Entradas/saídas (cont.)

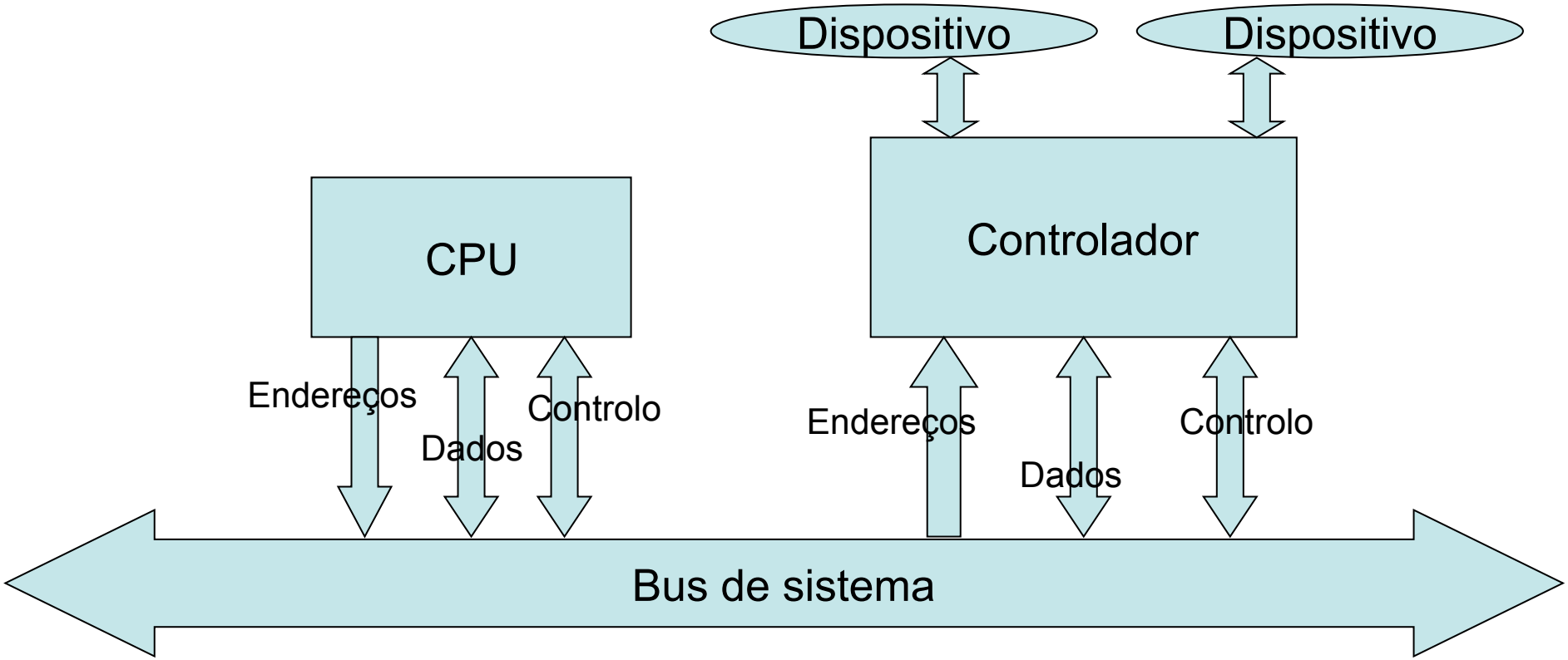
Programação de entradas/saídas por espera activa (conclusão).

Programação de um controlador série.

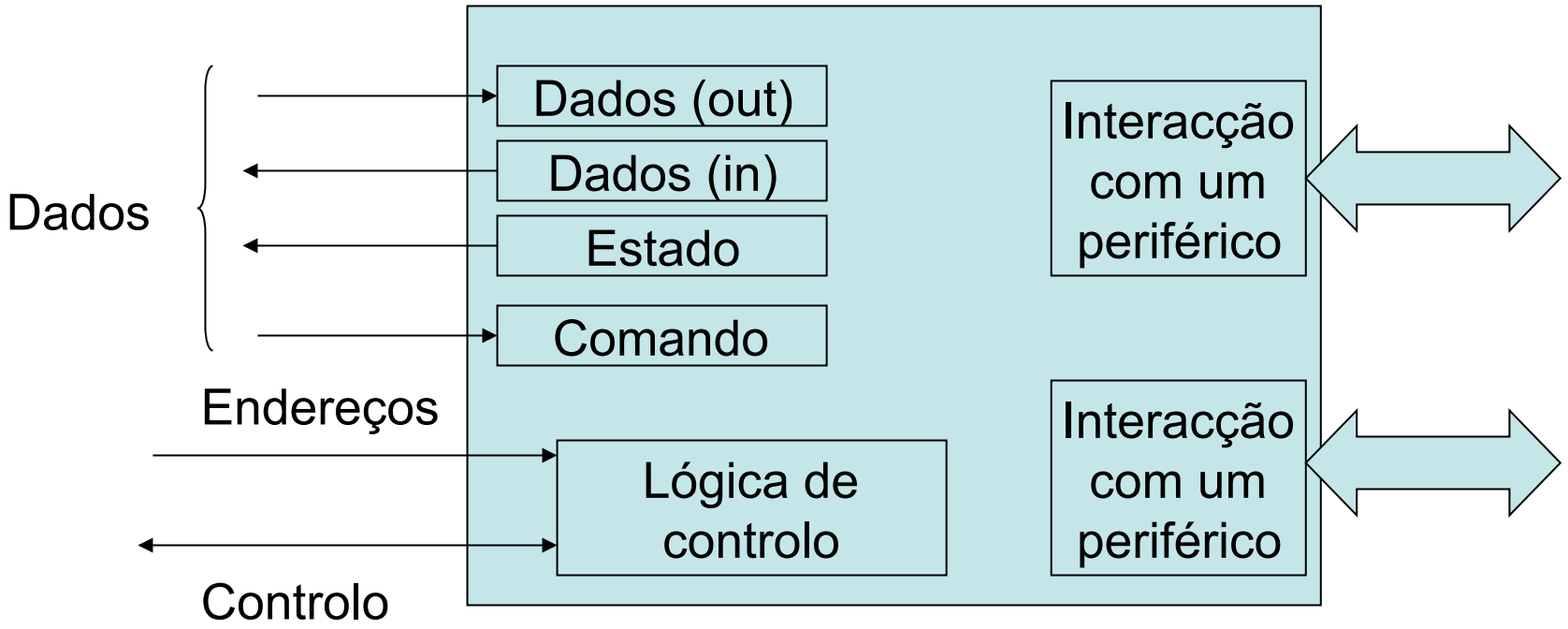
Funções do controlador

- Controlo e Temporização
- Comunicação com o CPU
- Comunicação com o dispositivo
- Armazenamento temporário de dados (Data Buffering)
- Detecção de erros

Controlador de periféricos



Controlador genérico



Interface com
o bus de sistema

Interface com
os periféricos

Registos de uma interface (1)

- Uma interface ocupa uma série de endereços (normalmente consecutivos) no espaço de endereçamento
  - Registo(s) de dados (leitura): onde se lêm os dados que vêm do exterior
  - Registo(s) de dados (escrita): onde se escrevem os dados a enviar para o exterior
  - Registo de comando (escrita): onde se dão comandos à interface
  - Registo de estado (leitura): conjunto de bits que dão informação sobre o estado do controlador: livre/ ocupado, transferência com/sem erro , ...

Registos de uma interface (2)

- Muitas vezes, o registo de comando e de estado ocupam o mesmo endereço
  - Uma escrita no endereço controla o periférico
  - Uma leitura do mesmo endereço retorna o estado do controlador
- Muitas vezes, uma operação de leitura, implica implicitamente uma ordem para ler mais um valor. Exemplo:
  - Quando chega um código ASCII pela porta série esse valor é colocado no registo DADOS (IN)
  - Quando o CPU lê esse registo de dados, desencadeia automaticamente um novo de leitura de um código ASCII que virá para o registo de dados.

Passos para realizar uma operação de E/S

- CPU pede ao contrololador de E/S o seu estado
- O controlador de E/S retorna o seu estado (CPU lê o reg. ESTADO)
- Se o controlador está disponível, o CPU pede uma transferência de dados (CPU escreve no reg. COMANDO)
- Se é uma operação de saída o CPU escreve os dados a sair no controlador (CPU escreve DADOS OUT)
- O controlador de E/S transfere dados de/para o periférico
- Se é uma operação de entrada, o CPU lê os dados que chegaram do controlador (CPU lê o registo DADOS IN)

Programação de E/S por espera activa

- O CPU tem controlo directo sobre as operações de E/S
  - Detecção do estado
  - Comandos de leitura/escrita
  - Transferência de dados
- O CPU espera pelo controlador para acabar a operação.
- Desperdiça tempo de CPU

Detalhes da programação por espera activa

- O CPU requer operação de E/S
- O controlador efectua a operação
- O controlador posiciona os seus bits de estado
- O CPU verifica os bits de estado periodicamente
- O controlador de E/S não informa o CPU directamente e não interrompe o que o CPU está a fazer
- O CPU pode esperar ou voltar mais tarde

Comandos de E/S

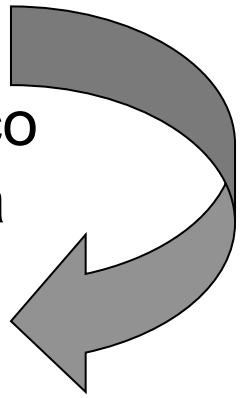
- CPU emite um endereço
  - Identifica o controlador (e o dispositivo se houver mais de um dispositivo por controlador)
- CPU emite um comando
  - Controlo – dizendo ao controlador o que fazer
    - e.g. colocar um disco em movimento
  - Teste- verificar estado
    - e.g. disponível? Erro?
  - Leitura/escrita
    - O controlador transfere dados de um “buffer” de/para o dispositivo

Programação por espera activa

1. O programa lê o estado do periférico:
   - CPU pede ao controlador (IN) o valor no registo ESTADO
   - O controlador retorna o seu estado
2. ***Se*** o controlador está disponível, o programa faz um pedido de transferência:
   - Se é para uma saída, o programa envia os dados:
     - CPU escreve os dados a sair para registo DADOS (OUT)
     - O CPU escreve o comando para enviar no registo COMANDO (OUT)
   - Se é uma entrada, o programa lê os dados que chegaram:
     - CPU lê o registo DADOS (IN)
     - CPU escreve o comando para ler de novo no reg. COMANDO (OUT)
3. O controlador transfere dados de/para o periférico

Programação por espera activa

1. O CPU requer uma operação
2. O CPU interroga repetidamente o controlador do periférico (bits de ESTADO) para saber se a operação anterior já terminou
   – O CPU pode esperar (com grande desperdício de tempo de CPU) ou voltar a testar mais tarde



- O controlador fica a efectuar a operação
- O controlador posiciona os seus bits de estado (registo de ESTADO)
  – O controlador de E/S não informa o CPU directamente e não interrompe o que o CPU está a fazer

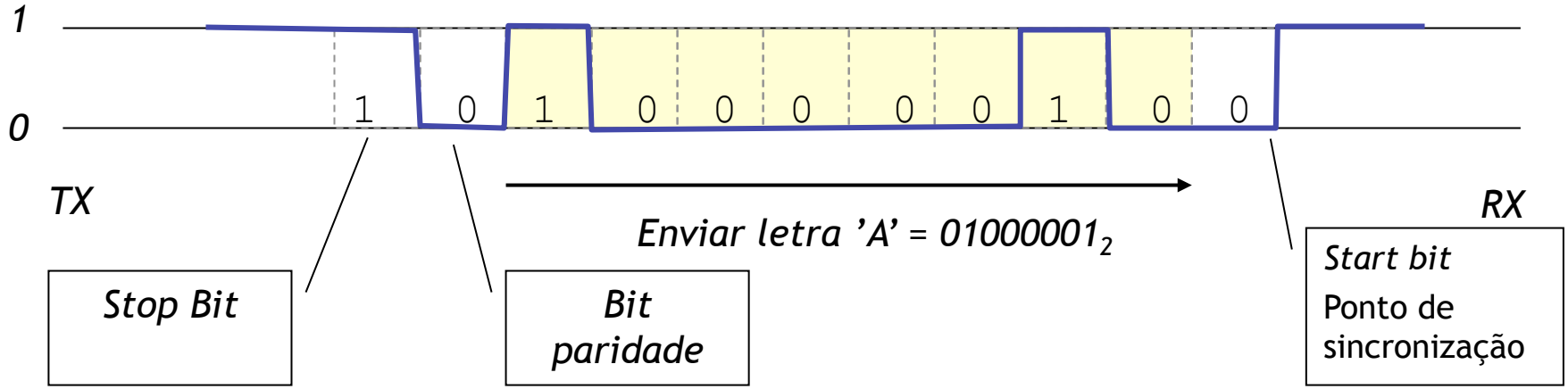
Inconvenientes do uso da espera activa

- Os dispositivos de E/S são muito lentos e o CPU vai passar grande parte do tempo nos ciclos de espera
- Enquanto se espera há potencial para o CPU executar muitas instruções
- Os controladores que só suportam espera activa são muito simples do ponto de vista *hardware*, mas implicam um grande desperdício de tempo de CPU
- Se o CPU executa outras operações o periférico pode entretanto ficar disponível e logo, desocupado → desperdício do periférico
- Mau para o controlo "simultâneo" de múltiplos periféricos

Programação por espera activa de um controlador série

Porta série:

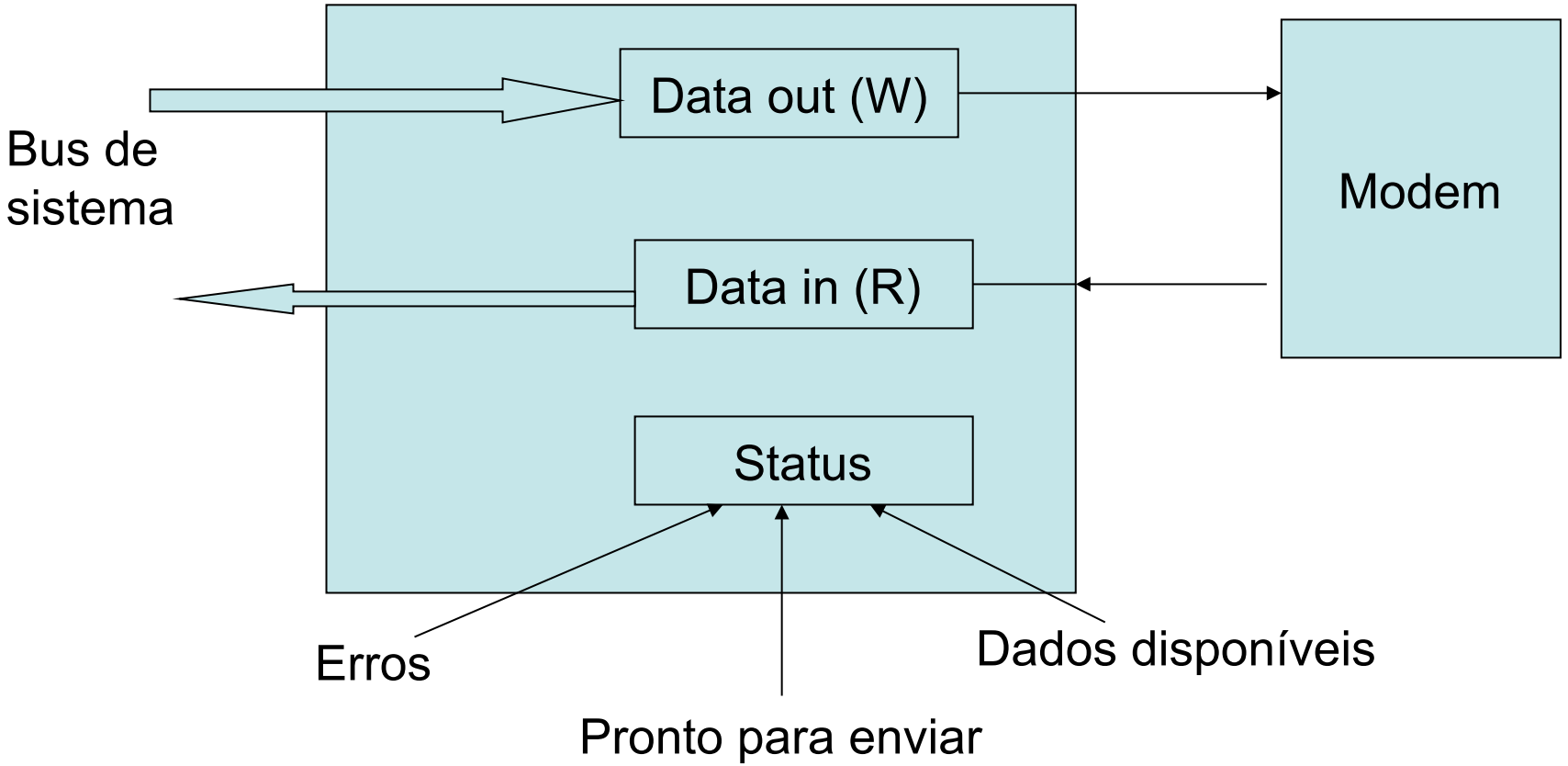
- Usada na ligação a modems, impressoras série, ...
- Faz uma conversão paralelo/série (saída) e série/paralelo (entrada)
- Comunicação assíncrona
  - Cada conjunto de bits de dados (por exemplo 8) é precedido por 1 ou 2 “start bits” e 1 ou 2 “stop bits”
  - Isto ajuda à sincronização entre emissor e receptor



Programação por espera activa de um controlador série

- Porta série:
  - Programação de vários parâmetros:
    - Baud rate (inverso da “duração” de um bit)
    - Paridade (detecção de erros)
    - Número de bits de dados: ASCII (7 ou 8)
    - Número de “start bits” e “stop bits”
  - No MS-DOS
    - `mode com1:9600,n,8,1`
    - Ritmo de transmissão: 9600 bit/s; sem paridade; 8 bits de dados; 1 start bit

Porta série



No PC o controlador série é constituído por um único circuito integrado, chamado UART (Universal Asynchronous Receiver Transmiter).
O nome no Windows e MSDOS é COM1:
No Linux /dev/ttys0

Porta série do PC

- Registo de dados de saída (DATA OUT)
  - 0x3F8
  - A escrita neste registo desencadeia a serialização dos dados e saída para o exterior
- Registo de dados de entrada (DATA IN)
  - Ox3F8
  - Este registo acumula um sequência de bits recebida em série
- Registo de estado
  - 0x3FD
  - Bit 0: a 1 indica que há um dado para ler em DATA IN; assim que o dado é lido passa a 0
  - Bit 5: a 1 indica que está pronto a transmitir; assim que é escrito um valor em DATA OUT passa a 0
  - Bits 1,2,3 – vários tipos de erros

Porta série

- Suponha-se a existência das duas seguintes funções C que são equivalentes às instruções máquina IN e OUT

```
unsigned char in_port(unsigned short endereço);
void out_port(unsigned short endereço,
              unsigned char valor);
```

Para enviar um byte teremos

```
void send_serial( unsigned char b ){
  unsigned char s;
  do {
      s = in_port(0x3fd);
  } while(s & 0x20) == 0;

  out_port( 0x3f8, b);
}
```

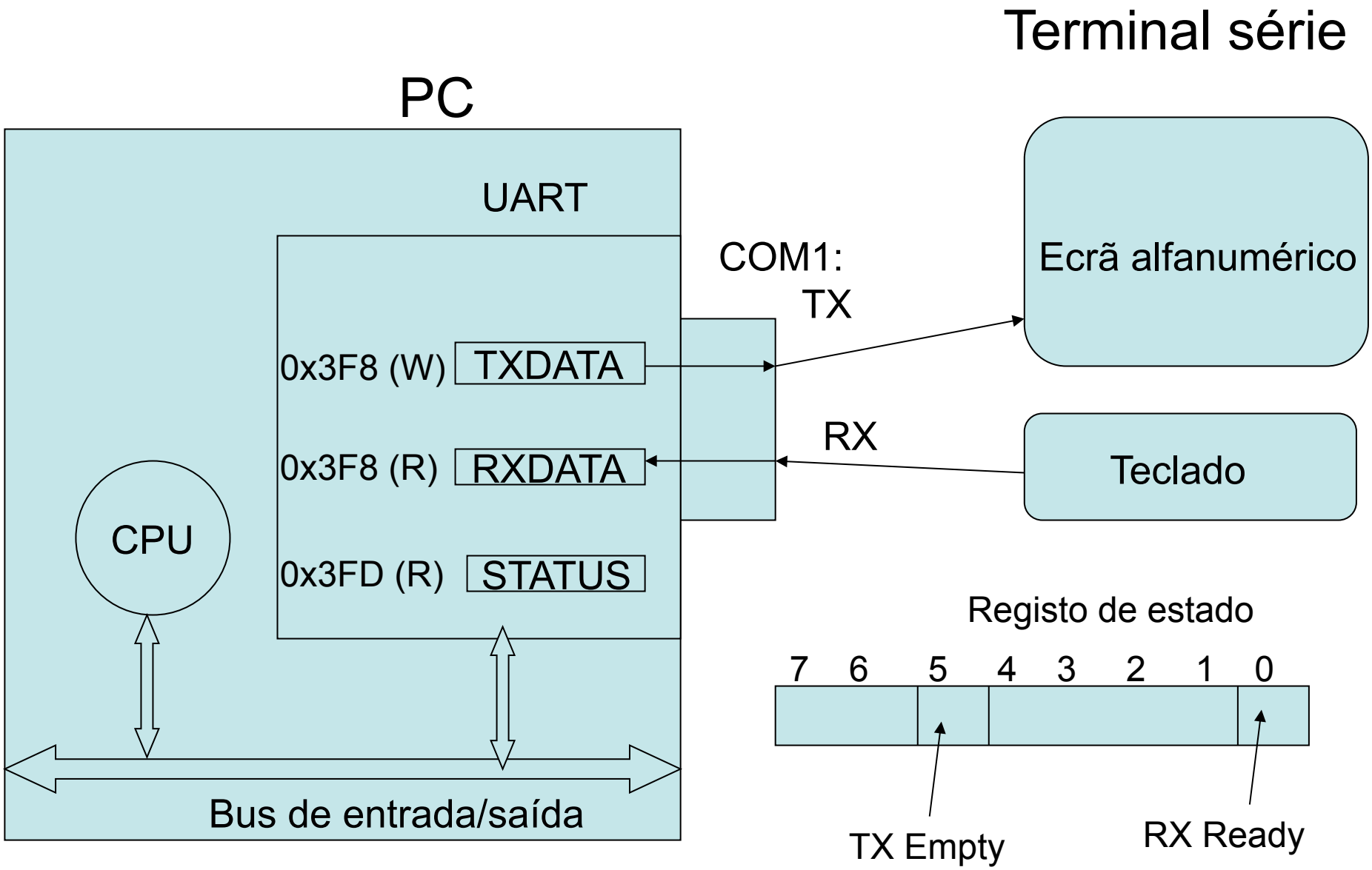
Porta série

```
unsigned char in_port( unsigned short endereço);
void out_port( unsigned short endereço, unsigned
  char valor);
```

Para receber um byte teremos

```
unsigned char receive_serial( ){
  unsigned char s;
  do {
     s = in_port(0x3fd);
  } while(s & 0x01) == 0;
  return in_port(0x3f8);
}
```

Porta série do PC



Device Driver

- ***Device Driver***: conjunto de subrotinas para efectuar operações de entrada/saída, escondendo os detalhes de controlo da interface de um periférico
- Escondem detalhes do *hardware*
  - Ex: vários fabricantes de placas de som, mas muitas das funcionalidades são comuns a todas e podem ser "vistas" da mesma maneira
- Estas rotinas fazem parte do SO
  - Só o SO é que gere o acesso aos periféricos
  - Oferece abstracções e funcionalidades de mais alto nível para os programas

Níveis no I/O

- Exemplo para a porta série:
  - O *device driver* controla a interface da porta
  - Oferece ao SO operações de leitura e escrita de bytes nas portas série
  - O SO oferece aos programas um canal virtual de comunicação: *stream* de bytes
    - Permite (ou não) que vários programas usem as portas série em cada instante, sem interferirem

Dois modos de operação

- A partilha de recursos exige que um programa incorrecto não prejudique outros programas (ou outros utilizadores)

- O Sistema de Operação não pode perder o controlo dos periféricos
  - Um programa não pode aceder directamente a um periférico sem que o SO confirme as permissões e se o acesso pretendido não é, por exemplo, inválido.

- Como impedir os programas normais de usar as instruções de I/O?

Modo utilizador/supervisor

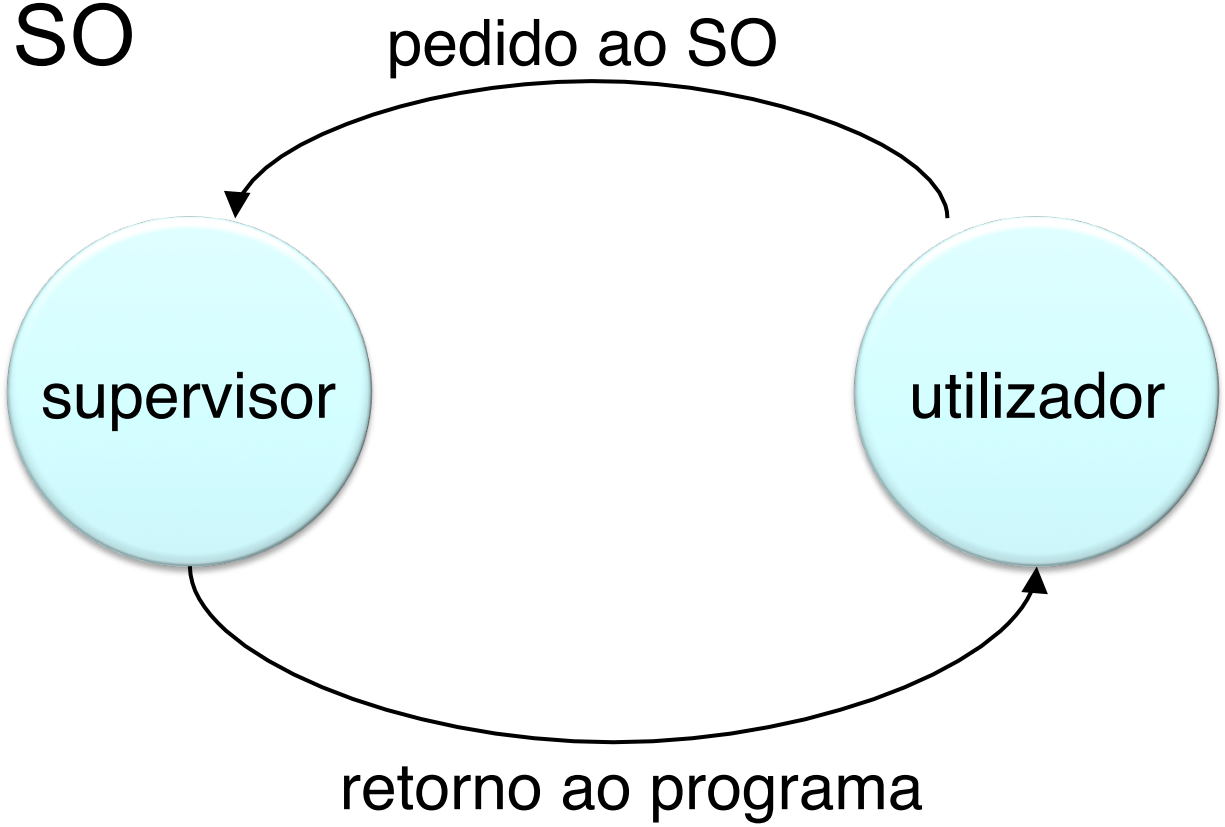
- O CPU tem de suportar pelo menos dois modos de operação:
  1. *Modo utilizador*

     CPU está a executar por conta de um programa normal
  2. *Modo supervisor* (*kernel mode* or *system mode*)

     Execução de código do SO
- No modo utilizador o CPU não executa as instruções consideradas privilegiadas
  - Obtém-se uma excepção e passa a executar código do SO para tratamento do erro
- Para algumas operações os programas têm de efectuar pedidos ao SO
  - o CPU muda então para modo supervisor.

Modo utilizador/supervisor

- Instruções privilegiadas só podem ser executadas em modo supervidor → logo, pelo SO



Protecção do I/O

- É preciso garantir que um programa utilizador nunca execute o seu código em modo supervisor (podia fazer o que quisesse aos periféricos)

- Todas as instruções de I/O são privilegiadas! (IN e OUT nos Intel)

- Existem mais instruções privilegiadas…

- …não porque o SO não use esses dois modos…

Trabalho prático

- Utilização da porta série COM1: em espera activa. Programação da UART.
- Para programar a UART são necessárias as instruções IN e OUT, que são privilegiadas – só acessíveis em modo supervisor dentro do código do SO
- A solução é utilizar um emulador de PC - QEMU - ver http://www.qemu.org
  - Sistema hospedeiro: Linux
  - Sistema hóspede: FreeDOS (“clone” do DOS). Ver http://www.freedos.org

Qemu, FreeDOS e Linux