Universidade Federal de Santa Catarina
Centro Tecnológico – CTC
Departamento de Engenharia Elétrica

http://spacelab.ufsc.br

“Porta serial UART”

Prof. Eduardo Augusto Bezerra

Florianópolis, março de 2021.

Gerência de Comunicação: SOs e Linguagens

Gerência de Comunicação: Visão Geral de Sistemas Operacionais e Linguagens

- Os sistemas operacionais dos primeiros computadores pessoais, como o MS-DOS rodando em processadores tais como o 8088 (PC-XT) e 80286 (primeiro PC-AT) permitiam ao desenvolvedor re-programar praticamente toda a máquina.

- Era possível, por exemplo, re-programar o teclado ou o relógio de tempo real do computador.

- Uma das razões para isso era o fato da arquitetura desses processadores não possuírem características tais como o modo protegido dos processadores atuais.

- Com o surgimento de processadores mais sofisticados e com a disseminação e evolução do sistema operacional MS-Windows para uma plataforma multitarefa 32 bits, e em modo protegido, o acesso direto ao hardware passou a ser um perigo para o bom funcionamento do sistema.

- Processadores ‘386 e superiores possuem o “modo protegido” onde o acesso direto a portas pode ser impedido.

Gerência de Comunicação: Visão Geral de Sistemas Operacionais e Linguagens

- Existem duas soluções para o problema de acesso as portas de entrada e saída nas versões 32 bits do Windows.

- A primeira solução é a criação de um *device driver* que executará com privilégio nível 0 para entrada/saída (modo *kernel*, ou super-usuário).

- Dados podem então ser enviados por programas do usuário (privilégio nível 3) para o *device driver*, e o *driver* executará a operação de entrada/saída.

- Outra possibilidade consiste na modificação da tabela de permissões de entrada/saída do sistema operacional. Existe um bit nessa tabela para cada porta do sistema.

- A tabela poderá ser modificada de forma a dar permissão de acesso irrestrito a uma determinada porta.

Gerência de Comunicação: Visão Geral de Sistemas Operacionais e Linguagens

- A opcão do *device driver* é a melhor. O *driver* deve ser escrito de forma a verificar se existem conflitos antes de acessar a porta.

- O conceito de *drivers* (*device drivers*) foi introduzido no Windows com o objetivo de isolar o hardware da aplicação do usuário.

- Desenvolvedores de aplicações para as versões mais recentes do Windows utilizam o Windows Driver Frameworks (WDF), que possibilita ao desenvolvedor escrever aplicativos que acessem virtualmente os recursos de entrada/saída e interrupções desse sistema operacional.

- Um *driver* é utilizado, basicamente, para realizar a interface entre um dispositivo de hardware e a CPU.

- Sistemas operacionais de 32 bits tais como o Windows NT, Windows XP e Unix, proíbem ou tentam evitar que os aplicativos acessem diretamente as portas de entrada e saída. É preciso utilizar os serviços dos *drivers*, que possuem os devidos privilégios, para obter acesso indireto às portas de entrada/saída.

Gerência de Comunicação: Visão Geral de Sistemas Operacionais e Linguagens

- A forma mais bem comportada e, conseqüentemente, a mais indicada para acessar um dispositivo externo, é por intermédio da API (*Application Programming Interface*) apropriada.

- É possível, por exemplo, escrever um *driver* USB utilizando diretamente as informações do protocolo para gerenciar um dispositivo conectado a essa porta de um computador pessoal.

- Porém é muito mais fácil utilizar as funções disponíveis na API da linguagem escolhida para a implementação.

- Essas funções se encarregam de "conversar" com o *driver*.

- A utilização das funções da API na construção do *driver* ou aplicação significa não apenas uma economia nas etapas de implementação e teste, mas também uma maior portabilidade.

Gerência de Comunicação: Visão Geral de Sistemas Operacionais e Linguagens

- Outro ponto é a questão de segurança existente em sistemas Unix e versões recentes do Windows.

- Como os programas de usuários comuns (sem permissões de super-usuário) normalmente não conseguem acessar diretamente as portas de entrada/saída da máquina, o uso de algumas APIs pode possibilitar ao usuário o acesso em mais alto nível da porta, ou melhor, do *driver* para acesso à porta.

- Para as portas que não possuem um padrão bem definido, é mais difícil de se encontrar APIs padronizadas, e o usuário acaba tendo que utilizar o conhecimento mais baixo nível sobre a porta para implementar o *driver* ou aplicação.

- O sistema de entrada/saída do Unix possui uma estrutura conhecida como *open-read-write-close*, uma vez que esses são os passos para a realização de uma operação de E/S com arquivos, dispositivos ou portas de E/S.

Gerência de Comunicação: Visão Geral de Sistemas Operacionais e Linguagens

- As operações de entrada/saída são realizadas por meio de arquivos. O sistema operacional atribui um descritor de arquivo, que é um número que identifica o arquivo, tanto para dispositivos, portas de E/S ou arquivos propriamente ditos. Um programa utiliza esse descritor de arquivo para acessar o "arquivo".

- Uma função de abertura de arquivo (*open*) é necessária para atribuir o descritor à variável no programa. Uma função de leitura (*read*) é utilizada para transferir informações do "arquivo" para o programa do usuário. Uma função de escrita (*write*) é utilizada para transferir informações do programa do usuário para o "arquivo". Uma função de fechamento de arquivo (*close*) é utilizada para devolver o descritor do arquivo ao sistema operacional.

- As funções de leitura/escrita utilizam o número do descritor do arquivo, o número de bytes a ser transferido, e o endereço de um buffer para o qual serão escritos ou lidos dados.

Gerência de Comunicação: Visão Geral de Sistemas Operacionais e Linguagens

```
#include <stdio.h>
#include <unistd.h>
#include <fcntl.h>
int main() {
   int fd, n;   char dado[1];    printf("Abrindo a porta paralela...\n");
   fd = open("/dev/port", O_RDWR | O_NOCTTY | O_NDELAY);
   if (fd != -1) {      // Sucesso!
      fcntl(fd, F_SETFL, 0);       dado[0] = 0xAA;   printf("Escreve: %x hexa\n", dado[0]);
      lseek (fd, 0x378, SEEK_SET);
      n = write(fd, dado, 1);  // envia 1 byte para a porta paralela
      sleep(1);          dado[0] = 0x55;
      printf("Escreve: %x hexa\n", dado[0]);
      lseek (fd, -1, SEEK_CUR);         n = write(fd, dado, 1);  // envia 1 byte para a porta paralela
      if (n < 0)             printf("Erro! write() falhou.\n");
      else {
         fcntl(fd, F_SETFL, FNDELAY);
         n = read(fd, dado, 1); // leitura de 1 byte da paralela
         printf("Leu: %x hexa\n", dado[0]);
         fcntl(fd, F_SETFL, 0);
      }
    } else         printf("Erro!! Nao conseguiu abrir a porta paralela!\n");
    return 0;
}
```

Gerência de Comunicação: Visão Geral de Sistemas Operacionais e Linguagens

#include <unistd.h> - Define tipos e constantes simbólicas.
Ex. SEEK_SET (offset); SEEK_CUR (atual + offset); SEEK_END (EOF + offset)

#include <fcntl.h> - Define opções de controle de arquivos:

O_RDWR – Abre para leitura e escrita

O_NOCTTY – Não atribui terminal de controle

O_NDELAY – Porta aberta no modo não bloqueante

fcntl(); Realiza operações em arquivos abertos // #include <unistd.h>

A função fcntl é utilizada para configurar a porta para possíveis operações de leitura.

O flag F_SETFL é utilizado em conjunto com o terceiro argumento para ativar ou desativar o modo de leitura da porta. Ao se utilizar 0 como terceiro argumento, uma operação de leitura na porta (read) irá bloquear a execução do programa até que um caracter seja recebido, um intervalo de tempo expire, ou um erro ocorra. Para realizar leituras não bloqueantes na porta, utilizar o flag FNDELAY, no lugar do 0.

Ex. acesso a periférico – porta paralela (inb/outb)

ioperm(endereço inicial, quantidade de endereços, 0/1)

- Instruções de acesso direto a I/O (in e out) resultam em *segmentation fault*;
- É necessário primeiro liberar o acesso a porta de I/O desejada;
- Programa em execução como *root* pode habilitar e desabilitar portas;
- Por exemplo, para permitir o acesso a 3 portas iniciando em 0x378 (os registradores utilizados para acesso ao /dev/lpta - porta paralela), usar:

ioperm(0x378, 3, 1);

- Para acessar apenas a porta 0x37A, usar:

ioperm(0x37A, 1, 1);

Essa função utiliza três parâmetros:
- endereço da primeira porta na faixa de portas desejada;
- a quantidade de portas na faixa desejada; e
- 1 para permitir acesso ou 0 para negar.

Ex. acesso a periférico – porta paralela (inb/outb)

Acesso direto com in/out no Linux – leitura da porta paralela:

```
#include <stdio.h>
#include <sys/io.h>          // em algumas distribuicoes pode ser  <asm/io.h>

int main() {

    char c;

    if(ioperm(0x378, 3, 1)) {          // acesso 0x378, 0x379, 0x37A
        printf("Erro! Precisa ser root.\n");
    } else {
        printf("Abriu a paralela\n");
            c = inb(0x379);            // Leitura do registrador de status
            printf("Recebido: %c", c);
        }
        return 0;
}
```

Ex. acesso a periférico – porta paralela (inb/outb)

- Apenas usuários pertencentes ao grupo *root* possuem permissão para executar programas com acesso direto ao hardware via inb e outb.

- Para possibilitar a execução por outros usuários utilizar a função *setuid*.

- Essa função altera a identidade do processo (*process ID*) para o valor passado como argumento.

- O valor 0 informa ao Linux que o processo pertence ao *root*.

- Para esse tipo de programa, um usuário *root* precisa compilar o programa, e a seguir alterar as permissões do arquivo para 4755 (*chmod 4755 nome_do_arquivo_executavel*), possibilitando que um usuário comum execute o programa com funções para acesso direto ao hardware.

Ex. acesso a periférico – porta paralela (inb/outb)

Exemplo:

// Compilar como root, e usar *chmod 4755 nome_do_executavel*

```
#include <stdio.h>
#include <sys/io.h>

int main(){
        if (setuid(0) == 0)          // sucesso! O processo do usuario virou root
            if(ioperm(0x378, 3, 1)) {          // acesso 0x378, 0x379, 0x37A
                printf("Erro! Precisa ser root.\n");
            } else {
                printf("Abriu a paralela\n");
                outb(0x55, 0x378);          // envia 01010101 para paralela
            }
        return 0;
}
```

Ex. acesso a periférico – porta paralela (open/read/write/close)

#include <unistd.h> - Define tipos e constantes simbólicas.
Ex. SEEK_SET (offset); SEEK_CUR (atual + offset); SEEK_END (EOF + offset)

#include <fcntl.h> - Define opções de controle de arquivos:

O_RDWR – Abre para leitura e escrita

O_NOCTTY – Não atribui terminal de controle

O_NDELAY – Porta aberta no modo não bloqueante

fcntl(); Realiza operações em arquivos abertos // #include <unistd.h>

A função fcntl é utilizada para configurar a porta para possíveis operações de leitura.

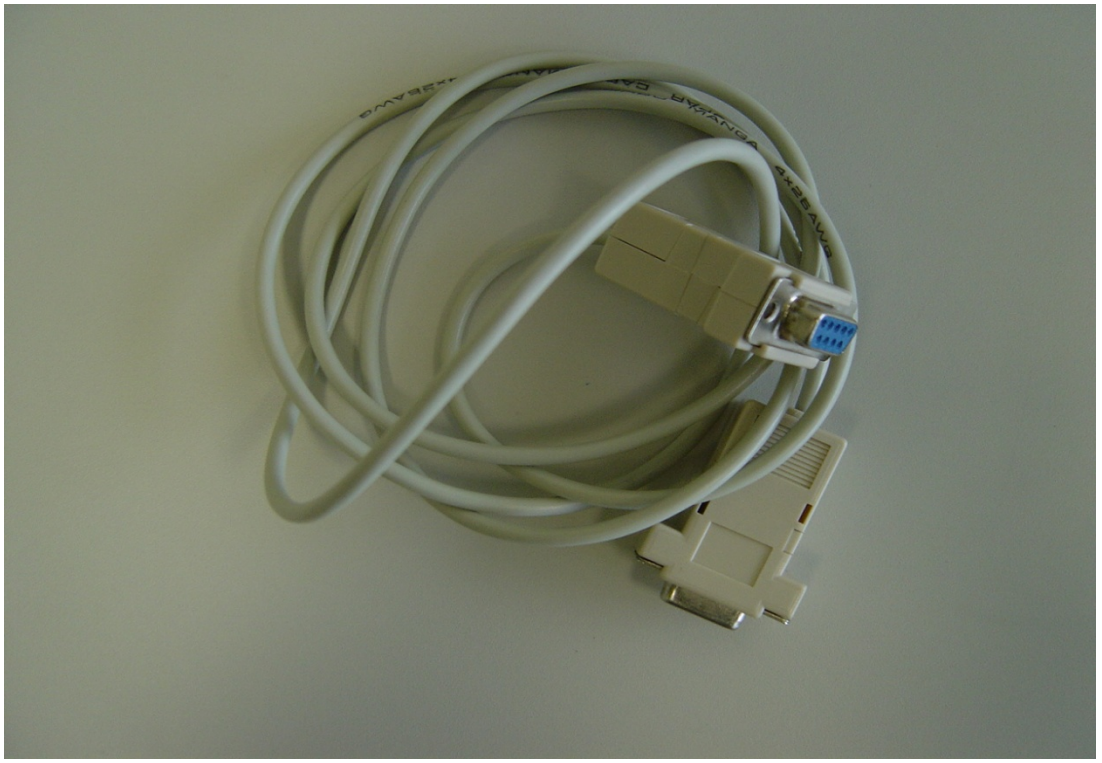
O flag F_SETFL é utilizado em conjunto com o terceiro argumento para ativar ou desativar o modo de leitura da porta. Ao se utilizar 0 como terceiro argumento, uma operação de leitura na porta (read) irá bloquear a execução do programa até que um caracter seja recebido, um intervalo de tempo expire, ou um erro ocorra. Para realizar leituras não bloqueantes na porta, utilizar o flag FNDELAY, no lugar do 0.

Acesso a periférico – Porta Serial RS-232C

Plataforma para testes

Lista de material para preparar um cabo para comunicação serial RS-232C entre dois microcomputadores:

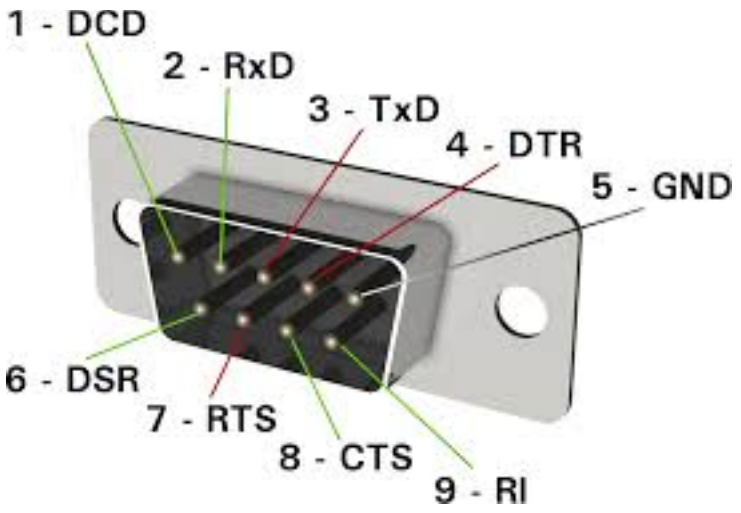
- Dois conectores DB-9 fêmea;
- Três fios (+/- 1 metro cada), ou um cabo como o da figura contendo pelo menos três fios;

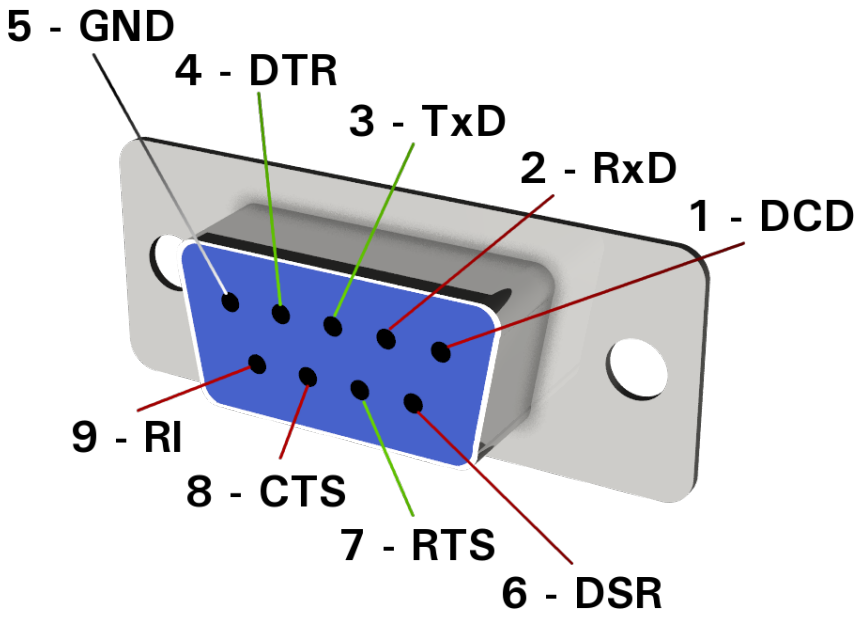


Instruções:

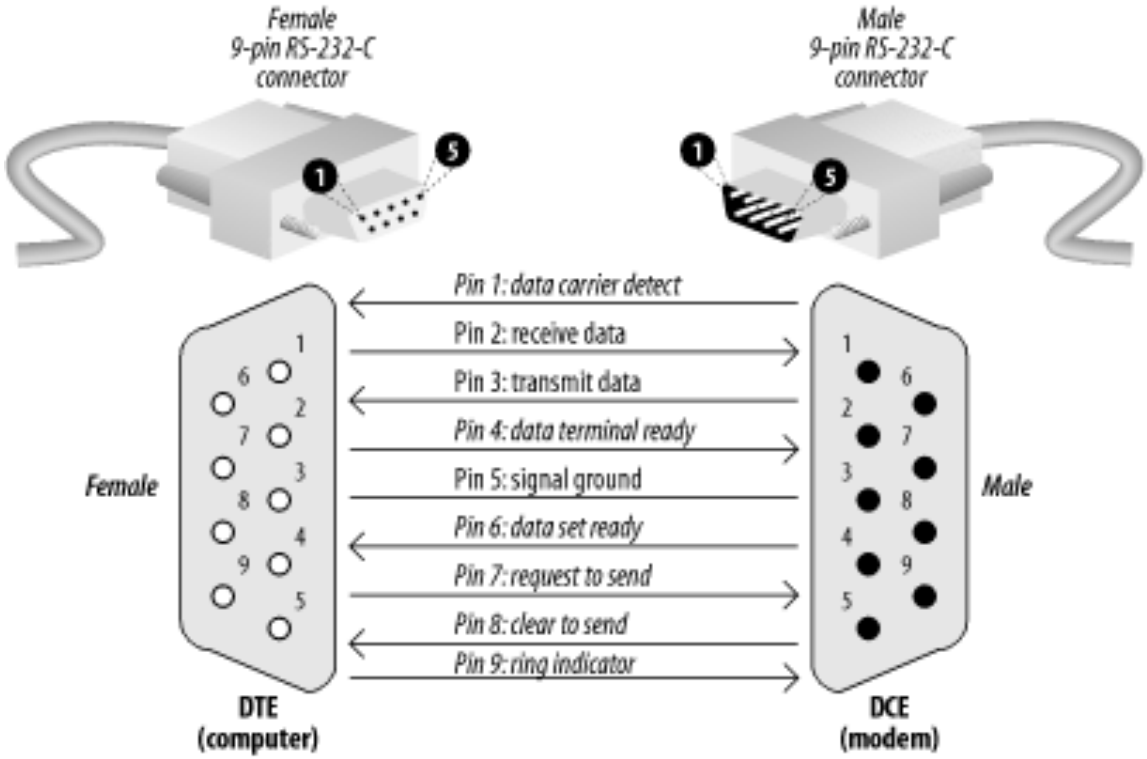
- Ligar o pino 2 do conector 1 ao pino 3 do conector 2;
- Ligar o pino 3 do conector 1 ao pino 2 do conector 2;
- Ligar o pino 5 do conector 1 ao pino 5 do conector 2;

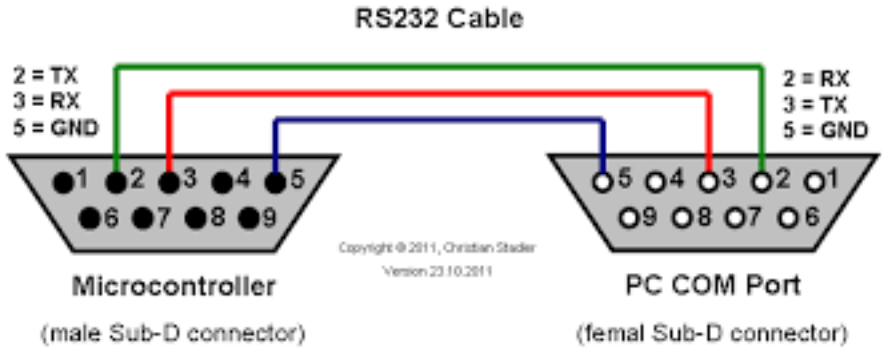
Pinos, conectores, cabos





Pinos, conectores, cabos





Acesso a periférico – porta serial

I/O no PC = 0H FFFH
Ou 0 a 65.535

/dev/ttyS1 ou COM 1

/dev/ttyS0 ou COM 0

Component	PC/XT	AT
DMA controller (8237A-5)	000-00F	000-01F
Interrupt controller (8259A)	020-021	020-03F
Timer	040-043	040-05F
Programmable Peripheral Interface (PPI 8255A-5)	060-063	none
Keyboard (8042)	none	060-06F
Realtime clock (MC146818)	none	070-07F
DMA page register	080-083	080-09F
Interrupt controller 2 (8259A)	none	0A0-0BF
DMA controller 2 (8237A-5)	none	0C0-0DF
Math coprocessor	none	0F0-0F1
Math coprocessor	none	0F8-0FF
Hard drive controller	320-32F	1F0-1F8
Game port (joysticks)	200-20F	200-207
Expansion unit	210-217	none
Interface for second parallel printer	none	278-27F
Second serial interface	2F8-2FF	2F8-2FF
Prototype card	300-31F	300-31F
Network card	none	360-36F
Interface for first parallel printer	378-37F	378-37F
Monochrome Display Adapter and parallel interface	3B0-3BE	3B0-3BF
Color/Graphics Adapter	3D0-3DF	3D0-3DF
Disk controller	3F0-3F7	3F0-3F7
First serial interface	3F8-3FF	3F8-3FF

Acesso a periférico – porta serial

• A porta serial sempre foi o meio preferido para a comunicação de dados desde a introdução dos primeiros PCs.

• No passado existiam poucas opções para comunicação serial, e o padrão RS-232C era a mais utilizada.

• Porém, essa opção sempre representou um gargalo com relação a velocidade de transmissão de dados. Os PCs sempre processaram dados milhares de vezes mais rapidamente do que o padrão RS-232C pode gerenciar.

• Novas tecnologias e padrões para comunicação serial resolvem um pouco esse problema: a interface infra-vermelho (IR); e a porta USB (Universal Serial Bus).

• Desde 1984, a porta RS-232C vem sendo utilizada como padrão em microcomputadores pessoais tipo IBM-PC, e esse foi o único padrão para comunicação serial em PCs até bem recentemente.

Acesso a periférico – porta serial

• A interface infra-vermelho (IR) forneceu ao RS-232C um novo meio para transmissão de sinais via ar (wireless) no lugar dos tradicionais cabos.

• A principal desvantagem é a baixa taxa de transferência, herdada do padrão RS-232C. Porém isso não chega a ser uma desvantagem para o tipo de acesso a dispositivos tratado nesse curso.

• USB é no momento uma ótima opção para comunicação serial. O padrão RS-232C foi idealizado para conectar diretamente 2 dispositivos, ponto-a-ponto.

• Já o USB funciona como um verdadeiro barramento podendo conectar ate 127 dispositivos, sem preocupações com diferentes pinagens de conectores e cabos.

• A velocidade de transferência de dados é muitas vezes superior a do padrão RS-232C.

• Apesar das diferencas entre esses padrões, eles possuem uma característica comum que é a transferência de dados de forma uni-dimensional (fila de bits).

Acesso a periférico – porta serial

Padrão	Taxa de transmissão	Meio	Dispositivos por porta
RS-232C	115.200 bps	Par trançado	1
IrDA	4 Mbps	Óptico	126
USB	12 Mbps	Cabo com 4 fios	127

Acesso a periférico – porta serial

• O padrão RS-232C define diversos parâmetros para a comunicação serial, entre eles os níveis de tensão que representam o 0 e o 1 lógicos, os conectores a serem utilizados (formato e número de pinos), e o formato dos dados a serem utilizados.

• Com relação a portabilidade, a existência do padrão RS-232C simplifica a vida do desenvolvedor de aplicações que utilizam a porta serial. A porta paralela e bem mais simples de se utilizar do ponto de vista do desenvolvedor, porém a diversidade de padrões pode fazer com que uma aplicação funcione em determinadas máquinas, e não funcione em outras.

• Por exemplo, uma vez identificada a porta paralela em uso, e o padrão utilizado, o desenvolvedor não precisa se preocupar em implementar nenhum protocolo especial para acionar ou receber informações do dispositivo externo. Todos os dados trafegados pela porta paralela ficarão armazenados (“bufferizados”) no registrador de dados da porta.

• Já em uma porta serial existe um módulo responsável pela gerência da comunicação, e o dispositivo externo deverá “falar” o mesmo protocolo (a mesma língua) desse módulo de comunicação serial.

Acesso a periférico – porta serial

• No caso dos computadores padrão IBM-PC, o módulo responsável pela gerência da porta paralela é implementado com um circuito integrado baseado no Intel 8250.

• O 8250 implementa uma UART (Universal Asynchronous Receiver / Transmitter) que, basicamente, se encarrega de serializar os bytes recebidos (conversão paralelo para serial), adicionar os bits de controle definidos no protocolo RS-232C (start bit, stop bit e paridade), e enviar os bits um a um para o pino de transmissão do conector.

• O 8250 realiza também a função inversa, ou seja, receber os bits serialmente, remover os bits de controle, remontar o byte recebido e disponibilizar esse byte no registrador de dados do 8250.

• A UART 8250 passou a fazer parte dos computadores padrão IBM-PC em 1981. Esse dispositivo possui capacidade de armazenamento para apenas um byte e não é o mais adequado, por exemplo, para comunicação utilizando os modems da época com velocidade mais rápida que esse dispositivo podia gerenciar. Nesse caso caracteres recebidos em uma comunicação podiam ser perdidos.

Acesso a periférico – porta serial

• Para solucionar esse problema, a partir de 1984 os computadores pessoais passaram a utilizar a UART 16450 que é uma versão melhorada da 8250.

• Essa nova UART também possui apenas um byte para armazenamento dos dados recebidos, mas o circuito interno e bem mais rápido do que o da 8250, resolvendo o problema de perda de dados para os modems da época.

• Com a utilização de sistemas multi-tarefas, em algumas situações os PCs passaram a não ter tempo para ler o byte recebido na UART antes que um novo byte chegasse.

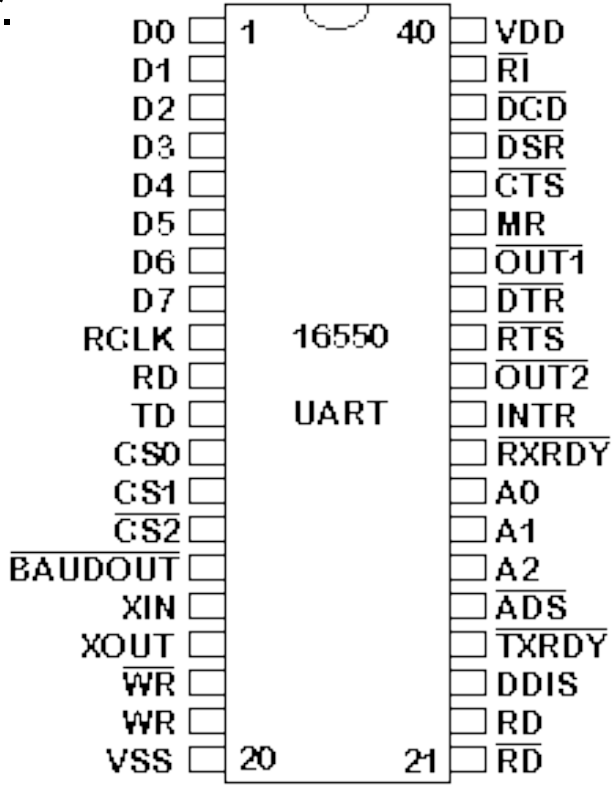
• O PC podia estar ocupado realizando alguma outra tarefa e um novo byte recebido na porta serial iria sobrepor o byte pronto para ser lido antes do PC o fazê-lo.

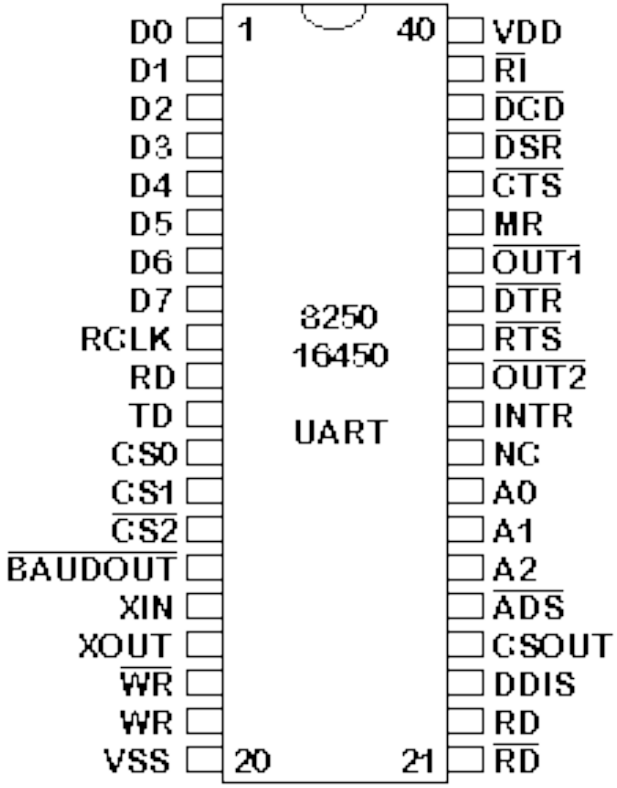
• Em 1987 os PCs passaram a utilizar a UART 16550A com capacidade de armazenamento para mais do que um byte recebido, e posteriormente a UART 166550 que possui uma FIFO com capacidade para armazenar ate 16 bytes recebidos.

Acesso a periférico – porta serial

• Essa fila é uma boa solução para armazenar os dados recebidos de modems de alta velocidade em ambientes multi-tarefa.

• As UARTs utilizadas nos PCs atuais são compativeis com a 166550, porém são encapsuladas em dispositivos do tipo ASIC (Application Specific Integrated Circuit) ficando difícil sua identificação visual na placa do computador.





Acesso a periférico – porta serial

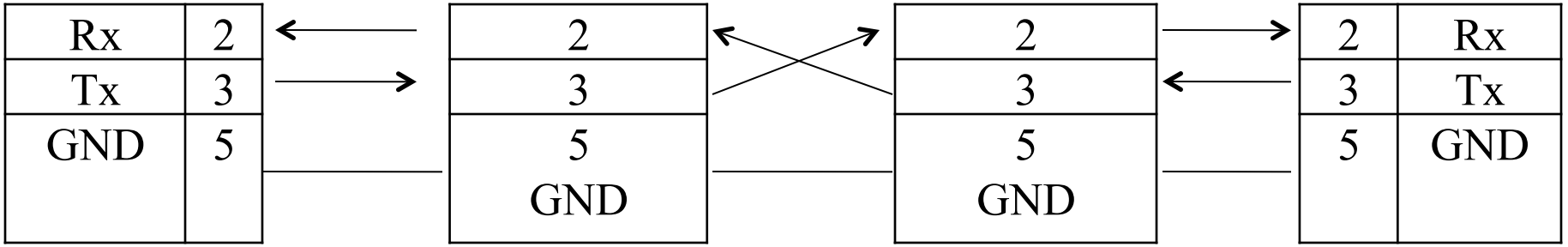
• Devido a utilização do padrão RS-232C, é possível interfacear um dispositivo externo a um IBM-PC via porta serial, bastando para isso incluir um 8250, ou qualquer outra UART compatível com esse padrão, no projeto do dispositivo.

25-pinos (DB-25)	9-pinos (DB-9)	Símbolo	Função
2	3	Tx	Transmite dados
3	2	Rx	Recebe dados
4	7	RTS	Permissão para envio
5	8	CTS	Permissão fornecida
6	6	DSR	Dado pronto
7	5	GND	Terra
8	1	CD	Deteção de portadora
20	4	DTR	Terminal pronto
22	9	RI	Indicador de Ring

Acesso a periférico – porta serial

• No projeto da interface para conexão a um dispositivo a ser controlado via porta serial, bastam 3 pinos (3 fios): pino 5 – Terra, pino 3 – transmissão de dados, pino 2 – recepção de dados.

• A figura a seguir mostra como ficaria o cabo serial com os devidos conectores para ligar dois PCs.



Acesso a periférico – porta serial

- A UART 8250 trabalha com níveis de tensão TTL (0V para 0 lógico, e +5V para 1 lógico)
- É preciso utilizar um conversor de níveis de tensão (CI MAX232) para realizar a transformação de 0V para +15V, e de +5V para –15V, e vice-versa.
- A 8250 necessita estar conectada a um dispositivo mestre de forma a ser programada sua forma de operação. Esse dispositivo mestre, que no caso dos PCs é a CPU (ex. Pentium), precisa programar a UART para trabalhar:
    - com uma certa velocidade de transferência de dados,
    - com um determinado tamanho de palavra de dados,
    - um determinado numero de stop bits, e
    - uma determinada paridade.

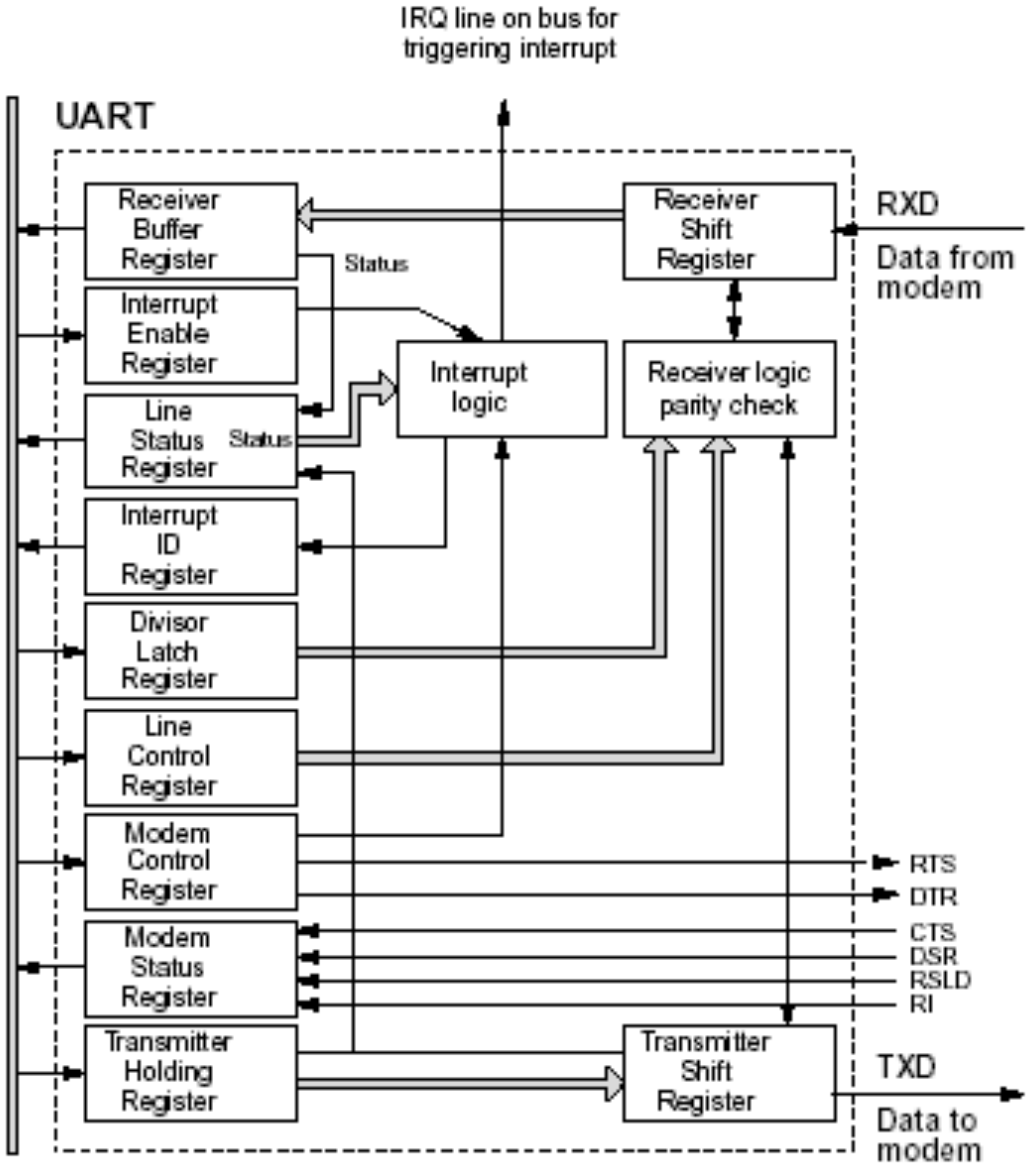
Acesso a periférico – porta serial

• Os mesmos parâmetros devem ser utilizados para programar a UART em utilização do outro lado da linha.

• Caso alguma dessas informações esteja diferente em um dos dois lados, não haverá comunicação pois as duas UARTs estarão utilizando protocolos diferentes.

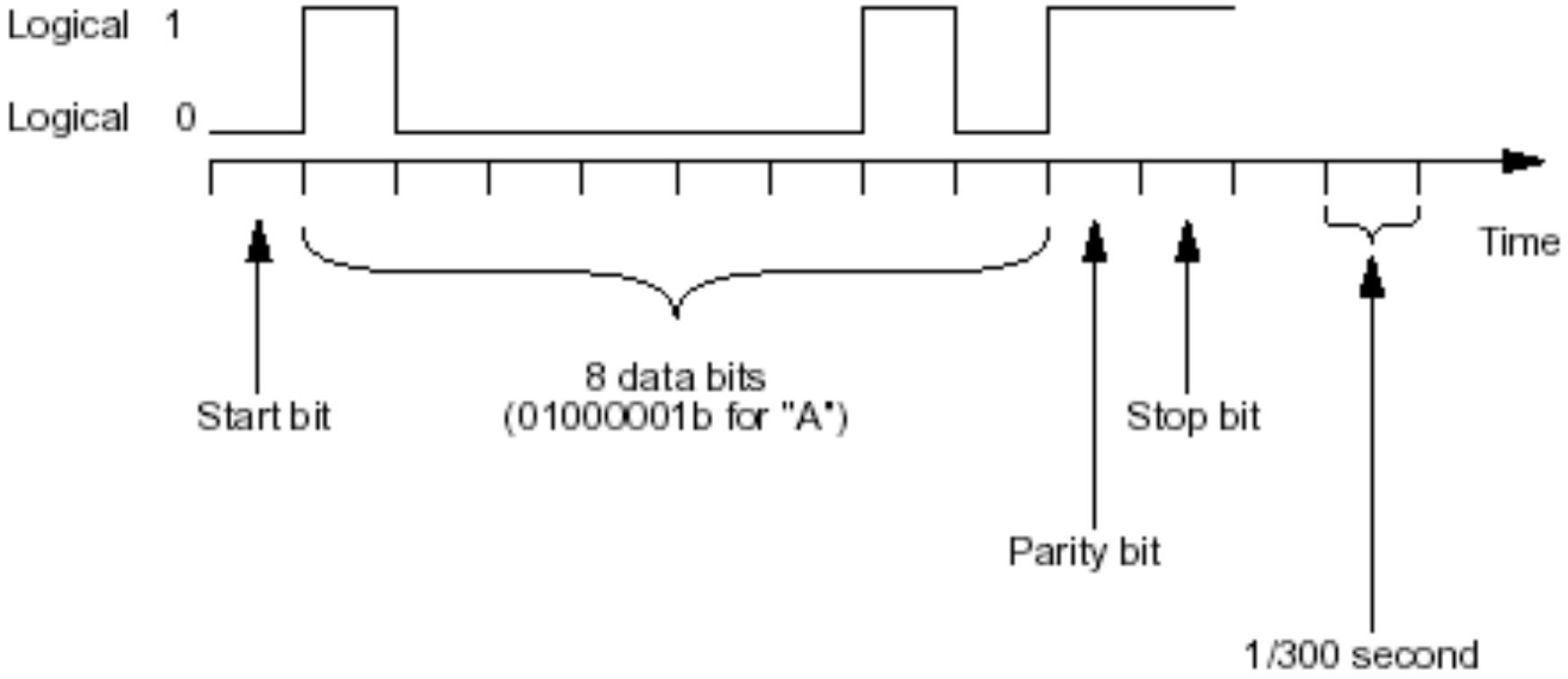
• Diagrama de blocos mostrando os registradores da UART 8250

Acesso a periférico – porta serial



Acesso a periférico – porta serial

• Envio da letra ‘A’ em uma UART programada para uma palavra de tamanho 8 bits, com 1 stop bit, paridade ímpar e com uma velocidade de 300 bps. Como a letra ‘A’ na tabela ASCII possui 2 bits em 1 e os demais em 0, o bit de paridade está em 1 de forma a gerar um número ímpar de bits em 1.



Acesso a periférico – porta serial

- ***Registrador de dados***:

  • Registrador de Recepção: ao se realizar uma leitura na 8250, obtêm-se o byte contido nesse registrador proveniente da conversão de serial para paralelo dos bits de entrada (pino Rx).

  • Registrador de transmissão: uma operação de escrita carrega um byte nesse registrador para ser convertido de paralelo para serial pela 8250, e posterior transmissão pelo pino Tx).

Acesso a periférico – porta serial

- ***Status***:

- **Registrador de Status de Linha**: utilizado para indicar condições de operação da linha.

7	6	5	4	3	2	1	0
0	TxE	TBE	Break	Erro de framming	Erro de paridade	Erro de overrun	RxRdy

Bit 0: 1 = existe byte pronto para ser lido no registrador de recepção.

Bit 1: 1 = um byte no registrador de recepção foi sobre escrito por um novo byte. 0 primeiro byte foi perdido.

Bit 2: 1 = erro de paridade.

Bit 3: 1 = stop bit inválido

Bit 4: 1 = interface detecta a linha em zero durante um tempo maior que a duração de um byte assíncrono.

Bit 5: Buffer de transmissão vazio. 1 = um byte é movido do buffer de transmissão para o registro de deslocamento, onde o byte é transmitido serialmente.

Bit 6: Transmissor vazio. 1 = registro de deslocamento vazio.

Acesso a periférico – porta serial

- ***Status***:
- **Registrador de Status do Modem**: utilizado para indicar status do modem.

7	6	5	4	3	2	1	0
DCD	RI	DSR	CTS	Δ DCD	Δ RI	Δ DSR	Δ CTS

Bits 0 – 3: 1 = ocorreu uma mudança no respectivo pino desde a ultima leitura na porta.

Bit 4 - 7: indica o status dos pinos da porta.

Acesso a periférico – porta serial

- ***Status***:

- **Registrador de Identificação de Interrupção**: Apos uma interrupção, o bit 0 recebe 0, e os bits 1 e 2 determinam a fonte da interrupção.

Bit 2	Bit 1	Bit 0	Identificador
0	0	1	Sem interrupção
1	1	0	Condição de erro
1	0	0	Byte recebido
0	1	0	Buffer de transmissão vazio
0	0	0	Mudança na entrada da porta serial

Acesso a periférico – porta serial

- ***Controle***:
- **Registrador de Controle de Linha:** utilizado para formatação dos dados.

7	6	5	4	3	2	1	0
DLAB	Break	Paridade			Stop bit	Bits de dados	

Bits 0 - 1: quantidade de bits por caracter.
00 = 5 bits
01 = 6 bits
10 = 7 bits
11 = 8 bits

Acesso a periférico – porta serial

- ***Controle***:

7	6	5	4	3	2	1	0
DLAB	Break	Paridade			Stop bit	Bits de dados	

Bit 2: quantidade de bits por caracter.
0 = 1 stop bit
1 = 2 stop bits

Obs. Se a quantidade de bits por caracter for igual a 5, o numero de stop bits será automaticamente 1 ½ stop bits.

Acesso a periférico – porta serial

- ***Controle***:

7	6	5 4 3	2	1 0
DLAB	Break	Paridade	Stop bit	Bits de dados

Bits 3 - 5: determinam a paridade.
000 = sem paridade
001 = paridade impar
011 = paridade par
101 = marca (mark)
111 = espaço (space)

Bit 6: 1 = saída Tx vai para o nível lógico 0.

Bit 7: 1 = registro de transmissão recebe o byte de menor ordem (LSB) da taxa de transmissão e o registro de controle de interrupção recebe o byte de maior ordem (MSB).

Acesso a periférico – porta serial

- ***Controle***:

- **Registrador de Controle de Interrupção**: utilizado para habilitar os quatro tipos de interrupção do 8250.

7	6	5	4	3	2	1	0
0	0	0	0	Pinos de entrada	Erro na recepção	TBE	RxRdy

Bit 0: 1 = uma interrupção e gerada quando um byte estiver disponível no registrador de recepção.
Bit 1: 1 = uma interrupção e gerada quando a 8250 puder receber um novo byte para transmissão.
Bit 2: 1 = uma interrupção e gerada quando ocorrer um erro de paridade, overrun (sobre escrita) ou stop bit.
Bit 3: 1 = uma interrupção e gerada quando qualquer entrada da porta serial mudar de estado.

Acesso a periférico – porta serial

- ***Controle***:
- **Registrador de Controle do Modem**

7	6	5	4	3	2	1	0
0	0	0	Loop	GP02	GP01	RTS	DTR

Bit 0: 1 = ativa a saida DTR.
Bit 1: 1 = ativa a saida RTS.
Bit 2: Saida definida pelo usuário. Normalmente em 0.
Bit 3: Saida definida pelo usuário. Normalmente em 0.

Acesso a periférico – porta serial

Para programar a taxa de transferência (velocidade da comunicação), é preciso utilizar o bit 7 (DLAB) do Registrador de Controle de Linha, em conjunto com o Registradores de Controle de Interrupção e com o Registrador de Dados, da seguinte forma:

• Colocar o bit DLAB em 1 para indicar que a parte baixa (LSB) da programação velocidade será colocada no Registrador de Dados, e a parte alta (MSB) no Registrador de Controle de Interrupção.

• Escrever no Registrador de Controle de Interrupção o valor desejado de acordo com a tabela a seguir (00H para 9600bps, por exemplo).

• Escrever no Registrador de Dados o valor desejado de acordo com a tabela a seguir (0CH para 9600bps, por exemplo).

Acesso a periférico – porta serial

velocidade (bps)	Divisor (Dec)	Divisor MSB Reg. Contr. Interrupção	Divisor (LSB) Reg. Dados
50	2304	09h	00h
300	384	01h	80h
600	192	00h	C0h
2.400	48	00h	30h
4.800	24	00h	18h
9.600	12	00h	0Ch
19.200	6	00h	06h
38.400	3	00h	03h
57.600	2	00h	02h
115.200	1	00h	01h

Acesso a periférico – porta serial

Endereços da UART 8250 (porta COM1):

Dados (escrita/leitura)	03F8H
Registrador de Controle de Interrupção	03F9H
Registrador de Identificação de Interrupção	03FAH
Registrador de Controle de linha	03FBH
Registrador de Controle do Modem	03FCH
Registrador de Status da Linha	03FDH
Registrador de Status do Modem	03FEH

Obs. A COM2 inicia no endereço 02F8H, a COM3 no endereço 03E8H e a COM4 no endereço 02E8H.

Acesso a periférico – porta serial

O funcionamento da troca de dados assíncrona entre duas UARTs é relativamente simples.

• No lado do transmissor o pino de envio de dados Tx permanece em nível lógico 1 até existir um carácter pronto para transmissão.

• Nesse instante o pino é colocado em zero, representando o start bit.

• Os bits de dados são enviados logo após o start bit, um após o outro. Um bit de paridade, opcionalmente, segue os bits de dados.

• Após isso um ou mais stop bits são enviados.

• O bit de paridade é a soma dos bits de dados e indica se o dado contém um numero ímpar ou par de bits 1. Para paridade par esse bit será 0. Para paridade ímpar o bit sera 1.

Porta serial – Linux – acesso direto inb/outb

```
/*
        serial_io.c
        Programa em C para gerencia da porta serial RS-232C
        utilizando acesso direto ao hardware pelas instrucoes
        in e out.
        Eduardo Augusto Bezerra
        Maio de 2003
        compilador gcc version 3.2 20020903 (Red Hat Linux 8.0 3.2-7)
*/
#include <stdio.h>
#include <stdlib.h>
#include <sys/io.h>
#include <unistd.h>
#include <malloc.h>
#include <string.h>

#define end_ctrl_50     0x03FD              /* Enderecos da UART 8250 */
#define end_dado_50     0x03F8
#define LRC                         0x03FB
#define MSB                         0x03F9
#define LSB                         0x03F8
#define MCR                         0x03FC
#define LSR                         0x03FD
#define MSR                         0x03FE
#define IOR                         0x03F8
#define IIR             0x03FA
#define IER                         0x03F9
```

Porta serial – Linux – acesso direto inb/outb

```
int main(){
        unsigned char caracter = ' ';      int a;                    a = setuid(0);
        printf("setuid = %d\n", a);
        if (ioperm(end_dado_50, 2, 1)) {
             perror("ioperm");
             printf("Erro: Nao liberou a porta serial!");
        }
        /* inicializacao da 8250 */
        outb(0x80, LRC);                /* 8 bits de dados */
        outb(0x00, MSB);                              /* 1 stop bit                    */
        outb(0x0C, LSB);                /* sem paridade              */
        outb(0x07, LRC);
        /* reseta a 8250  */
        inb(LSR); inb(MSR); inb(IOR); inb(IIR);
        outb(0x00, IER);                /* sem interrupcoes */
        caracter = ' ';
        /* transmissao de um caracter */
        while (caracter != 0x1B){     /* repete ate' pressionar ESC */
                printf("\nEntre com o caracter a ser transmitido ... ");
                scanf("%c", &caracter);
                printf("%c", caracter);
                outb(caracter, end_dado_50);        /* envia um caracter */
        }
        return 0;
}
```

Porta serial em C++

Porta serial – Linux – C++

Arquivo: mySerial.h

(autor desconhecido)

```
#include <string>

class  mySerial {

public:

  int handle;
  std::string  deviceName;
  int baud;

  mySerial(std::string deviceName, int baud);
  ~mySerial();

  bool Send( unsigned char  * data,int len);
  bool Send(unsigned char value);
  bool Send( std::string value);
  int Receive( unsigned char  * data, int len);
  bool IsOpen(void);
  void Close(void);
  bool Open(std::string deviceName, int baud);
  bool NumberByteRcv(int &bytelen);
};
```

Porta serial – Linux – C++

Arquivo: mySerial.cpp

(autor desconhecido)

```
extern "C" {
#include <asm/termbits.h>
#include <sys/ioctl.h>
#include <unistd.h>
#include <fcntl.h>
}
#include <iostream>
using namespace std;

#include "mySerial.h"
```

```
mySerial::mySerial(string deviceName, int baud {
   handle = -1;
   Open(deviceName,baud);
}
```

```
mySerial::~mySerial(){
  if(handle >=0)
      Close();
}
```

```
bool mySerial::IsOpen(void){
   return( handle >=0);
}
```

```
bool mySerial::Send( unsigned char value){
   if(!IsOpen()) return false;
   int rlen= write(handle,&value,1);
   return(rlen == 1);
}
```

```
void mySerial::Close(void){
   if(handle >=0)
      close(handle);
   handle = -1;
}
```

```
bool mySerial::Send(std::string value){
   if(!IsOpen()) return false;
   int rlen=
write(handle,value.c_str(),value.size());
   return(rlen == value.size());
}
```

Porta serial – Linux – C++

Arquivo: mySerial.cpp

(autor desconhecido)

```
bool mySerial::Send( unsigned char  * data,int len){
   if(!IsOpen()) return false;
   int rlen= write(handle,data,len);
   return(rlen == len);
}
```

```
int  mySerial::Receive( unsigned char  * data, int len){
   if(!IsOpen()) return -1;

   // this is a blocking receives
   int lenRCV=0;
   while(lenRCV < len){
       int rlen = read(handle,&data[lenRCV],len - lenRCV);
       lenRCV+=rlen;
     }
   return  lenRCV;
}
```

```
bool mySerial::NumberByteRcv(int &bytelen){
   if(!IsOpen()) return false;
   ioctl(handle, FIONREAD, &bytelen);
   return true;
}
```

Porta serial – Linux – C++

```
bool mySerial::Open(string deviceName , int baud){
    struct termios tio;        struct termios2 tio2;
    this->deviceName=deviceName;       this->baud=baud;
    handle  = open(this->deviceName.c_str(),O_RDWR | O_NOCTTY /* | O_NONBLOCK */);

    if(handle <0)        return false;
    tio.c_cflag =  CS8 | CLOCAL | CREAD;
    tio.c_oflag = 0;
    tio.c_lflag = 0;       //ICANON;
    tio.c_cc[VMIN]=0;
    tio.c_cc[VTIME]=1;     // time out every .1 sec
    ioctl(handle,TCSETS,&tio);

    ioctl(handle,TCGETS2,&tio2);
    tio2.c_cflag &= ~CBAUD;
    tio2.c_cflag |= BOTHER;
    tio2.c_ispeed = baud;
    tio2.c_ospeed = baud;
    ioctl(handle,TCSETS2,&tio2);

//   flush buffer
    ioctl(handle,TCFLSH,TCIOFLUSH);

    return true;
}
```

Arquivo: mySerial.cpp

(autor desconhecido)

Porta serial – Linux – C++

```
#include "mySerial.h"
#include <iostream>
using namespace std;

int  main(void) {

    mySerial serial("/dev/ttyAMA0",115200);

    // One Byte At the time
     serial.Send(128);
     serial.Send(132);

    // An array of byte
    unsigned char  dataArray[] = { 142,0};
    serial.Send(dataArray,sizeof(dataArray));

    // Or a string
    serial.Send("this is it\r\n");

    return 0;
}
```

Arquivo: test.cpp

(autor desconhecido)