MAKRON
Books

Capítulo 2

COMUNICAÇÃO SERIAL

INTRODUÇÃO

Um computador pode ser conectado a outro computador ou a uma grande variedade de acessórios por meio da interface serial.

O objetivo deste capítulo é explorar os fundamentos da operação da placa serial e a conexão direta com outro computador, incluindo os cabos e soquetes necessários e a definição de quais fios são usados para quais propósitos.

Este capítulo é considerado fundamental para a compreensão da comunicação com outros periféricos ligados ao computador pela placa serial.

Desenvolveremos programas que fazem a comunicação micro a micro e que podem ser modificados e adaptados para se comunicar com qualquer outro acessório serial, visto que a conexão é análoga.

COMUNICAÇÃO PARALELA E SERIAL

Quando aprendemos uma linguagem de programação, geralmente começamos aprendendo a nos comunicar com os periféricos básicos de entrada e saída como vídeo, teclado, drives e impressora. Esses periféricos são conectados diretamente ao microprocessador.

Para esses periféricos, os dados são enviados, recebidos ou deslocados de parte a parte em paralelo. Assim, 8 bits de informações (um byte) são movidos de um lugar para outro todos de uma única vez, em 8 linhas diferentes.

O número de bits enviados de uma vez varia de máquina para máquina, mas são sempre múltiplos de 8. Desta forma, um computador trabalha com pelo menos um byte e muitas vezes com 2 ou mais bytes de uma vez.

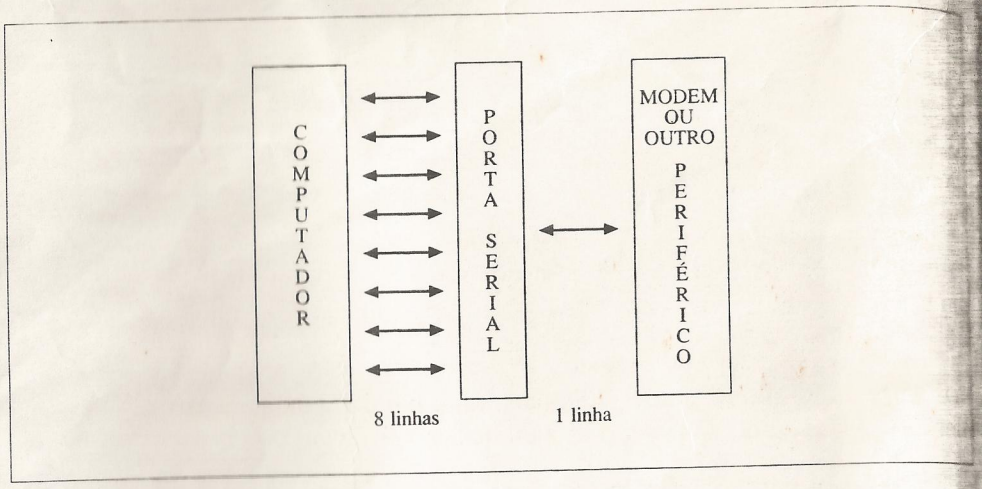
Vários periféricos, como modems, plotters, mouses e outros foram fabricados de forma não-compatível com a comunicação paralela do computador. Esses periféricos usam comunicação serial.

Em comunicações seriais, os dados são enviados e recebidos um único bit de cada vez. Isso requer, teoricamente, um único fio de linha físico.

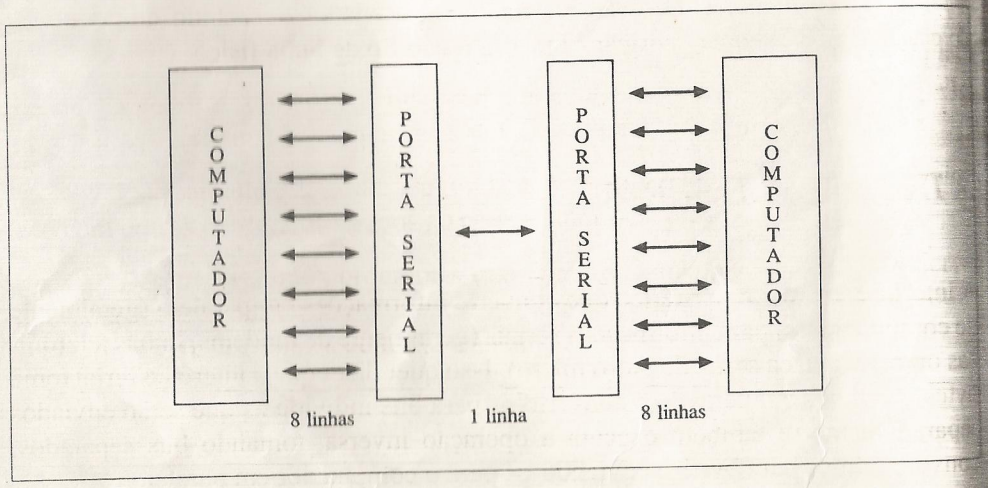
A INTERFACE SERIAL

A interface serial é o periférico que converte informações em paralelo (geralmente de computadores) para informações seriais (geralmente de modems, linhas telefônicas ou para a placa serial de outro micro). Isso quer dizer que a interface serial toma bytes recebidos em paralelo e converte-os para bits individuais que serão enviados separadamente, e também executa a operação inversa, tomando bits separados, convertendo-os para bytes e enviando-os para o computador em paralelo.

A Comunicação com um Periférico Serial



Comunicação Micro a Micro



A comunicação paralela é mais rápida, visto que vários bits podem ser enviados de uma vez, enquanto a comunicação serial tem a vantagem de requerer um cabo mais simples.

CONECTORES

O mercado oferece vários tipos de conectores próprios para a conexão de cabos a periféricos seriais.

As portas seriais geralmente utilizam conectores de 25 pinos ou conectores de 9 pinos, chamados "conectores tipo D" e muitas vezes referenciados como DB-25 e DB-9. O nome "D" vem de sua forma bastante semelhante à letra D.

Os conectores com pinos são conhecidos como conectores machos, e os com soquetes são conhecidos como conectores fêmeas. Cada pino ou soquete tem um número próprio que geralmente está impresso no conector.

O PADRÃO RS-232

Para tornar equipamentos diferentes compatíveis entre si, vários padrões foram desenvolvidos. O mais usado é o RS-232, divulgado em 1969 pela Electronic Industries Association.

O padrão RS-232 foi desenhado para especificar as características elétricas dos circuitos e dar nomes e números às linhas necessárias para a conexão entre equipamentos.

As placas seriais de computadores são fabricadas com base no padrão RS-232. Por esse motivo, são também chamadas placas RS-232.

Os microcomputadores utilizam somente 9 pinos conectados em lugar dos 25 pinos necessários para o circuito completo RS-232.

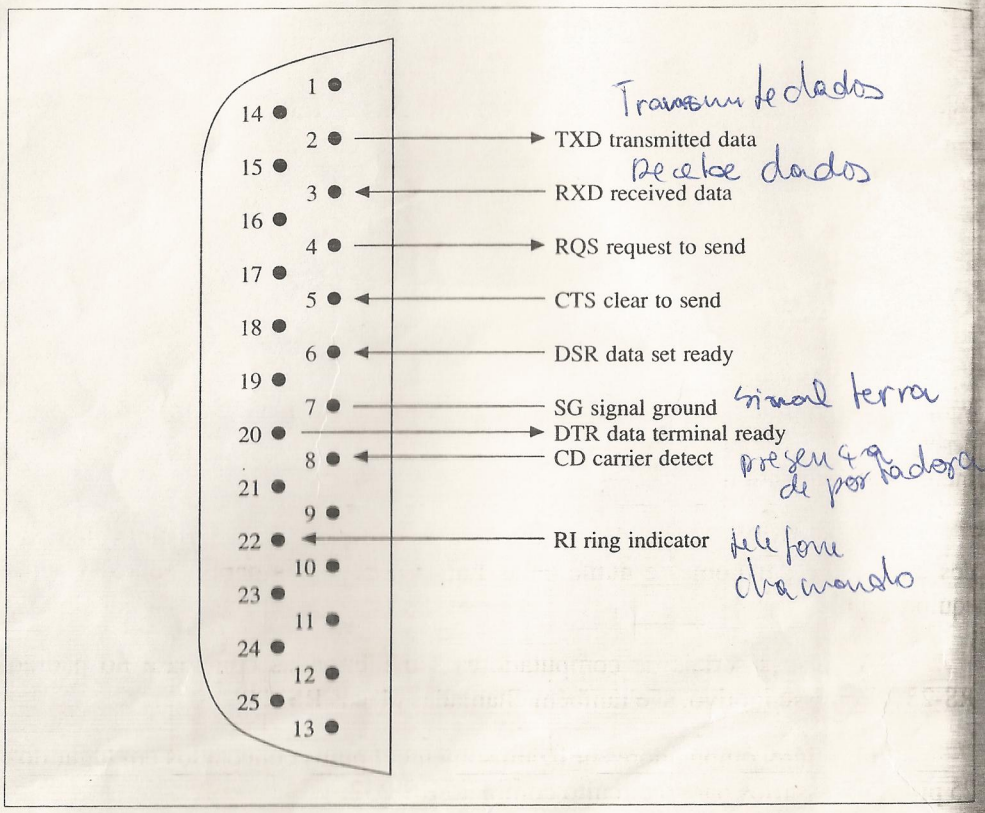
Para que periféricos possam conversar através da mesma linha, eles são divididos em dois tipos: o tipo terminal, que usa o pino 2 para mandar dados, e o tipo modem, que usa o pino 2 para receber dados.

De acordo com o padrão RS-232, periféricos terminais devem ser acoplados de um conector macho, e periféricos modem, de um conector fêmea.

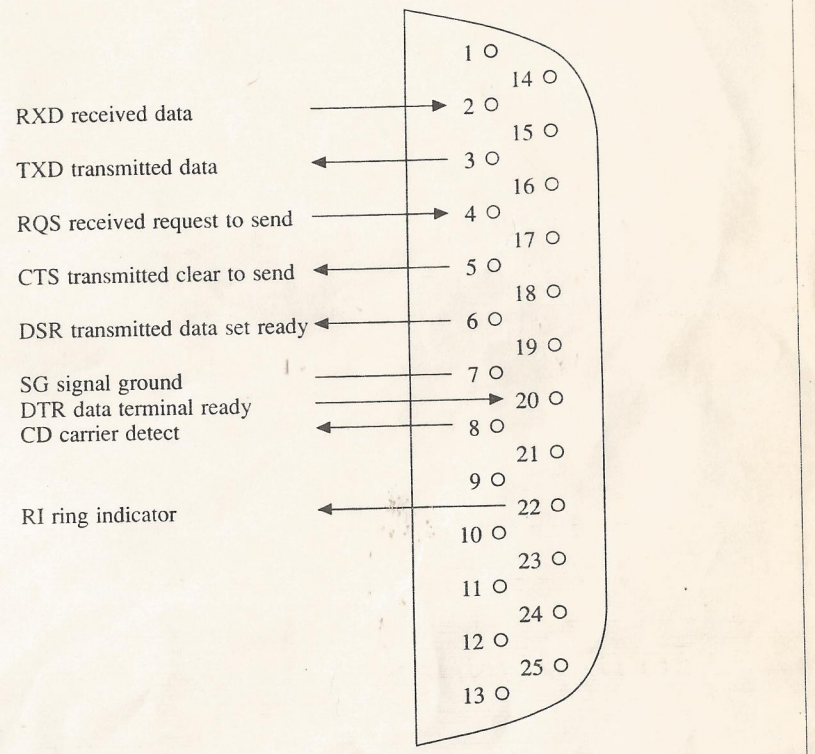
AS 9 LINHAS RS-232 PARA CONECTORES DE 25 PINOS

A definição das 9 linhas RS-232 utilizadas pelo microcomputador é a seguinte:

Periférico Terminal (IBM-PC)

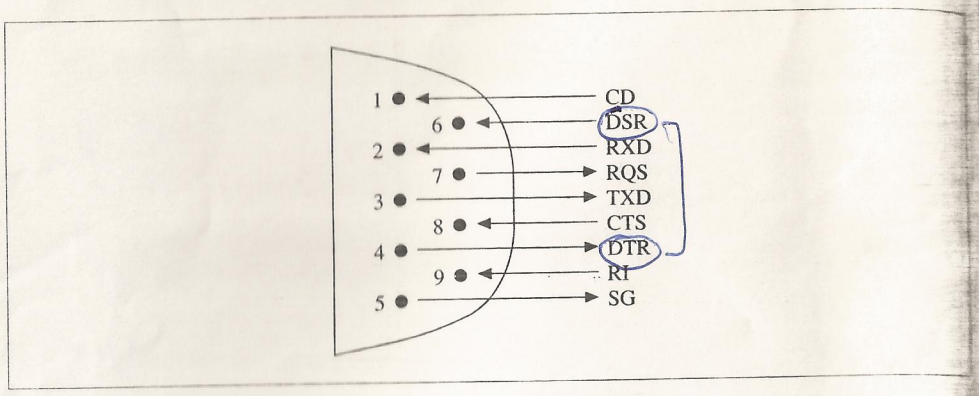


Periférico Modem (Modem, Impressora etc.)



AS 9 LINHAS RS-232 PARA CONECTORES DE 9 PINOS

Periférico Terminal (IBM-PC)



DIVISÃO DAS LINHAS

LINHAS 2 e 3

TXD – "transmitted data" – Transmite dados

RXD – "received data" – Recebe dados

LINHAS 4 e 5

RQS – "request to send" – Handshaking

CTS – "clear to send" – Handshaking

LINHAS 6 e 20

DSR – "data set ready" – Handshaking

DTR – "data terminal ready" – Handshaking

LINHA 7

SG – "signal ground" – Sinal terra

LINHA 8

CD – "carrier detect" – Presença de portadora

modem ocupado

LINHA 22

RI – "ring indicator" – Telefone chamando

Indica que modem está chamando

UMA COMUNICAÇÃO SIMPLES

Quando um periférico somente transmite e o outro somente recebe, a comunicação necessita apenas de dois sinais em cada periférico: TXD e SG (linhas 2 e 7) no periférico que transmite (IBM-PC) e RXD e SG (linhas 3 e 7) no periférico que recebe (modem).

HANDSHAKING

Handshaking é o método pelo qual periféricos que se comunicam podem controlar o fluxo de transmissão de dados entre eles.

Podemos imaginar, por exemplo, que um periférico recebe os dados mais depressa do que tem capacidade de processá-los. Nesse caso, ele deve ser capaz de avisar o transmissor para parar a transmissão até que tenha processado os caracteres já recebidos.

Isso ocorre, por exemplo, quando mandamos dados de um computador para outro que não processa os dados tão rapidamente quanto eles vêm chegando. Outro exemplo ocorre quando enviamos dados a uma impressora serial que não imprime na velocidade de envio.

Nesses casos, o periférico pode transmitir um ou dois sinais de handshaking indicando que a sua capacidade de recepção se esgotou e solicitando uma parada temporária do envio de dados. A linha 20 e/ou 4 é usada pelo periférico terminal (IBM-PC) e a linha 6 e/ou 5 é usada pelo periférico modem (modem, impressora) para esta finalidade.

COMUNICAÇÃO NAS DUAS DIREÇÕES

Se cada um dos dois periféricos interligados tanto transmite como recebe, o número mínimo de linhas necessárias em cada periférico é 3: TXD, RXD e SG. Quando se deseja controlar o fluxo de transmissão de dados (handshaking), devem ser adicionadas as linhas DSR e DTR em cada periférico. Quando uma segunda linha de handshaking for desejada, devem ser adicionadas as linhas RQS e CTS em cada periférico. Assim, o número total de linhas sobe para 7.

Duas outras linhas adicionais são geralmente usadas. A linha 8 (CD), que indica a presença de portadora, isto é, indica que o modem da outra ponta está pronto. E a linha 22 (RI), que indica que o telefone está chamando, ou seja, que o modem está sendo chamado pelo periférico remoto.

COMUNICAÇÃO MICRO A MICRO

A conexão de dois periféricos terminais (micro e micro) ou de dois periféricos modems é chamada NULL MODEM. Esse tipo de ligação requer uma atenção especial pelo fato de os dois periféricos transmitirem pela mesma linha e receberem pela mesma linha. Ou seja, o problema surge porque o primeiro periférico transmite pela linha 2 e recebe pela linha 3 e o segundo periférico também transmite pela linha 2 e recebe pela linha 3. Assim sendo, é necessário que o cabo de ligação tenha essas linhas cruzadas para permitir que a linha ligada ao pino 2 do primeiro periférico seja ligada ao pino 3 do segundo periférico, e a linha ligada ao pino 2 do segundo periférico esteja ligada ao pino 3 do primeiro.

As linhas 2 e 3 cruzadas permitem que um periférico transmita pela linha 2 e os dados transmitidos sejam recebidos pela linha 3 do outro periférico.

Caso se ligue o pino 2 com o 2, o 3 com o 3 etc., estará sendo ligada saída com saída e entrada com entrada e a conexão não será bem-sucedida.

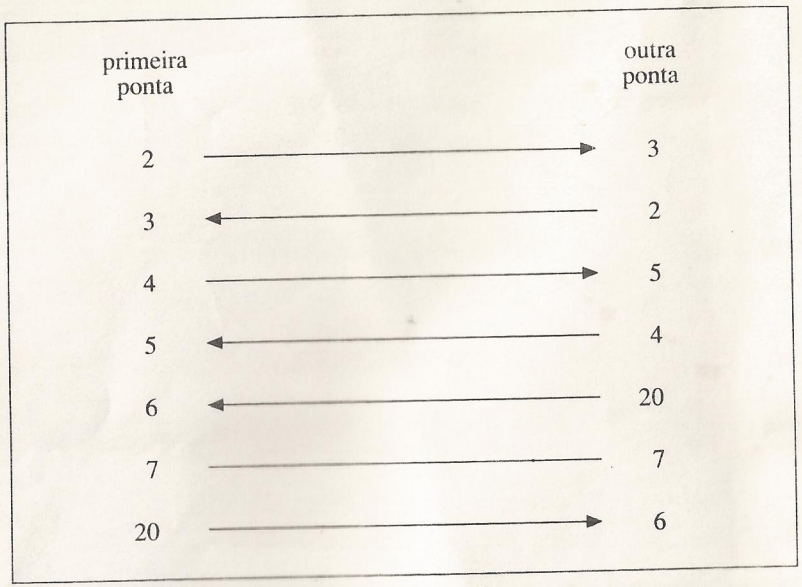
Para conectar os dois micros você pode usar um cabo serial com as linhas cruzadas ou colocar um conector que conecte os dois cabos e faça o cruzamento necessário internamente. Em qualquer dos casos, o cabo ou o conector é chamado *null modem*, pois faz o papel de dois modems (terminal:modem:modem:terminal).

O cabo de ligação deve seguir as seguintes especificações:

1. O pino 2 do primeiro micro deve ser ligado ao pino 3 do segundo.
2. O pino 4 do primeiro micro deve ser ligado ao pino 5 do segundo.
3. O pino 20 do primeiro micro deve ser ligado ao pino 6 do segundo.

AS LINHAS DE UM CABO NULL MODEM

Você pode fazer um cabo para interligar dois micros pela placa serial, soldando os fios nos conectores, seguindo o seguinte esquema:



UART

O coração da porta serial é o chip chamado UART (Universal Asynchronous Receiver Transmitter). O trabalho do UART é tanto transformar cada byte de informação paralela numa corrente de bits seriais como também executar a operação inversa. Os computadores IBM-PC usam o UART National 8250.

O UART deve ser configurado conforme as necessidades da nossa comunicação, antes que as informações possam ser transmitidas ou recebidas. Os elementos de configuração que devem ser considerados são os seguintes.

TAXA DE TRANSMISSÃO

A velocidade de transmissão deve ser escolhida considerando-se a velocidade de recepção do equipamento com o qual vamos comunicar-nos.

Podemos escolher uma entre várias taxas de transmissão suportadas pelo UART para o envio dos dados. A escolha comum com modems é de 300 bits por segundo (bps), 1.200 bps ou 2.400 bps. O UART pode enviar e receber dados em várias outras velocidades, chegando até a 115.200 bps.

NÚMERO DE BITS DE DADOS

O número de bits de dados pode ser 7 se os dados a serem transmitidos estiverem no formato texto e 8 no caso de os dados estarem no formato binário.

PARIDADE

Se forem transmitidos 7 bits de dados, o oitavo bit pode ser usado para indicar paridade.

A paridade pode ser par, ímpar ou nenhuma. Se for escolhida paridade par, os bits de dados transmitidos mais o bit de paridade formarão um número par. Se for escolhida paridade ímpar, os bits de dados transmitidos mais o bit de paridade formarão um número ímpar.

O bit de paridade age como um simples esquema de detecção de erro. Se um dos 7 bits de dados é perdido na transmissão, o bit de paridade não estará correto e pode ser detectado pelo receptor.

Muitas vezes o bit de paridade não é usado. Nesse caso, deve-se avisar o UART desse fato.

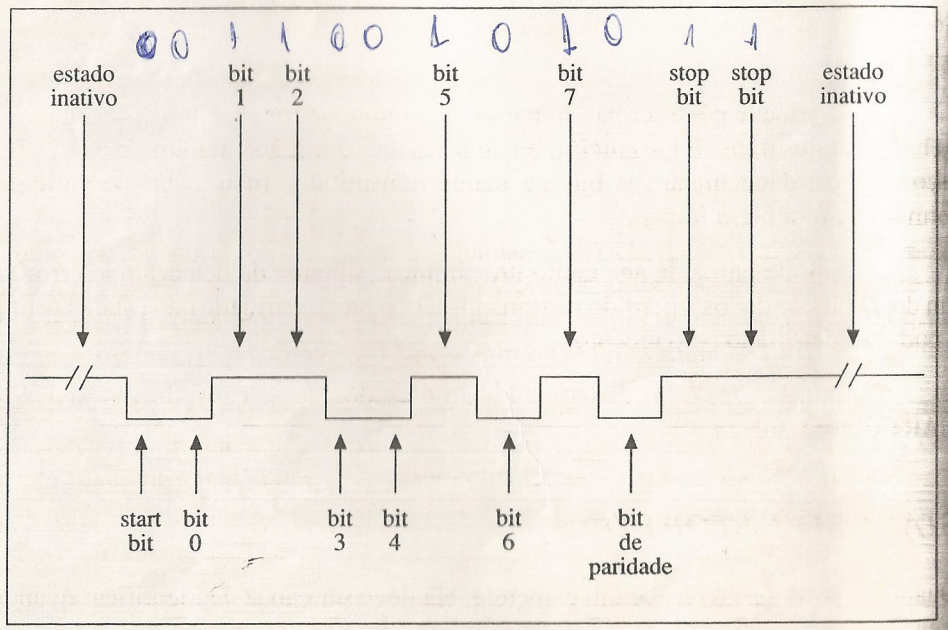
START BIT E STOP BIT

Quando a porta serial recebe um caractere, ela deve ser capaz de identificar quando o envio do caractere começa e quando termina. Isso é feito por meio do bit de início (start bit) e por um ou dois bits de finalização (stop bits).

Quando não há caracteres sendo mandados, o estado dos bits da porta será sempre 1. Para sinalizar a chegada de um caractere, enviamos um bit de início com o valor zero. Em seguida, são enviados os 7 ou 8 bits de dados, o bit de paridade e finalmente um ou dois bits de finalização com o valor um.

SEQÜÊNCIA DE SINAIS PARA A TRANSMISSÃO DE UM BYTE

Quando enviamos um bit ligado (1) para a porta serial, estamos colocando o estado da linha alto, e quando enviamos um bit desligado (0), estamos abaixando a linha. O esquema do envio dos bits para a transmissão de um byte é o seguinte:



CONFIGURANDO A PORTA SERIAL – UART

O UART contém registradores internos que devem ser programados conforme os elementos citados anteriormente para especificar as opções desejadas. Os endereços desses registradores estão relacionados com as portas COM1 e COM2, conforme a tabela a seguir:

Endereços das Portas COM1 e COM2

COM 1	*COM 2*	*Descrição*
0x3F8	0x2F8	Buffer de transmissão
0x3F8	0x2F8	Buffer de recepção
0x3F8	0x2F8	Byte baixo da taxa de transmissão
0x3F9	0x2F9	Byte alto da taxa de transmissão
0x3F9	0x2F9	Permite habilitar interrupções
0x3FA	0x2FA	Identifica última interrupção ocorrida
0x3FB	0x2FB	Controla características da comunicação
0x3FC	0x2FC	Controla sinais do modem
0x3FD	0x2FD	Status
0x3FE	0x2FE	Status (Modem)

REGISTRADOR DE BUFFER DE TRANSMISSÃO

TRANSMIT HOLDING REGISTER

O registrador de buffer de transmissão armazena o próximo caractere a ser transmitido. O caractere é colocado nesse registrador pelo nosso programa e, quando é transmitido, o registrador de status indica o fato.

REGISTRADOR DE BUFFER DE RECEPÇÃO

RECEIVE BUFFER REGISTER

O registrador de buffer de recepção armazena o último caractere recebido. Quando o caractere for lido pelo nosso programa, o registrador de status indicará que o buffer de recepção está vazio até que um outro caractere seja recebido. Se um segundo caractere for recebido antes que o primeiro seja lido, um erro de *overrun* ocorrerá.

REGISTRADORES DE TAXA DE TRANSMISSÃO DLL DLM

DIVISOR LATCH LSB e DIVISOR LATCH MSB

O número a ser enviado para o UART para informar a taxa de transmissão desejada não é o número de bps (bits por segundo) que queremos. Esse valor é chamado divisor e tem a seguinte relação com o número de bps:

Divisor = 1.843.200 / (16 * bps)

Visto que esse valor é um número de 16 bits e cada registrador do UART é de 8 bits, precisamos enviar nosso valor em dois registradores: o byte menos significativo para o registrador de endereço 0x03F8 e o byte mais significativo para o registrador de endereço 0x3F9 (0x2F8 e 0x2F9 para a porta COM2).

REGISTRADOR QUE PERMITE HABILITAR INTERRUPÇÕES IER

INTERRUPT ENABLE REGISTER

Podemos instruir o UART para gerar um sinal de interrupção sempre que certos eventos ocorrerem. Esse registrador é usado para informar ao UART quais eventos causarão a interrupção.

O método que usaremos aqui não necessita permitir interrupções, pois o programa examinará continuamente o registrador de status para verificar o que está ocorrendo.

Os bits desse registrador correspondem às seguintes solicitações de interrupções:

Bit	*Interrupção Selecionada*
0	Dado presente
1	Registrador de transmissão vazio
2	Linha de status
3	Status do modem
4-7	Sempre zero

REGISTRADOR QUE IDENTIFICA ÚLTIMA INTERRUPÇÃO OCORRIDA

O registrador de identificação de interrupções informa sobre o status de interrupções pendentes. Se não há interrupções pendentes, o bit 0 estará ligado (1). Se esse bit estiver desligado (0), os bits 1 e 2 indicarão qual interrupção está pendente, conforme a tabela a seguir.

Bit 1	*Bit 2*	*Interrupção Selecionada*
0	1	Dado presente
1	0	Registrador de transmissão vazio
1	1	Linha de status
0	0	Status do modem

Os bits de 3 a 7 estarão sempre desligados.

REGISTRADOR DE CONTROLE DAS CARACTERÍSTICAS DE COMUNICAÇÃO LCR

LINE CONTROL REGISTER

O registrador de controle é usado para selecionar os parâmetros de comunicação. O significado de cada bit do registrador é mostrado na tabela a seguir.

Bit	*Objetivo*
0 e 1	Número de bits de dados
	00 – 5 bits
	01 – 6 bits
	10 – 7 bits
	11 – 8 bits
2	Stop bits
	0 – Um stop bit
	1 – Dois stop bits
3	Paridade
	0 – Sem paridade

Bit	Objetivo (continuação)
	1 – Permite paridade
4	Seleciona Paridade (ignorando se bit 3=0)
	0 – Paridade ímpar
	1 – Paridade par
5	Paridade um
	0 – 0 bit de paridade será 1 lógico
	1 – 0 bit de paridade será 0 lógico
6	Usando para gerar um comando Break
	1 – Força o envio de um 0 lógico até que esse bit seja desligado
7	Taxa de Transmissão
	1 – Acessa os registradores de velocidade nas operações de leitura e envio
	0 – Acessa os registradores de interrupção

REGISTRADOR DE CONTROLE DE SINAIS DO MODEM

MODEM CONTROL REGISTER MCR

O registrador de controle de sinais do modem é usado para selecionar os parâmetros de comunicação do modem. O significado de cada bit do registrador é mostrado na tabela a seguir.

Bit	Objetivo
0	Data Terminal Ready
1	Request to Send
2	Carrier Detect
3	Ring Indicator
4	Permite teste de diagnóstico
5-7	Sempre zero

REGISTRADOR DE STATUS

LSR

LINE STATUS REGISTER

O registrador de status é usado para obter informações que dizem respeito aos dados recebidos ou transmitidos. O significado de cada bit do registrador é mostrado na tabela a seguir.

Bit	Objetivo se o bit estiver ligado (1)
0	Dados prontos (*Data Ready*) Um dado foi recebido. Esse bit permanecerá ligado até que o caractere seja lido.
1	Erro de Overrun Outro dado foi recebido antes que o anterior fosse removido. Isso indica que os dados têm sido recebidos mais rápido que o tempo para processá-los.
2	Erro de Paridade Um erro de paridade ocorreu no dado recebido.
3	Erro de Framing O caractere recebido não tem o número de stop bits selecionados.
4	Interrupção por Break Um break foi recebido. Isto é, os bits recebidos são zero para mais bits que o tamanho selecionado.
5	Registrador de transmissão vazio O UART está pronto para receber um novo caractere para ser transmitido.
6	Registrador de conversão vazio O registrador de conversão de paralelo para serial está pronto para receber outro dado.
7	Permanentemente zero.

REGISTRADOR DE STATUS DO MODEM

MODEM STATUS REGISTER

O registrador de status do modem é usado para obter informações que dizem respeito às linhas de handshaking. O significado de cada bit do registrador é mostrado na tabela a seguir.

Bit	Objetivo
0	Mudança na linha "clear to send"
1	Mudança na linha "data set ready"
2	Mudança na linha "ring indicator"
3	Mudança na linha "signal detect"
4	Linha "clear to send" OK
5	Linha "data set ready" OK
6	Linha "ring indicator" OK
7	Linha "signal detect" OK

UM PROGRAMA SIMPLES DE COMUNICAÇÃO MICRO A MICRO – S115200.C

Nosso primeiro exemplo mostra um programa de comunicação que pode ser usado como programa básico ou esqueleto para comunicação micro a micro. Para testá-lo, você deve executá-lo em dois micros ligados por um cabo *null modem*.

Esse programa aguarda que sejam digitados caracteres no teclado, envia-os para a placa serial, toma-os pelo programa do outro computador, que, finalmente, imprime-os em seu vídeo.

Escolhemos fazer a comunicação com os seguintes parâmetros: 115.200 bps, sem paridade, 1 stop bit e 8 bits de dados.

```
/* S115200.C */
/*****************************************************
 * Comunicação direta com a porta serial através da *
 * programação do UART                               *
 *****************************************************/
#define COM0 0x03f8     /* para COM1. COM2 usar 0x02f8     */
#define TAXA 115200L    /* bps                             */
#define PARIDADE 0      /* 0=Nenhuma, 1=ímpar, 2=par       */
#define STOP 1          /* 1 stop bit                      */
#define BITS_DADOS 8    /* 8 bits de dados                 */
#define TRUE 1
void inicializa (unsigned int,long,int,int,int);
main( )
{
   int status;
   unsigned char ch;
   /* inicializa UART */
   inicializa(COM0,TAXA,PARIDADE,STOP,BITS_DADOS);
   while(TRUE) {
     if(kbhit( )) {          /* se tecla pressionada        */
```



```
       ch=getch( );          /* lê char                     */
       outportb(COM0,ch);    /* envia char                  */
     }
     status=inport(COM0+5);
     if((status & 1)){       /* dados disponíveis ?         */
       ch=inportb(COM0);
       if(ch == '\r') {      /* se carriage return          */
         putch('\r');        /* imprime CR + LF             */
         putch('\n');
       } else /* se outro caractere */
         putch(ch); /* imprime caractere */
     }
   }

 * inicializa( ) */
/*****************************************************
 * Inicializa porta serial com os parâmetros         *
 * enviados como argumentos                          *
 *****************************************************/
void inicializa (porta,bps,p, stop, bits_dados)
unsigned int porta; /* 0x03f8 se com1 e 0x02f8 se com2 */
long bps; /* bits por segundo - taxa de transmissão */
int p; /* paridade - 0=Nenhuma, 1=ímpar, 2=par */
int stop; /* 1 ou 2 stop bits */
int bits_dados; /* 7 ou 8 bits de dados */
{
   union{
     unsigned int nr;
     unsigned char hilo[2];
   }    divisor;
   unsigned char byte_de_param;
```

```
    /* Calcula valor a ser enviado para registradores      */
    /* de taxa de transmissão -                            */
    divisor.nr= (unsigned int)(1843200L/(bps*16L));
    /* Liga bit do registrador de controle das caract.     */
    /* que permite acesso aos registradores de taxa de     */
    /* transmissão                                         */
    outportb(porta+3,0x80); // LCR
    /* Byte menos significativo para porta                 */
    outportb(porta,divisor.hilo[0]);
    /* Byte mais significativo para porta                  */
    outportb(porta+1,divisor.hilo[1]);
    /* Formata byte de parâmetros                          */
    byte_de_param = bits_dados-5;
    byte_de_param |= (stop - 1)<<2;
    if (p){
      byte_de_param |= 0x08; /* liga bit de paridade       */
      byte_de_param |= p&0x10; /* paridade par ou ímpar     */
}
    /* Envia byte de parâmetros para registrador de        */
    /* controle de linha                                   */
    outportb(porta+3,byte_de_param); // LCR
}
```

TRANSFERINDO ARQUIVOS ENTRE COMPUTADORES

Uma alteração útil do programa S115200.C é a de incluir uma função que recebe um arquivo pela placa serial e o grava em disco e outra função que envia um arquivo do disco para a placa serial.

Vamos mostrar a seguir um programa que inclui essa alteração e trabalha com todos os tipos de arquivos, tendo eles qualquer formato e qualquer tamanho.

Para que possamos transferir e receber qualquer arquivo, a manipulação dos mesmos deve ser feita em modo binário. Como nesse modo todos os bytes são números que pertencem ao arquivo, há um pequeno problema a ser levantado: a marca de fim de arquivo não poderá ser identificada pela função receptora.

Então, solucionamos o problema contando o número de bytes do arquivo a ser transferido e enviando esse número para a função receptora antes que ela leia os bytes do arquivo propriamente dito. Desta maneira, a função lerá um número fixo de bytes.

A função envia_arq( ) primeiramente verifica a existência em disco do arquivo indicado na linha de comando na chamada ao programa. Em seguida, conta o número de bytes contidos nele, armazena esse valor numa variável do tipo long e envia os 4 bytes desta variável, um por vez, pela porta serial.

Após comunicar o tamanho do arquivo que será transmitido e receber um sinal indicando que esse valor foi recebido corretamente, a função envia_arq( ) começa a transmitir byte a byte do arquivo até encontrar a marca de fim de arquivo (EOF). A cada byte enviado, a função aguarda um sinal da função receptora indicando que o byte foi recebido e lido corretamente.

Nós fixamos o reconhecimento do sinal pelo recebimento do caractere "*". Assim, a cada byte recebido pela função recebe_arq( ), é enviado o caractere "*" como sinal de leitura bem-sucedida. A função recebe_arq( ) inicia-se recebendo o tamanho do arquivo que será transmitido. Aguarda e lê byte a byte desse arquivo por meio da porta serial e grava-os no arquivo em disco indicado pela linha de comando na chamada ao programa.

Eis a listagem:

O PROGRAMA SRE.C

```
/* SRE.C */
/*************************************************************
 * Este programa faz transferência de arquivos através da    *
 * porta serial. Utiliza comunicação direta com a porta      *
 * serial por meio da programação do UART.                   *
```

```
*                        MODO DE USO                              *
*                      = = = = = = = =                            *
* Deve ser instalado em dois computadores da seguinte             *
* forma:                                                          *
*        A linha de comando para o computador                     *
*        que receberá o arquivo é:                                *
*                A>SRE R <nomearq.ext>                            *
*        A linha de comando para o computador                     *
*        que enviará o arquivo é:                                 *
*                A>SRE E <nomearq.ext>                            *
*****************************************************************/

#define COM0 0x03f8     /* para COM1. COM2 usar 0x02f8     */
#define VELOC 115200L   /* bps                             */
#define PARIDADE 0      /* 0=Nenhuma, 1=ímpar, 2=par       */
#define STOP 1          /* 1 stop bit                      */
#define BITS_DADOS 8    /* 8 bits de dados                 */
#define TRUE 1
void inicializa (unsigned int,long,int,int,int);
void recebe_arq (char *);
void envia_arq (char *);
main(int argc,char **argv)
{
   if(argc < 3)
   {
      printf("\nModo de uso: SRE R <nomearq.ext>");
      exit( );
   }
   if(argv[1][0]=='r' || argv[1][0] == 'R')
      recebe_arq(argv[2]);
   else if(argv[1][0]=='e' || argv[1][0] == 'E')
      envia_arq(argv[2]);
```



```
   else
   {  printf("\nParâmetro invalido.");
      printf("\nDigite R para receber arquivo");
      printf("\n E para enviar arquivo");
      exit( );
   }

#include <stdio.h> /* arquivo de alto nível */
/*****************************************************
 * recebe_arq( )                                     *
 * recebe um arquivo pela serial e grava-o em nomearq *
 *****************************************************/
void recebe_arq(char *nomearq)
{
   int status;
   FILE *f;
   unsigned char ch=0;
   union{
     unsigned long arq;
     unsigned char byte[4];
   } tamanho;
   unsigned long i=0;
   if((f=fopen(nomearq,"wb")) == NULL)
   {
      printf("\nNao posso abrir arquivo %s",nomearq);
      exit( );
   }
   /* inicializa UART */
   inicializa(COM0,VELOC,PARIDADE,STOP,BITS_DADOS);
   while(TRUE){
     status=inport(COM0+5);
```

```
        if((status & 1))        /* dados disponíveis ? */
        {
            tamanho.byte[i++]=inportb(COM0);
            outportb(COM0,'*');
            if(i==4) break;
            }
        }
        i=0L;
        while( i < tamanho.arq)
        {
            status=inport(COM0+5);
            if((status & 1)) /* dados disponíveis ? */
            {
                ch=inportb(COM0); /* recebe caractere */
                putc(ch,f); /* grava no arquivo */
                outportb(COM0,'*'); /* envia sinal de pronto */
                i++;
            }
        }
        fclose(f);
}
/*********************************************************
 * envia_arq( )                                          *
 * envia o arquivo nomearq para a porta serial           *
 *********************************************************/
void envia_arq(char *nomearq)
{
    FILE *f;
    unsigned int ch;
    int status,i=0;
    union  {
```



```
    unsigned long arq;
    unsigned char byte[4];
    } tamanho;
if((f=fopen(nomearq,"rb")) == NULL)
{
    printf("\nNao posso abrir arquivo %s",nomearq);
    exit( );
}
/* calcula tamanho do arquivo a ser enviado              */
fseek(f,0L,2);
tamanho.arq = ftell(f);
fseek(f,0L,0); /* devolve ponteiro ao início do arquivo */
/* inicializa UART      */
inicializa(COM0,VELOC,PARIDADE,STOP,BITS_DADOS);
for(i=0;i<4;i++) /* envia tamanho do arquivo             */
{
    outportb(COM0,tamanho.byte[i]);
    status=inport(COM0+5);
    while(!(status&1)) /* aguarda sinal de pronto       */
      status=inport(COM0+5);
    /* dados disponíveis ?      */
    ch=inportb(COM0);
    if(ch!='*') /* se sinal não identificado            */
  {
    printf("\nErro na comunicacao");
    exit( );
  }
}
while((ch=getc(f))!=(unsigned int)EOF)
{
  outportb(COM0,ch); /* envia caractere                 */
```

```
    status=inport(COM0+5);
    while(!(status&1)) /* aguarda sinal de pronto          */
      status=inport(COM0+5);
    /* dados disponíveis */
    ch=inportb(COM0);

    if(ch!='*') /* se sinal não identificado              */
    {
        printf("\nErro na comunicacao");
        exit( );
    }
  }
  fclose(f);
  exit( );
}
/* inicializa( ) */
 ************************************************************
 * Inicializa porta serial com os parâmetros enviados como  *
 * argumentos                                               *
 ************************************************************/
void inicializa (porta,bps,p, stop, bits_dados)
unsigned int porta; /* 0x03f8 se com1 e 0x02f8 se com2     */
long bps;         /* 110,150,300,600,1200,2400,4800 ou 9600 */
int p;            /* paridade - 0=Nenhuma, 1=ímpar, 2=par   */
int stop;         /* 1 ou 2 stop bits                       */
int bits_dados; /* 7 ou 8 bits de dados                     */
{
  union{
    unsigned int nr;
    unsigned char hilo[2];
  }   valor;
```



```
unsigned char byte_de_param;
/* Calcula valor a ser enviado para registradores         */
/* de velocidade - Valor = 1843200 / (16 * bps)           */
valor.nr= (unsigned int)(1843200L/(bps*16L));
/* Liga bit do registrador da linha de controle que       */
/* permite acesso aos registradores de velocidade         */
outportb(porta+3,0x80);
/* Byte menos significativo para porta                    */
outportb(porta,valor.hilo[0]);
/* Byte mais significativo para porta                     */
outportb(porta+1,valor.hilo[1]);
/* Formata byte de parâmetros                             */
byte_de_param = bits_dados-5;
byte_de_param |= (stop - 1)<2;
if (p){
  byte_de_param |= 0x08;
  byte_de_param |= p&0x10;
}
/* Envia byte de parâmetros para registrador de           */
/* controle de linha                                      */
outportb(porta+3,byte_de_param);
}
```

IDÉIAS DE MELHORIAS

O programa SRE.C poderá ser modificado para imprimir uma lista dos diretórios e arquivos do disco, de forma que o usuário possa escolher o arquivo a ser transmitido ou recebido por meio das teclas de movimento do cursor.