf(" endereço 378h, Bit1 e Bit2 -> Pinos 2 e 3\n");}

          if(op==80)
            {outportb(porta,0x00);
             printf(" endereço 378h, apagado\n"); }
           }
      }
```

Exercício Proposto 2:

Desenvolver um programa em C para enviar sinais para porta paralela de acordo com numero de entrada .
Ex.: entrada no teclado = 2 , envia sinal para pino 2.

Dica: 1. Não utilizar o pino 1.
2. Endereço 378h

Parte 5 – Programando a Porta Serial

5.1 – Comunicação Serial no PC

As portas seriais do PC são totalmente programáveis e somente permitem a comunicação assíncrona. E possível programar os bits de partida, os bits de paridade e os bits de parada. Um gerador de taxa de comunicação *("baud-rate')* programável permite a operação a até 115.200 bauds. Pode-se transmitir caracteres com 5, 6, 7 ou 8 bits, com 1, 1 1/2 ou 2 bits de parada. Um sistema de interrupções priorizáveis controla as interrupções de transmissão, recepção, erro ou estado de linha. Existe internamente um recurso para diagnóstico, através da função de *"Ioop-back"*. O coração da interface serial é o circuito 8250, ou seu equivalente funcional. Com ele, conseguem-se outras características adicionais:

- amplificação dupla, que elimina a necessidade de uma sincronização precisa;
- entrada independente para o relógio de recepção;
- funções para controle de modem:
- CTS - *"Clear to Send'*
- RTS - *"Request to Send'*
- DSR - *"Data Set Ready'*
- DTR - *"Data Terminal Ready'*
- RI - *"Ring lnd,cator"*
- CD - *"Carrier Detecr';*
- detecção de partida falsa;
- geração e detecção de *"une break'.*

Os diferentes modos de operação são selecionados através da programação do 8250. Esse circuito usa 8 endereços consecutivos. A porta COM 1 do PC tem seu 8250 mapeado a partir do endereço 3F8h e a COM 2 a partir do endereço 2F8h. Estão também previstas a COM 3 e a COM 4, que usam os endereços bases 3E8h e 2E8h, respectivamente. Os dados são transmitidos pelo 8250, em seqüência, do bit menos significativo para o mais significativo. Isso está ilustrado na figura 4.1.1. A interface insere automaticamente o bit de partida, o bit de paridade, se programado, e os bits de parada.

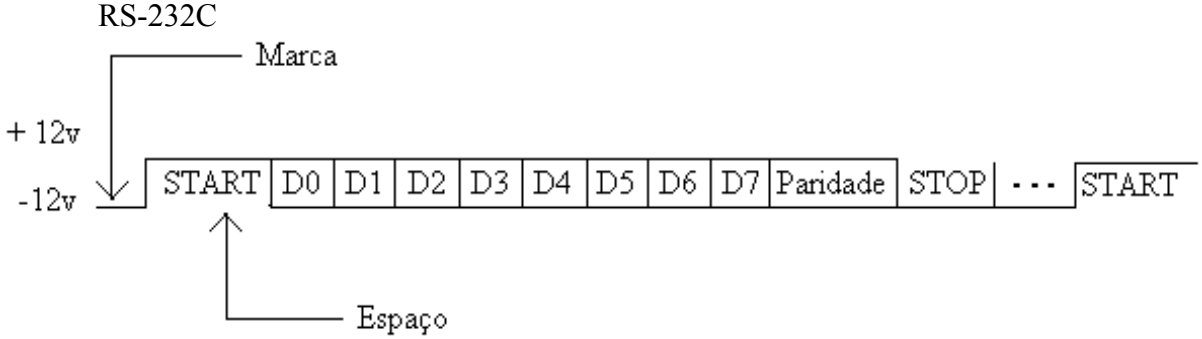


Figura 5.1.1. Seqüência de bits numa transmissão serial.

Uma saída RS 232C inativa fica transmitindo "marca" (-12 V). Isto é uma excelente indicação para que, em caso de dúvida, possa descobrir-se quem é saída e quem é entrada. Com um multímetro, medem-se as tensões dos pinos: os que estiverem em -12 V serão saídas, os demais serão entradas, exceto o terra e os pinos desconectados. O outro nível permitido, +12 V, recebe o nome de "espaço".

Um sinal será considerado "marca" quando a tensão na linha, medida no ponto da interface, for mais negativa que -3 V, e será considerado "espaço" quando a tensão medida for mais positiva que +3 V. A região entre +3 V e -3 V é denominada região de transição e é considerada como nível inválido. Uma tensão acima de +15 V ou abaixo de -15 V também será considerada inválida.

Durante uma transmissão de dados, a "marca" (-12 V) é usada para caracterizar o estado binário 1 (bit = 1) e o "espaço" (+12V) é usado para caracterizar o estado binário 0 (bit = 0). Para os circuitos de controle da interface, uma função estará em ON (ativada) quando sua tensão for +12 V e estará em 0FF (desativada) quando sua tensão for -12 V. Em outras palavras, uma saída em +12 V indica que a função que ela simboliza está ativada; já uma saída em -12 V indica função desativada.

5.2 – Descrição dos Pinos da Porta Serial

Os sinais da porta serial do PC estão disponíveis em um conector tipo DB macho, de 9 ou 25 pinos. Este conector, normalmente, está na parte traseira do computador. A distribuição dos sinais RS-232C está mostrada na tabela 4.2.1.

9 Pinos	25 Pinos	Sinal
3	**2**	**TD**
2	**3**	**RD**
7	**4**	**RTS**
8	**5**	**CTS**
6	**6**	**DSR**
5	**7**	**GND**
1	**8**	**CD**
4	**20**	**DTR**
9	**22**	**RI**

Tabela 5.2.1.

5.2.1 – Sinais de Entrada (Recebidos pelo PC)

Os sinais de entrada são aqueles gerados por outro dispositivo serial. Esses sinais são recebidos pelo PC e, depois de passarem por um conversor RS 232CITTL, são entregues ao 8250. A seguir, é feita uma descrição da finalidade de cada um deles.

RD - Received Data: Esse sinal é entregue ao pino 10 do 8250 (SIN - *"Serial lnput")*. Por ele, chegam os dados seriais.

CTS - Clear To Send: Por essa linha, o periférico informa que está pronto para transmitir dados. O sinal CTS pode ser lido através do bit 4 do Registrador de Estado do Modem. O bit O (DCTS), desse mesmo registrador, indica se CTS foi alterado desde a última leitura. Também é possível gerar uma interrupção quando CTS muda de estado.

DSR - Data Set Ready: Por essa linha, o periférico informa que está pronto para se comunicar. O estado da entrada DSR pode ser determinado através do bit 5 do Registrador de Estado do Modem. O bit 1 (DDSR) desse mesmo registrador indica se a entrada DSR sofreu alguma alteração desde a última leitura. Também é possível gerar uma interrupção quando a entrada DSR muda de estado.

CD - Carrier Detect: Por essa linha, o periférico indica que detectou a portadora, o que é mais usual no caso de um modem. O estado dessa entrada pode ser determinado através do bit 7 (RLSD) do Registrador de Estado do Modem. O bit 3 (DLSR) desse mesmo registrador indica se a entrada CD sofreu alguma alteração desde a última leitura. Também é possível gerar uma interrupção quando a entrada CD (RLSD> muda de estado.

RI - Ring Indicator: Serve para o periférico, neste caso um modem, indicar o recebimento do tom de discar da linha telefônica. O estado dessa entrada pode ser verificado através do bit 6 (RI) do Registrador de Estado do Modem. O bit 2 (TERI) desse mesmo registrador indica se RI sofreu uma transição de baixo para alto, desde a última leitura. Também é possível gerar uma interrupção com a sua transição, de alto para baixo.

5.2.2 – Sinais de Saída (Enviados pelo PC)

Os sinais de saída são aqueles gerados pelo PC e enviados a outro dispositivo serial. Esses sinais são gerados no 8250 e, depois de passarem por um conversor TTLIRS 232C, são entregues ao conector serial. A seguir, é feita uma descrição da finalidade de cada um desses sinais.

TD - Transmitted Data: Esse sinal é refletido no pino 11 (SOUT - *"Serial Our')* do 8250. Por ele, são enviados os dados seriais. A saída SOUT vai para o estado de marca (-12 V) após a inicialização (reset).

RTS - Request to Send: Informa ao periférico que o 8250 está pronto para transmitir dados. A saída RTS pode ser ativada através do bit 1 (RTS) do Registrador de Controle do Modem. A saída RTS é colocada em alto (+12V = ativa) após a operação de inicialização (reset).

DTR - Data Terminal Ready: Informa ao periférico que o 8250 está pronto para se comunicar. A saída DTR pode ser ativada através do bit O (DTR) do Registrador de Controle do Modem. A saida DTR é colocada em alto (+12 V = ativa) após a operação de inicialização (reset).

5.3 – P*rogramação do 8250*

Todo o controle da porta serial do PC é feito através do controlador 8250. Existem funções da BIOS que se prestam para a programação deste CI; porém, grandes vantagens são obtidas com o controle direto deste circuito. A tabela 5.3.1 mostra os endereços dos registradores do 8250, que serão posteriormente estudados.

COM4	COM3	COM2	COM1	REG	Nome dos Registradores	bit DLAB/LCR
2E8h	3E8h	2F8h	3F8h	TxB	Buffer de Transmissão	0 (escrita)
2E8h	3E8h	2F8h	3F8h	RxB	Buffer de Recepção	0 (leitura)
2E8h	3E8h	2F8h	3F8h	DLL	Latch para divisor (LSB)	1
2E9h	3E9h	2F9h	3F9h	DLM	Latch para divisor (MSB)	1
2E9h	3E9h	2F9h	3F9h	lER	Reg.HabilitadordeInterrupção	-
2EAh	3EAh	2FAh	3FAh	IIR	Reg. Identificador de lnterrup.	-
2EBh	3EBh	2FBh	3FBh	LCR	Reg. Controlador de Linhas	-
2ECh	3ECh	2FCh	3FCh	MCR	Reg. Controlador de Modem	-
2EDh	3EDh	2FDh	3FDh	LSR	Reg. de Estado de Linha	-
2EEh	3EEh	2FEh	3FEh	MSR	Reg. de Estado de Modem	-
2EFh	3EFh	2FFh	3FFh	SCR	Reg. de uso geral	-

Tabela 5.3.1

5.3.1 TXB – Buffer de Transmissão

Neste registrador, deve-se escrever o byte a ser transmitido bit a bit pelo 8250. Notar que ele só é acessado quando o bit DLAB (bit7 do LCR) estiver em zero.

5.3.2 RXB – Buffer de Recepção

Neste registrador, é lido o byte que chegou bit a bit pelo canal serial. Notar que ele só é acessado quando o bit DLAB (bit 7 do LCR) estiver em zero.

5.3.3 DLL – Latch para o LSB do Divisor

No DLL, escreve-se o byte menos significativo do número pelo qual se quer dividir o relógio que é fornecido ao 8250; ou seja, programa-se, junto com o DLM, a velocidade de comunicação (ver figura 13.14). Este registrador só é acessado quando o bit DLAB (bit 7 do LCR) estiver em 1.

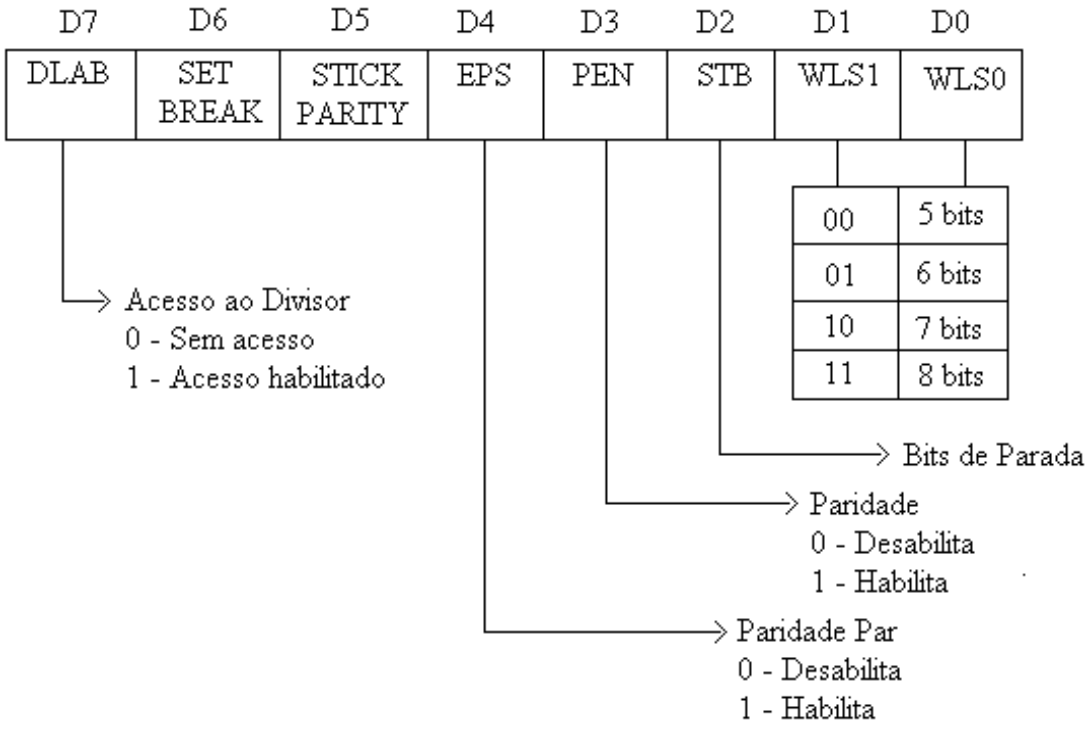
5.3.4 DLM - Latch para o MSB do Divisor

No DLM, escreve-se o byte mais significativo do número pelo qual se quer dividir o relógio que é fornecido ao 8250, isto para estabelecer-se a velocidade de comunicação. Esse registrador só é acessado quando o bit DLAB (bit 7 do LCR) estiver em 1. Através da programação dos valores de DLM e DLL, geram-se as diversas taxas de transmissão:

Taxa	N	DLM (em hexa)	DLL (em hexa)
50	2304	09	00
2400	48	00	30
9600	12	00	0C
115200	1	00	01

5.3.5 - LCR Registrador de Controle de linha

No LCR, especifica-se o formato da comunicação assíncrona. Este registrador também pode ser lido, o que facilita o controle da porta serial. A descrição dos seus bits está na figura 5.3.5.1



F*igura 5.3.5.1*

D0 e D1 : Especificam a quantidade de bits em cada caracter transmitido
D2: Especifica a quantidade de bits de parada

D2	D1 e D0	Bits de Parada
0	Não importa	1 stop
1	(00) 5 bits	1 ½ stop
1	6,7,8 bits	2 stops

D3 - (PEN): Este é o bit habilitador de paridade. Quando ativado, um bit de paridade é gerado, na transmissão, ou verificado, na recepção. O bit de paridade está entre o último bit do campo de dado e o bit da parada. Esse bit é colocado em 1 ou 0, de forma a gerar um número par ou ímpar de "1's".

D4 - (EPS): Este é o bit selecionador de paridade par. A figura 13.18 apresenta uma tabela que mostra as paridades geradas para cada caso.

D3 (PEN)	D4 (EPS)	Ação
1	0	Número ímpar de 1s transmitidos ou verificados
1	1	Número par de 1s transmitidos ou verificados

D5 - *Stick Parity bit:* Especifica o nível lógico do bit de paridade a ser transmitido. A figura 13.18 apresenta uma tabela ilustrando o uso deste bit.

D6 - "Set *Break":* Este bit controla a ativação do *"break line"*. Quando colocado em 1, a saída serial é forçada para o estado espaço (+12 V) e assim permanece, independente de qualquer outra atividade do transmissor. O *"break line"* é desativado ao colocar-se esse bit em 0.

D7 (DLAB) : Este é o bit que controla o acesso ao latch do divisor:

DLAB = 1 - permite o acesso aos latches do divisor que gera a taxa de comunicação,

DLAB = 0 - permite o acesso aos buffers de transmissão e recepção.

Normalmente, esse bit é colocado em 1 apenas uma vez, ao programar-se a taxa de comunicação e depois é deixado em 0, permanecendo assim durante toda a operação, permitindo portanto acesso aos registradores TXB e RXB.

5.3.6 - LSR Registrador de Estado de linha

Este é um registrador de 8 bits onde estão as informações que dizem respeito à transferência de dados. Sua descrição é feita na figura 5.3.6.1.

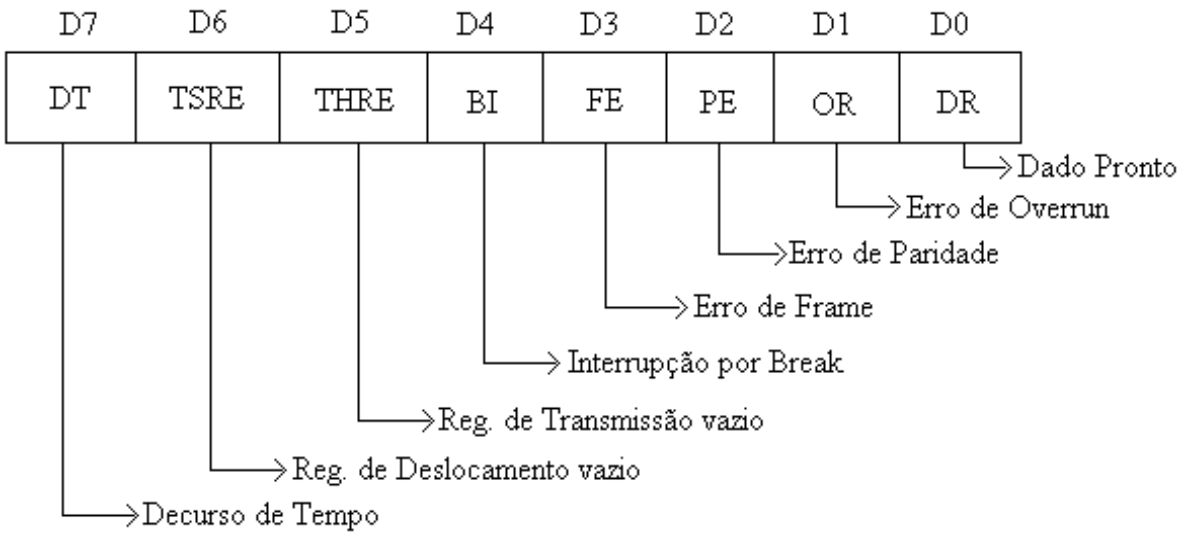


figura 5.3.6.1

D0 (DR): Esse é um indicador de que foi recebido um dado, ou seja, que existe um dado já pronto no buffer. Esse bit vai a 1 sempre que for completada uma recepção e o dado for transferido para o buffer e vai a 0 quando o processador ler o dado recebido, ou quando se escreve neste registrador com o bit DR = 0.

Dl (OE): Este é o bit de erro de *"overrun"*. Ele indica que o dado que estava no buffer ainda não havia sido lido quando o dado seguinte foi escrito e destruiu o antigo. Este bit é zerado toda vez que o LSR é lido.

D2 (PE): Este é o bit de erro de paridade. Ele indica que o caracter recebido não tem a paridade correta. O bit PE vai a 1 quando é detectado um erro de paridade e é zerado quando a CPU ler o LSR.

D3 (FE): Este é o bit de erro de *"frame"*. Ele indica que o caracter recebido não tinha um bit de parada válido. Esse bit vai a 1 sempre que o bit que se segue ao último bit de dado for detectado como espaço (+12V) e é zerado com a leitura LSR.

D4 (BI): Este é o indicador de *"line break"*. Ele vai a 1 sempre que a entrada de dados for mantida em espaço (+12 V) por um período de tempo maior que o necessário para transmitir uma palavra completa e é zerado com a leitura do LSR.

D5 (THRE): Este bit indica que o registrador de transmissão está vazio. Ou seja, significa que o 8250 está pronto para aceitar um novo caracter para transmissão. O bit THRE vai para 1 quando o byte for transferido do registrador de transmissão para o registrador de deslocamento, que é o local onde se serializa o byte a ser transmitido, e vai a 0 quando se escrever no registrador de transmissão.

D6 (TSRE): Indica que o registrador de deslocamento está vazio. Este bit vai para 1 sempre que o registrador de deslocamento estiver ocioso e é zerado quando se transferir um byte do registrador de transmissão para o registrador de deslocamento. O bit TSRE é apenas de leitura.

D7 (DT): Este bit acusa o decurso de tempo nas operações seriais.

```
/* Rotina para transmitir caracter */

while (!(inportb(0x2FD) & 0x20)); /* aguarda o bit THRE ir a 1*/
outportb(0x2F8, caracter);/* escreve no reg. de transmissão */

/* Rotina para receber caracter */
while (!(inportb(0x2FD) & 0x01)); /* aguarda o bit DR ir a 1*/
caracter = inportb(0x2F8);    /* lê dado recebido */
```

5.3.7 - lIR Reg. Identificador de Interrupção

O 8250 possui recursos de interrupção que permitem uma grande flexibilidade no interfaceamento com a maioria dos microprocessadores. Ele prioriza as interrupções em 4 níveis:

Prioridade	Interrupção
1	Estado de linha do receptor
2	Dado recebido
3	Reg. de transmissão vazio
4	Estado do modem

A informação de que existe uma interrupção pendente e a identificação dessa interrupção são fornecidas pelo IIR. O registrador IR, quando endereçado, congela a interrupção pendente de mais alta prioridade e não aceita outras interrupções até que aquela seja servida. A figura 5.3.7.1 apresenta este registrador.

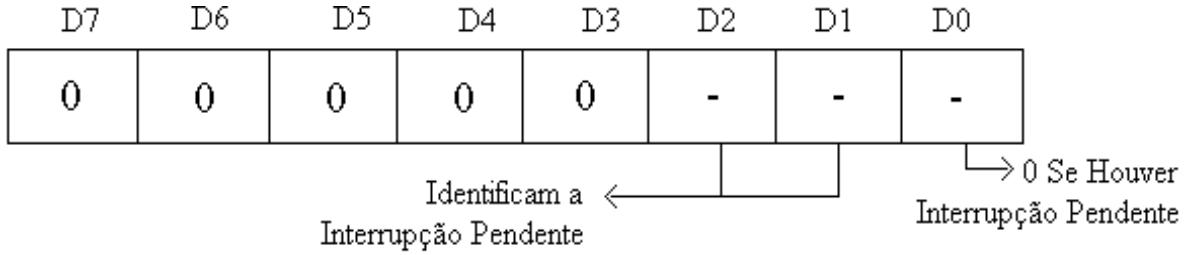


Figura 5.3.7.1. Descrição do registrador IIR.

D0: Este bit indica se existe uma interrupção pendente. Quando este bit estiver ativado, significa que não existe interrupção pendente.

Dl E D2: Identificam a interrupção pendente de prioridade mais alta. Vide tabela

D2	D1	D0	Prior.	Tipo de Int.	Org. da Int.	Reset da Int.
0	0	1	--	Nenhuma	Nenhuma	---
1	1	0	1 (mais alta)	Estado de Linha do Receptor	Erro Overrun Erro Paridade Erro Framing Break lnterrupt	Lendo o Registrador LSR
1	0	0	2	Dado Recebido	Dado Recebido	Lendo o Dado
0	1	0	3	THR Vazio	THR Vazio	Lendo IIR ou Escrevendo em THR
0	0	0	4	Estado do Modem	CTS, DSR, RI, CD	Leitura do MSR

5.3.8 - IER Reg. Habilitador de Interrupção

Este registrador permite habilitar e desabilitar, individualmente, as interrupções do 8250. Ao desabilitarem-se as interrupções, inibe-se o registrador IIR; porém, todas as outras funções operam da maneira normal. A figura 5.3.8.1 mostra como habilitar ou desabilitar interrupções geradas pelo recebimento de um caracter, por ocasião do esvaziamento do registrador de transmissão (THR), por erro de transmissão, ou por alteração do estado do modem.

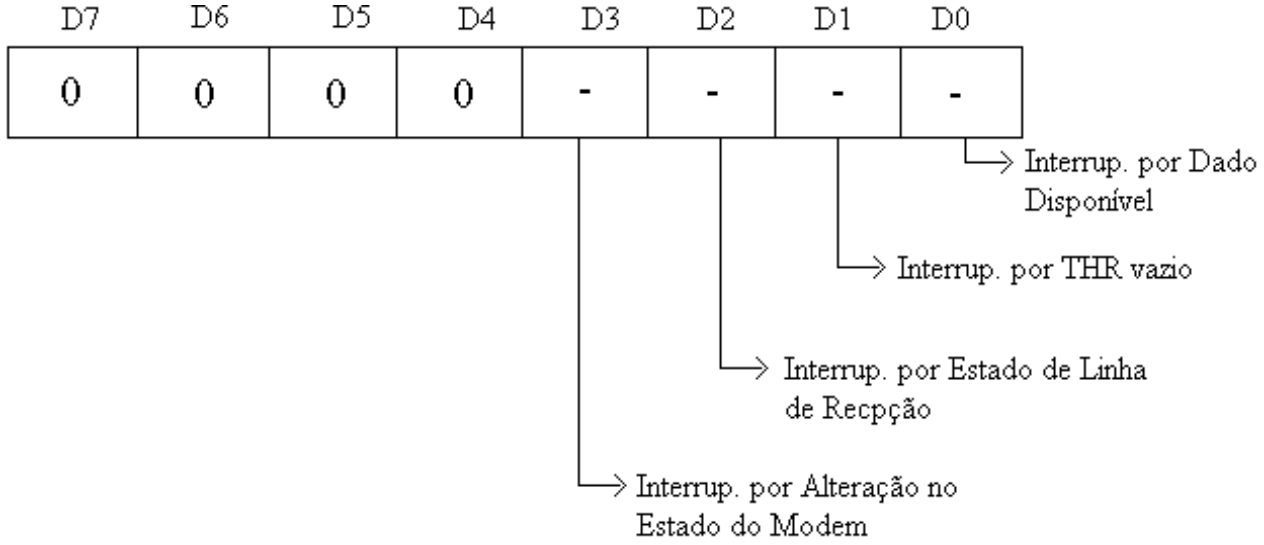


Figura 5.3.8.1

5.3.9 - MCR Reg. de Controle de Modem

Por esse registrador, controla-se a interface com o modem. A descrição desses pinos está na figura 5.3.9.1.

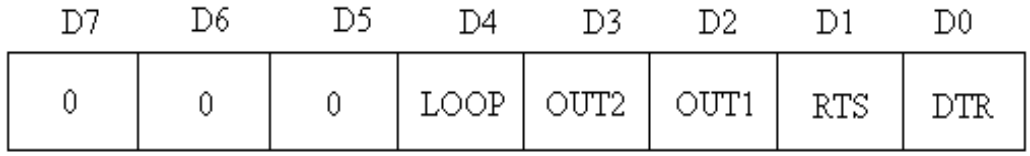


Figura 5.3.9.1

D0 (DTR): Este bit controla a saída DTR. Quando se coloca o bit DTR em 0, a saída vai para -12V. Com este bit em 1, a saída vai para +12V (ativada).

Dl (RTS): Este bit controla a saída RTS, de forma idêntica ao anterior.

D2 (OUT): Este bit controla a saída *OUT1 do 8250, invertendo o valor programado.

D3 (OUT2): Esse bit controla a saída *OUT2 do 8250, invertendo o valor programado. No PC, quando ativado, permite que o pedido de interrupção do 8250 seja diretamente conectado à IRQ alocada para a porta serial, geralmente a IRQ3 ou IRQ4.

D4: Este bit controla o recurso de diagnóstico chamado de *"loop back"*.

No modo diagnóstico *("Ioop back"),* o dado transmitido é imediatamente recebido. Isto permite checar os caminhos de transmissão e de recepção do 8250. Nesse modo, se OUT2 = 1, todas as interrupções estão operacionais e podem ser testadas.

5.3.10 - MSR Reg. de Estado do Modem

Este registrador indica o estado das linhas de controle do modem. Além de indicar o estado, também pode indicar uma mudança de estado desde a última leitura. Existem bits que vão para 1 quando há uma alteração na entrada e que são zerados por ocasião da leitura. A figura 5.3.10.1 ilustra este registrador.

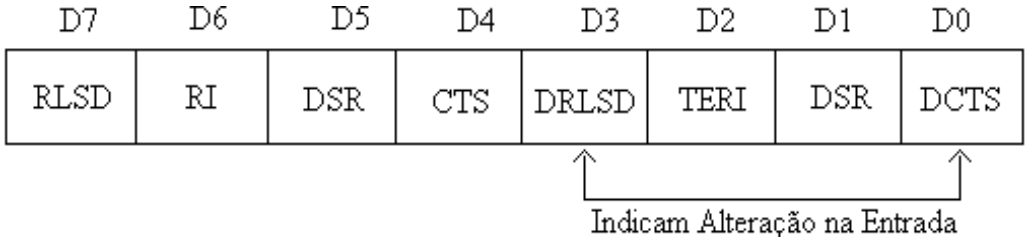


Figura 5.3.10.1

D0 (DCTS): Quando ativado, indica que a entrada CTS mudou de estado desde a última leitura. É zerado por ocasião da leitura deste registrador.

Dl(DDSR): Funcionamento idêntico ao anterior, mas com a entrada DSR.

D2 (TREI): Este bit indica o fim do sinal do detector de tom. Quando ativado, ele

indica que a entrada RI passou do estado ON (+12V) para o estado 0FF (-12 V).

D3 (DRLSD OU DCD): Quando ativado, indica que a entrada CD (RLSD) mudou de estado desde a última leitura. E zerado por ocasião da leitura deste registrador.

D4 (CTS): Indica o estado da entrada CTS. Se CTS = +12 V, então D4 = 1.

D5 (DSR): Idêntico ao anterior, para a entrada DSR.

D6 (RI): Idêntico ao anterior, para a entrada RI.

D7 (RLSD OU CD): Idêntico ao anterior, para a entrada CD.

5.3.11 - SCR "Scratchpad Register"

SCR é um registrador que não influencia na operação do 8250. Mas é bastante utilizado para verificar se existe porta serial conectada ao PC. Para isto, escreve-se um dado nos possíveis endereços deste registrador; se o valor lido for igual ao escrito, há um forte indício da presença da porta serial.

5.4 Interrupção

A interrupção é um recurso muito interessante para diminuir a sobrecarga de software em uma comunicação serial No projeto do PC, geralmente, as portas COM1 e COM3 podem gerar a interrupção IRQ4 (INT 0Ch no DOS e 54h nos sistemas de 32 bits) e, então, para que não haja conflitos no barramento, apenas um dos bits OUT2, referentes aos circuitos da COM1 e COM3, deve estar ativado. O mesmo cuidado deve ser tomado com as portas COM2 e COM4, isto em relação à IRQ3 (INT 0Bh no DOS e 53h nos sistemas de 32 bits).

5.5 – Programa Exemplo para identificar a Porta Serial

Indent.c

```
#define COM1 1
#define COM2 2
#define COM3 3
#define COM4 4
#define COM1D  0x3F8
#define COM1N  0x3F9
#define COM1F  0x3FA
#define COM1B  0x3FB
#define COM1C  0x3FC
#define COM1I  0x3FD
#define COM1R  0x3FE
#define COM2D  0x2F8
#define COM2N  0x2F9
#define COM2F  0x2FA
#define COM2B  0x2FB
#define COM2C  0x2FC
#define COM2I  0x2FD
#define COM2R  0x2FE
#define COM3D  0x3E8
```

```
#define COM3N  0x3E9
#define COM3F  0x3EA
#define COM3B  0x3EB
#define COM3C  0x3EC
#define COM3I  0x3ED
#define COM3R  0x3EE
#define COM4D  0x2E8
#define COM4N  0x2E9
#define COM4F  0x2EA
#define COM4B  0x2EB
#define COM4C  0x2EC
#define COM4I  0x2ED
#define COM4R  0x2EE
#define MASK_LCR_QTD 0x3
#define MASK_LCR_STB 0x4
#define MASK_LCR_PEN 0x8
#define MASK_LCR_EPS 0x10
#define COM1_SCR 0x3FF
#define COM2_SCR 0x2FF
#define COM3_SCR 0x3EF
#define COM4_SCR 0x2EF
#define IRQ3   0x0B
#define IRQ4   0x0C
#define IRQ5   0x0D
#define IRQ7   0x0F
#define TXRDY  0x20
#define RBR    0x01
#define OUT2   0x08
#define TEMPO  5

  int identifica (int port);
  void interrupt (*oldint3)();
  void interrupt (*oldint4)();
  void interrupt (*oldint5)();
  void interrupt (*oldint7)();

static int addr,testint;
int old_LCR;
struct _config
  {  long int baud;
     int tam_buf;
     float stop_bit;
     int paridade;
     int paridade_par;
     int irq;  } config;
void interrupt tint3()
{    switch (inp(addr | 0x02) & 0x06) {
     case 0:inp(addr | 0x06);
            break;
     case 2:inp(addr | 0x02);
            break;
     case 4:inp(addr);
            break;
     case 6:inp(addr | 0x03);
            break;     }
     testint = 3;
     outp(0x20, 0x20);}
void interrupt tint4()
{    switch (inp(addr | 0x02) & 0x06) {
     case 0:inp(addr | 0x06);
            break;
     case 2:inp(addr | 0x02);
```

```
            break;
     case 4:inp(addr);
            break;
     case 6:inp(addr | 0x03);
            break;     }
     testint = 4;
     outp(0x20, 0x20);}
void interrupt tint5()
{    switch (inp(addr | 0x02) & 0x06) {
     case 0:inp(addr | 0x06);
            break;
     case 2:inp(addr | 0x02);
            break;
     case 4:inp(addr);
            break;
     case 6:inp(addr | 0x03);
            break;     }
     testint = 5;
     outp(0x20, 0x20);}
void interrupt tint7()
{    switch (inp(addr | 0x02) & 0x06) {
     case 0:inp(addr | 0x06);
            break;
     case 2:inp(addr | 0x02);
            break;
     case 4:inp(addr);
            break;
     case 6:inp(addr | 0x03);
            break;     }
     testint = 7;
     outp(0x20, 0x20);}

 int detc_int(COMD,COMN,COMF,COMB,COMC,COMI,COMR)
    {int enab,bits,test1,mask,mask_test,byte,old_COMC,old_COMN;
     char *env;
     oldint3 = getvect(IRQ3);
     oldint4 = getvect(IRQ4);
     oldint5 = getvect(IRQ5);
     oldint7 = getvect(IRQ7);
     setvect(IRQ3, tint3);
     setvect(IRQ4, tint4);
     setvect(IRQ5, tint5);
     setvect(IRQ7, tint7);
/*  Existem perif,ricos seriais que necessitam que o CTS
     (ou o DSR) deles seja setado para que a mensagem serial
     seja emitida. O comando "set MASK=1" faz com que o DTR
     das   portas seriais seja setado, "set MASK=2" liga o RTS
     e "set MASK=3" liga os dois. Basta conectar o DTR (ou o
     RTS) do computador ao CTS (ou o DSR) do periferico. */
     bits = OUT2;
/*  Enable nas quatro interrup‡oes (3, 4, 5 e 7) */
     outp(0x20, 0x20);
     mask = inp(0x21);
     mask_test = mask & 0x47;
     outp(0x21, mask_test);
     enab = 0xFF;
/*  Descobrir qual IRQ est  associada … COM1 */
     addr = COMD;
     testint = 0;
/* Limpar a porta */
     inp(COMD);
     inp(COMF);
```

```
      inp(COMI);
      inp(COMR);
      byte=inp(COMB);         /* Obter os settings da COM1 */
      outp(COMB,byte&0x7f);   /*Setar o(s) bit(s) necess rio(s) */
      old_COMC=inp(COMC);
      outp(COMC, inp(COMC) | bits);
      old_COMN=inp(COMN);  /*Programar a porta para gerar
                             interrupcao Qdo um caracter
                             tiver sido transmitido */
      outp(COMN, 0x03);     /*Aguardar que a aporta esteja
                              pronta para transmitir */
      while (!(inp(COMI) & TXRDY));  /* Emitir um caracter (0x07) */
         outp(COMD, 0x07);       /* Aguardar 5 milisegundos */
         delay(TEMPO);           /* Se ocorreu interrupçao, o número
                                    dela estará na variável
                                    global testint */
      test1 = testint;
      outp(COMN,old_COMN);
      outp(COMC,old_COMC);
      outp(COMB,byte);
      outp(0x21,mask);
      setvect(IRQ3, oldint3);
      setvect(IRQ4, oldint4);
      setvect(IRQ5, oldint5);
      setvect(IRQ7, oldint7);
      return(test1);}

int main(int argc, char *argv[])
 {  int existe;
    if (!strcmp(argv[1],"COM1"))
       {   existe=identifica(COM1);
           outp(COM1B,old_LCR);      }
    if (!strcmp(argv[1],"COM2"))
       {   existe=identifica(COM2);
           outp(COM2B,old_LCR);      }
    if (!strcmp(argv[1],"COM3"))
       {   existe=identifica(COM3);
           outp(COM3B,old_LCR);      }
    if (!strcmp(argv[1],"COM4"))
       {   existe=identifica(COM4);
           outp(COM4B,old_LCR);      }

  if(!existe) exit(0);
  printf("Porta %s\n", argv[1]);
  printf("  baud: %ld\n", config.baud);
  printf("  buffer: %d\n", config.tam_buf);
  printf("  stop bit: %.1f\n", config.stop_bit);
  if(config.paridade==0)
     { printf("  paridade: inexistente\n");}
     else
     if(config.paridade_par==1)
              { printf("  paridade: par\n");  }
  else printf("  paridade: impr\n");
  printf("  IRQ: %d\n", config.irq);
  return(0); }

int identifica (int port)
 {  int valor,mask;
    int addr_LCR,addr_SCR;
    int addr_DLL, addr_DLM;
    int DLL,DLM;
    int LCR;
```

```
  switch (port)
     { case COM1:old_LCR=inp(COM1B);
                 addr_LCR = COM1B;
                 addr_DLL = COM1D;
                 addr_DLM = COM1N;
                 addr_SCR = COM1_SCR;
           config.irq=detc_int(COM1D,COM1N,COM1F,COM1B,COM1C,COM1I,COM1
       R);
                 break;
       case COM2: old_LCR=inp(COM2B);
                 addr_LCR = COM2B;
                 addr_DLL = COM2D;
                 addr_DLM = COM2N;
                 addr_SCR = COM2_SCR;

config.irq=detc_int(COM2D,COM2N,COM2F,COM2B,COM2C,COM2I,COM2R);
                 break;
       case COM3: old_LCR=inp(COM3B);
                 addr_LCR = COM3B;
                 addr_DLL = COM3D;
                 addr_DLM = COM3N;
                 addr_SCR = COM3_SCR;

      config.irq=detc_int(COM3D,COM3N,COM3F,COM3B,COM3C,COM3I,COM3R);
                 break;
       case COM4: old_LCR=inp(COM4B);
                 addr_LCR = COM4B;
                 addr_DLL = COM4D;
                 addr_DLM = COM4N;
                 addr_SCR = COM4_SCR;

      config.irq=detc_int(COM4D,COM4N,COM4F,COM4B,COM4C,COM4I,COM4R);
                 break; }
/* ---------- verifica a existencia da porta -------------*/
  outportb(addr_SCR,1);
  valor=inportb(addr_SCR);
  if(valor!=1)
{ printf("Porta nao identificada!");
       return(0);}
/* ----------------------------------------------------------*/
  valor = inport(addr_LCR);
  valor= valor | 0x80;
  outport(addr_LCR,valor);
  DLL = inport(addr_DLL);
  DLM = inport(addr_DLM);
  if (DLM==257) DLM=384;
  config.baud = 115200/DLL;

  valor = inport(addr_LCR);
/*  -----------    tamanho do buffer   ------------------- */
  mask = valor | MASK_LCR_QTD;
  mask-=valor;
  switch(mask)
       {  case 0:  config.tam_buf=8;
                 break;
        case 1:    config.tam_buf=7;
                 break;
        case 2:   config.tam_buf=6;
                 break;
        case 3:   config.tam_buf=5;
                 break;  }
  mask=0;
```

```
/*  ----------------- Stop bits  ------------------------ */
  mask = valor | MASK_LCR_STB;
  mask-=valor;
  switch(mask)
       {   case 0:if(config.tam_buf>5) config.stop_bit=2;
                  else config.stop_bit=1.5;
                  break;
           case 4: config.stop_bit=1;
                   break;  }
  mask=0;
/*  ------------------ Paridade ------------------------ */
  mask = valor |  MASK_LCR_PEN;
  mask-=valor;
    switch(mask)
       {   case 0:config.paridade=1;
                  break;
            case 8:config.paridade=0;
                   break; }
  mask=0;
/*  ----------------- Paridade - par/impar -------------- */
  if(valor & MASK_LCR_PEN)
     { if(valor & MASK_LCR_EPS)  config.paridade_par=1;
       else  config.paridade_par=0;
  } else config.paridade_par=0;
  mask=0;
  return(1); }
```

Exercício Proposto 3:

Desenvolver um programa em C para alterar as configurações da COM1 .
(Tamanho do Buffer, no de Stop Bits, Paridade)

Exercício Proposto 4:

Desenvolver um programa em C para enviar sinais para porta paralela de acordo com o movimento do mouse .

Ex.: movimento para cima => Sinal para o pino 2.
movimento para baixo => Sinal para o pino 3.
movimento para direita => Sinal para o pino 4.
movimento para esquerda => Sinal para o pino 5.

Apêndice A: Linguagem C

1. C Versus C++

Algumas vezes os novatos confundem o que é C++ e como difere de C. Para ser breve, C++ é uma versão estendida e melhorada de C que é projetada para suportar programação orientada a objetos (OOP, do inglês Object Oriented Programming). C++ contém e suporta toda a linguagem C e mais um conjunto de extensões orientadas a objetos. (Ou seja, C++ é um superconjunto de C.)

Hoje em dia, e por muitos anos ainda, a maioria dos programadores ainda escreverá, manterá e utilizará programas C, e não C++. Como mencionado, C suporta programação estruturada. A programação estruturada tem-se mostrado eficaz ao longo dos 25 anos em que tem sido usada largamente. C++ é projetada principalmente para suportar OOP, que incorpora os princípios da programação estruturada, mas inclui objetos. Embora a OOP seja muito eficaz para uma certa classe de tarefas de programação, muitos programas não se beneficiam da sua aplicação. Por isso, "código direto em C" estará em uso por muito tempo ainda.

2. Um Compilador C++ Funcionará com Programas C?

Hoje em dia é difícil ver um compilador anunciado ou descrito simplesmente como um "compilador C". Em vez disso, é comum ver um compilador anunciado como "compilador C/C++", ou às vezes simplesmente como compilador C++. Esta situação faz surgir naturalmente a pergunta: "Um compilador C++ funcionará com programas C?". A resposta é: "Sim!". Qualquer um e todos os compiladores que podem compilar programas C++ também podem compilar programas C. Portanto, se seu compilador é denominado um "compilador C++", não se preocupe, também é um compilador C.

3. Compiladores *Versus* Interpretadores

Os termos *compiladores* e *interpretadores* referem-se à maneira como um programa é executado Existem dois métodos gerais pelos quais um programa pode ser executado. Em teoria, qualquer linguagem de programação pode ser compilada ou interpretada, mas algumas linguagens geralmente são executadas de uma maneira ou de outra Por exemplo, BASIC é normalmente interpretada e C, compilada. Interpretadores e compiladores são simplesmente programas sofisticados que operam sobre o código-fonte do seu programa Como a diferença entre um compilador e um interpretador pode não ser clara para todos os leitores, a breve descrição seguinte esclarecerá o assunto.

Um interpretador lê o código-fonte do seu programa uma linha por vez, executando a instrução específica contida nessa linha Um compilador lê o programa inteiro e converte-o em um *código-objeto,* que é uma tradução do código-fonte do programa em uma forma que o computador possa executar diretamente. O código-

objeto é também conhecido como código binário ou código de máquina Uma vez que o programa tenha sido compilado, uma linha do código-fonte, mesmo alterada, não é mais importante na execução do seu programa.

Quando um interpretador é usado, deve estar presente toda vez que você executar o seu programa Por exemplo, em BASIC você precisa primeiro executar o interpretador, carregar seu programa e digitar RUN cada vez que quiser usá-lo. O interpretador BASIC examina seu programa uma linha por vez para correção e então executa-o. Esse processo lento ocorre cada vez que o programa for executado. Um compilador, ao contrário, converte seu programa em um código-objeto que pode ser executado diretamente por seu computador. Como o compilador traduz seu programa de uma só vez, tudo o que você precisa fazer é executar seu programa diretamente, geralmente apenas digitando seu nome. Assim, o tempo de compilação só é gasto uma vez, enquanto o código interpretado incorre neste trabalho adicional cada vez que o programa executa.

Apêndice B: INT33

1. Serviços oferecidos pela INT33

Serviço: 00h
Descrição: Inicializa o mouse.
Entrada: AX = 0000h
Saldas:
AX = 0000h (mouse não instalado) ou FFFFh (mouse instalado)
BX = no de botões

Serviço: 0lh
Descrição: Mostra o cursor no Vídeo.
Entrada: Ax = 000lh

Serviço: 02h
Descrição: Esconde o cursor, não o mostrando no Vídeo.
Entrada: AX = 0002h

Serviço: 03h
Descrição: Obtém estado dos botões e posição do cursor.
Entrada: AX = 0003h
Saldas:
BX = estado dos botões (bit0 - esquerdo, bit1 - direito, bit2 - do meio; 1 = pressionado,0= solto)
CX = coordenada X
DX = coordenada Y

Serviço: 04h
Descrição: Seta posição do mouse.
Entradas:
AX = 0004h
CX = nova coordenada X
DX = nova coordenada Y

Serviço: 05h
Descrição: Obtém informações sobre últimos diques.
Entradas:
AX = 0005h
BX = botão (0 - esquerdo, 1 - direito, 2 - do meio)
Saldas:
AX = estado dos botões (bit0 - esquerdo, bit1 - direito, bit2 - do meio; 1 = pressionado, 0=solto)
BX = nº de vezes que o botão foi pressionado(este nº é zerado após a chamada desta função)
CX = coordenada X do último dique
DX = coordenada Y do último dique

Serviço: 06h
Descrição: Obtém informações sobre últimas vezes que o botão foi solto.
Entradas:
AX = 0006h
BX = botão (0 - esquerdo, 1 - direito, 2 - do meio)
AX = estado dos botões (bit0 - esquerdo, bit1 - direito, bit2 - do meio; 1 = pressionado, 0=solto)
BX = nº de vezes que o botão foi liberado (este nº é zerado após a chamada desta função)
CX = coordenada X da última liberação do botão
DX = coordenada Y da última liberação do botão

Serviço: 07h
Descrição: Restringe varredura horizontal do cursor.
Entradas:
AX = 0007h
CX = valor mínimo para a coordenada X
DX = valor máximo para a coordenada X

Serviço: 08h
Descrição: Restringe varredura vertical do cursor.
Entradas:
AX = 0006h
CX = valor mínimo para a coordenada Y
DX = valor máximo para a coordenada Y

Serviço: 09h
Descrição: Define o cursor gráfico e a referência do ponto das coordenadas.
Entradas:
AX = 0009h
BX = coordenada X em relação ao cursor
CX = coordenada Y em relação ao cursor
ES:DX = endereço onde está definido o cursor:

16 words - máscara
16 words - cursor

Serviço: 0Ah
Descrição: Define o cursor texto.
Entradas:
AX = 000Ah
BX = tipo do cursor (0 - software, 1 - hardware)
CX = máscara da tela (tipo software) ou linha inicial de varredura (tipo hardware)
DX = máscara do cursor (tipo software) ou linha final de varredura (tipo hardware)

Serviço: 0Bh
Descrição: Retorna deslocamento do mouse desde a última chamada à função.
Entrada: AX = 000Bh
Saídas:
CX = deslocamento horizontal, em 1/200 de polegada (mouse de 200 ppi) ou 1/400 de polegada (mouse de 400 ppi)
DX = deslocamento vertical, em 1/200 de polegada (mouse de 200 ppi) ou 1/400 de polegada (mouse de 400 ppi)
Interrupção: 33h

Serviço: 0Ch
Descrição: Define uma rotina de interrupção a ser tratada toda vez que for constatada uma condição definida no byte de máscara.
Entradas:
AX = 000Ch
CL = máscara
Bit0 - alteração da posição do cursor bit1 - pressão do botão esquerdo bit2 - liberação do botão esquerdo bit3 - pressão do botão direito bit4 - liberação do botão direito bit5 - pressão do botão do meio bit6 - liberação do botão do meio
ES:DX = endereço da subrotina
Observação: Para a subrotina construída, os seguintes valores são passados:
AX = máscara com o bit relativo à condição geradora da interrupção setado
BX = estado dos botões
CX = coordenada X do cursor
DX = coordenada Y do cursor
SI = deslocamento horizontal do mouse
Dl = deslocamento vertical do mouse
Interrupção: 33h

Serviço: 0Dh
Descrição: Faz o mouse emular uma caneta ótica.
Entrada: AX = 000Dh

Serviço: 0Eh
Descrição: Faz o mouse deixar de atuar como uma caneta ótica.
Entrada: AX = 000Eh

Serviço: 0Fh

Descrição: Define sensibilidade do mouse.
Entradas:
AX = 000Fh
CX = taxa horizontal
DX = taxa vertical

Serviço: 10h
Descrição: Define condições para esconder cursor.
Entradas:
AX = 0010h
CX = limite à esquerda da coordenada X (condição = X <CX)
DX = limite superior da coordenada Y (condição = Y < DX)
SI = limite à direita da coordenada X (condição = X > SI)
Dl = limite inferior da coordenada Y (condição = Y> DI)

Serviço: 13h
Descrição: Define limiar de velocidade do mouse.
Entradas:
AX = 0013h
DX = limiar de velocidade

Serviço: 14h
Descrição: Substitui rotina de interrupção definida no serviço 0Ch.
Entradas:
AX = 0014h
ES:DX = endereço da nova subrotina
CX = nova máscara (como no serviço 0Ch)
Saídas:
ES:DX = endereço da subrotina substituída
CX = máscara substituída

Serviço: 15h
Descrição: Obtém n^{o} de bytes gastos na instalação do driver.
Entrada: AX = 001 5h
Saída: BX = no de bytes

Serviço: 18h
Descrição: Define máscara alternativa e uma rotina de interrupção a ser tratada toda vez que for constatada uma condição especial.
Entradas:
AX = 001Bh
CX = máscara especial
Bit0 - alteração da posição do cursor bit1 - pressão do botão esquerdo bit2 - liberação do botão esquerdo bit3 - pressão do botão direito bit4 - liberação do botão direito
bit5 - Shift pressionado e botão pressionado ou liberado
bit6 - Ctrl pressionado e botão pressionado ou liberado bit7 - Alt pressionado e botão pressionado ou liberado
ES:DX = endereço da subrotina
Saída: AX = -1 (erro)

Observação: Para a subrotina construída, os seguintes valores são passados:
AX = máscara com o bit relativo à condição geradora da interrupção setado
BX = estado dos botões
CX = coordenada X do cursor
DX = coordenada Y do cursor
SI = deslocamento horizontal do mouse
Dl = deslocamento vertical do mouse

Serviço: 19h
Descrição: Obtém endereço da rotina de interrupção definida no serviço 18h.
Entradas:
AX = 0019h
CX = nova máscara alternativa (como no serviço 1 8h)
Saldas:
AX = -1 (erro)
BX:DX = endereço da subrotina
CX = máscara alternativa substituída

Serviço: lAh
Descrição: Define sensibilidade do mouse.
Entradas:
AX = 00lAh
BX = sensibilidade horizontal (x 1/200 " - mouse 200 ppi, x 1/400 " - mouse 400 ppi)
CX = sensibilidade vertical (X 1/200 " - mouse 200 ppi, x 1/400" - mouse 400 ppi)
DX = limiar de velocidade

Serviço: lBh
Descrição: Obtém sensibilidade do mouse.
Entrada: AX = 001 Bh
Saldas:
BX = sensibilidade horizontal (x 1/200 - mouse 200 ppi, x 1/400 "- mouse 400 ppi)
CX = sensibilidade vertical (x 1/200 " - mouse 200 ppi, x *1/400"* - mouse 400 ppi)
DX = limiar de velocidade

Serviço: lCh
Descrição: Seta taxa de interrupção do mouse.
Entradas:
AX = 00lCh
BX = no de interrupções por segundo (0 - sem interrupção, 1 - 30,2 - 50,3 - 100,4 – 200)

Serviço: lFh
Descrição: Desabilita driver do mouse.
Entrada: AX = 001Fh
Saldas:
AX = -1 (erro)
ES:BX = endereço do vetor de interrupção da INT 33h associado ao driver

Serviço: 20h
Descrição: Habilita driver do mouse.

Entrada: AX = 0020h

Serviço: 21h
Descrição: Reseta software.
Entrada: AX = 0021 h
Salda: AX = -1 (driver instalado), 21 h (caso contrário)

Serviço: 24h
Descrição: Obtém versão do driver, tipo do mouse e IRO associada.
Entrada: AX = 0024h
Saldas:
BX = versão em BCD
CH = tipo (1 - mouse de barramento, 2 - mouse serial, 4 - PS/2)
CL = IRO
O driver, geralmente, inicializa seus parâmetros com os seguintes valores:
Posição do cursor: centro da tela;
Cursor escondido;
Formato do cursor: uma seta (gráfico) ou um quadrado (texto);
Máscara: todos os bits ressetados;
Caneta ótica: emulada;
Região de varredura: toda a tela;
Deslocamento horizontal: 8 1/200" por 8 pixels;
Deslocamento vertical: 16 1/200" por 8 pixels;
Limiar de velocidade: 64 1/200" por segundo.

Apêndice C: Biblioteca “Graphics.h”

1. Graphics.h

A biblioteca <graphics.h> definida no compilador Borland Turbo C possui funções para permitir o uso de recursos gráficos em ambientes DOS.

<graphics.h>

arc	getpalette	getpalettesize		setallpalette
bar	getcolor	textheight	line	setaspectratio
bar3d	getdefaultpalette	getpixel	linerel	setbkcolor
circle	getdrivername	gettextsettings	lineto	setcolor
cleardevice	getflllpattern	getviewsettings	moverel	setfillpattern
clearviewport	getfillsettings	getx	moveto	setfillstyle
closegraph	getgraphmode	gety	outtext	setgraphbufsize
detectgraph	getimage	graphdefaults	outtextxy	setgraphmode
drawpo1y	getlinesettings	grapherrormsg	pieslice	setlinestyle
ellipse	getmaxcolor	_graphfreemem	putimage	setpalette
fillellipse	getmaxmode	_graphgetmem	putpixel	setrgbpalette
fillpoly	getmaxx	graphresult	rectangle	settextjuatify
floodfill	getmaxy	imagesize	registerbgidriver	settextstyle
getaracoords	getmodename	initgraph	sector	setusercharaize

getaspectratio	getmoderange	initalluserdriver	setactivepage	setviewport
getbkcolor	setwritemode	installuserfont	setvisualpage	textwidth

Dica : Como inicializar o Modo Gráfico.

```
driver=DETECT;
modo=getgraphmode();
initgraph(&driver,&modo,"");
```

Apêndice D: Exercícios Propostos

1. Programa de Desenho Simplificado

```
#include <mouse.h>
#include <graphics.h>
#include <stdio.h>
#include <conio.h>
int gdriver = VGA, gmode = VGAHI;
struct mouse_set teste;

void main()
{
   int op=0,cor1,cor2;
   if (!(inicializa_mouse()))/*verifica se existe mouse */
      { cprintf("\n\rMouse nao instalado\n");
        exit(0);
      }

   initgraph(&gdriver,&gmode,""); /* inicializa modo
                                     grafico */
  do{
   /*####################################
      ## define parametro para desenho ##
      ####################################*/
   outtextxy(1,5,"Entre com o numero da cor desejada para o
                  pincel =>");
   gotoxy(53,1);scanf("%d",&cor1);

   outtextxy(1,35,"Entre com o numero da cor desejada para o
                   fundo =>");
   gotoxy(53,3),scanf("%d",&cor2);

   setbkcolor(cor2);
   cleardevice();

   mostra_cursor();
      while((op=kbhit())==0)
        { teste=mouse_clique();
            /* desenha na tela se botao esquerdo pressionado */
```

```
            while(teste.esquerdo)
             {
              teste=mouse_clique();
              esconde_cursor();
              putpixel(teste.x,teste.y,cor1);
              mostra_cursor();
             }
         }
   }
   while((op=getch())!=27);
}
```

2. Programa para enviar sinais para Porta Paralela

```
#include <dos.h>
#include <stdio.h>

#define porta 0x378
#define pino2 0x02
#define pino3 0x03
#define pino4 0x04
#define pino5 0x05
#define apagado 0x00

main()
       {
        int op=50;

           outportb(porta,apagado);
           do
           {
              clrscr();
              switch (op)

                 {
                  case 50:
                              outportb(porta,pino2);
                              break;
                  case 51:
                              outportb(porta,pino3);
                              break;
                  case 52:
                              outportb(porta,pino4);
                              break;
                  case 53:
                              outportb(porta,pino5);
                              break;
                  default:
                             clrscr();
                             printf("Tecla inv lida  !tecle enter!");
                             getch();
                             clrscr();
                             break;

                 }
              printf("2 - pino 2\n");
              printf("3 - pino 3\n");
              printf("4 - pino 4\n");
              printf("5 - pino 5\n");
              printf("ESC para sair\n");
```

```
        }
    while((op=getch())!=27);
}
```

3. Programa para Configurar a COM1.

```
#include <stdio.h>
#include <dos.h>
void set_buf(void);
void set_stop(void);
void set_paridade(void);
void set_par_imp(void);

main()
  {int op=0;

   do
     {clrscr();
     switch (op)
       {
           case 49:
                  set_buf();
                  break;
          case 50:
                  set_stop();
                  break;
           case 51:
                  set_paridade();
                  break;
           case 52:
                  set_par_imp();
                  break;}
    printf("1 - Setar tamanho do buffer\n");
     printf("2 - Setar bit de stop bits\n");
     printf("3 - Setar bit de paridade\n");
     printf("4 - Setar paridade par/impar\n");
     printf("ESC para sair\n");}
     while((op=getch())!=27);
  }

void set_buf(void)
 {  unsigned char porta;
    int valor,tamanho;
     clrscr();
     printf("Entre com o tamanho do buffer ->");
     scanf("%d",&tamanho);
    valor=(inportb(0x3fb)|0x80); /* setar DLAB do LCR */
    outportb(0x3fb,valor);
    switch (tamanho)
      { case 5:
           outportb(0x3fb,0x0);
           break;
        case 6:
           outportb(0x3fb,0x1);
           break;
        case 7:
           outportb(0x3fb,0x2);
           break;
        case 8:
           outportb(0x3fb,0x3);
           break;}
     clrscr(); }

void set_stop(void)
```

```
 {  unsigned char porta;
    int valor;
    clrscr();
     valor=(inportb(0x3fb)|0x80); /* setar DLAB do LCR */
     outportb(0x3fb,valor);
    valor=valor|4;               /*setar bit2 – stopbit */
     outportb(0x3fb,valor); }

void set_paridade(void)
 {   unsigned char porta;
      int valor,op;
     clrscr();
     printf("Setar ou Resetar? (1/0)");
      scanf("%d",&op);
     valor=(inportb(0x3fb)|0x80); /* setar DLAB do LCR */
     outportb(0x3fb,valor);
     switch (op)
      { case 0: valor=valor&0xf7; /* setar bit3 em 0 */
                  outportb(0x3fb,valor);
                  break;
            case 1:valor=valor|8;
                     outportb(0x3fb,valor); /* setar bit3 em 1 */
                     break;}
     clrscr();}
void set_par_imp(void)
 {   unsigned char porta;
     int valor,op;
     clrscr();
     printf("Par ou Impar? (1/0)");
      scanf("%d",&op);
     valor=(inportb(0x3fb)|0x80); /* setar DLAB do LCR */
     outportb(0x3fb,valor);
        switch (op)
        {   case 0:valor=valor&0xef; /* setar bit4 em 0 */
                     outportb(0x3fb,valor);
                     break;
            case 1:valor=valor|16;
                     outportb(0x3fb,valor); /* setar bit4 em 1*/
                     break;}
    clrscr();}
```

4. Programa para enviar sinais para porta Paralela de acordo com os movimentos do mouse.

```
#include <c:\tc\include\mouse.h>
#include <graphics.h>
#include <conio.h>
#include <process.h>
#include <stdio.h>

#define porta 956
#define apagado 0
#define tempo  300

int gdriver=VGA, gmode=VGAHI;
char nome[250];
struct mouse_set teste;
int acumulador;
void (*rotina_antiga)();
```

```
int mascara_antiga;

int main()
{
   int x=0,y=0,op=0, flag=1;
   if (!(inicializa_mouse()))
     { cprintf("\n\rMouse nao instalado. Programa abortado.\n\r");
       exit(0); }
   initgraph(&gdriver,&gmode,"");
   cursor_area(200,400,200,300);
   setbkcolor(9);
   cleardevice();
   mostra_cursor();
   outportb(porta,apagado);
   mouse_interrupt(2);
   while(1)
     {teste=mouse_clique();
      if(teste.esquerdo==1)
       {if(flag)
           {x = teste.x;
            y = teste.y;
            esconde_cursor();
            putpixel(x,y,4);
            mostra_cursor();
            flag=0;}
        else
           if(!flag)
            {esconde_cursor();
             putpixel(teste.x,teste.y,4);
             mostra_cursor();
             if(teste.x>x) /* pino4*/
                 {gotoxy(1,1);
                  printf("direita  teste.x:%d e x:%d",teste.x,x);
                  outportb(porta,8);}
             if(teste.x<x)/* pino3 */
                 {gotoxy(1,1);
                  printf("esquerda teste.x:%d e x:%d",teste.x,x);
                  outportb(porta,4);}
             if(teste.y<y) /* pino2 */
                 {gotoxy(1,2);
                  printf("p cima  teste.y:%d e y:%d",teste.y,y);
                  outportb(porta,2);}
             if(teste.y>y) /* motor para baixo */
                 {gotoxy(1,2);
                  printf("p baixo teste.y:%d e y:%d",teste.y,y);
                  outportb(porta,1);}
             x = teste.x;
             y = teste.y;
             outportb(porta,apagado);
            }
          }
        }
   esconde_cursor();
   restorecrtmode();
   outportb(porta,apagado);
   return 0;
}
```

Bibliografia

[1] Schildt, Herbert. **"C Completo e Total"**, Ed. Makron Books,
São Paulo, SP, 1996.
[2] Zelenovsky Ricardo, **"PC e Periféricos – Um Guia Completo de Programação"**, Ed. Ciência Moderna, Rio de Janeiro, RJ, 1996.
[3] Zelenovsky Ricardo, **"PC: Um Guia Completo de Hardware e Interfaceamento"**, Ed. MZ,Rio de Janeiro, RJ, 1999.
[4] Manual "TURBO C Reference Guide", Borland, 1988.