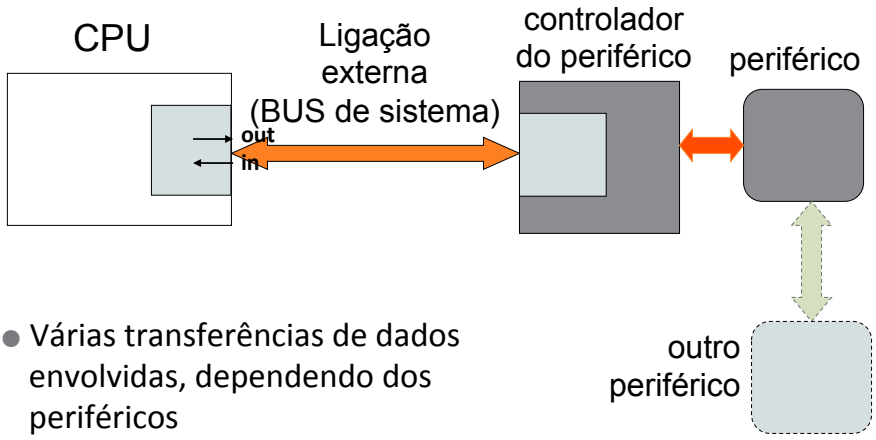
Arquitetura de Computadores

MiEI – 2016/17

DI-FCT/UNL

Aula 13

Transferências nas Entradas/Saídas



- Várias transferências de dados envolvidas, dependendo dos periféricos

Visão do CPU em relação às entradas/saídas

- O CPU não acede diretamente aos periféricos
- Usa uma interface programável para passar pedidos ao controlador de interface e para receber as repostas a esses pedidos
- Os programas só podem controlar o I/O usando o bus de sistema para "falar" com o controlador
- Este é que interage diretamente com o dispositivo físico periférico ou com outros dispositivos via interfaces / bus próprios e especializados

Transferência em série vs. paralelo

- Interface paralelo
  - Num dado instante de tempo, transferem-se vários bits em **simultâneo**;
  - "Largura" da interface é o nº de bits transferidos em paralelo (8bits, 16, ...);
  - Por exemplo, usam-se vários fios para transferir os dados, em que cada fio transporta em simultâneo um bit
- Interface série
  - 1 bit enviado de cada vez (para enviar um byte tem de ser enviada uma **série** de bits)
  - Só um fio para dados (mais a referência ou massa,...)
  - Em princípio mais lento que a interface paralela mas usa menos fios, contactos e electrónica mais simples
    - **A tecnologia tem permitido continuar a criar ligações série cada vez mais rápidas**

Sentido das transferências

- De um modo geral as transferências são bidireccionais. Por exemplo, num disco pode-se ler e escrever.
- A bidireccionalidade pode ser conseguida de duas formas:
  - Interacção ***full-duplex***: são permitidas transferências nos dois sentidos em simultâneo; conseguido através da duplicação das ligações
  - Interacção ***half-duplex***: em cada momento pode-se transferir apenas num sentido; as entidades nas extremidades têm de negociar a inversão do sentido

Sincronização entre os intervenientes

- O periférico e o CPU não trabalham ao mesmo ritmo, nem partilham um relógio único
- É preciso acomodar diferentes ritmos de transmissão
- Duas formas possíveis
  - O sinal é codificado de forma a que se conseguem identificar as "fronteiras" entre bits
  - Há linhas de controlo extra que transportam um sinal de relógio ou marcam a existência de novos dados

Latência e débito binário

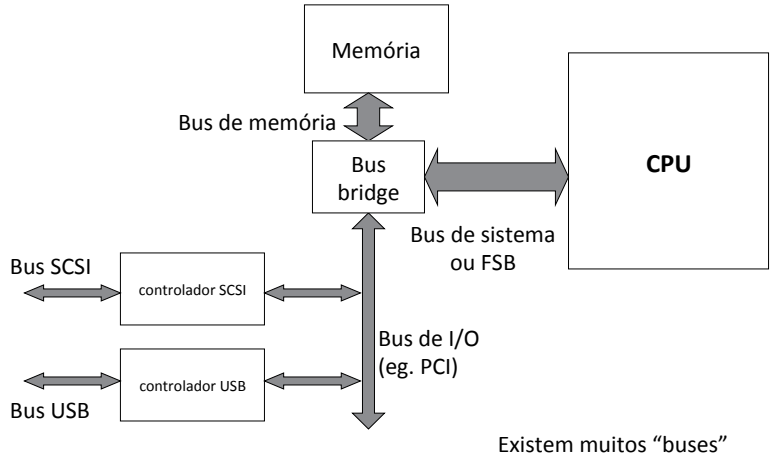
- **Latência** (*latency*): é o tempo que demora a transferir um só bit; ou o tempo que um dispositivo demora a detetar esse bit – medida em unidades de tempo (p.ex. ns)
- **Débito** (*throughput*): número de bits que se conseguem transmitir por unidade de tempo; velocidade de um fluxo de bits – medido p.ex. em Mbit/s (Mbps) ou Mbytes/s (MBps)

Compromissos custo/velocidade

Para aumentar o débito binário, dever-se-iam usar interfaces paralelas com um grande número de linhas e full-duplex

- Há muitas considerações contra esta solução: o número de contactos nos circuitos integrados, interferências entre os sinais, o custo das ligações e dos controladores, etc.
- Uma solução: os dados são divididos em blocos que são transmitidos de forma independente. Permite transferir uma quantidade arbitrária de dados sobre uma ligação com um número limitado de linhas.
  - 1 sector de 512 bytes pode ser transferido em blocos consecutivos de 4 bytes, ou byte a byte, ou mesmo bit a bit (em série)

Arquitecturas de computadores/Buses



Existem muitos "buses"
Diferentes larguras e velocidades

Partilha do "bus"

- "Buses" ligam em simultâneo múltiplas entidades
  - CPUs e periféricos, múltiplos CPUs,...
  - Cada bus é assim partilhado
- A norma do "bus" tem de incluir um protocolo de acesso:
  - Determina qual a entidade que pode usar o "bus" num dado momento
  - Todos os dispositivos ligados ao "bus" têm de obedecer a este protocolo

Características físicas do "bus"

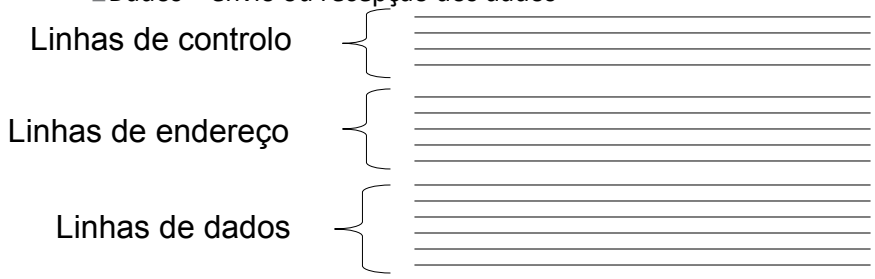
- Transferência de dados em paralelo
  - Pode transferir múltiplos bits em simultâneo
  - Largura típica do bus de sistema: 32 ou 64 bits
- O "bus" é normalmente passivo
  - Não contém muita lógica própria; pode ser visto como um mero conjunto de fios (linhas)
  - São os dispositivos a ele ligados que suportam a comunicação
  - Pode conter um árbitro: suporta a partilha, decidindo quem é tem acesso

Suporte do "bus"

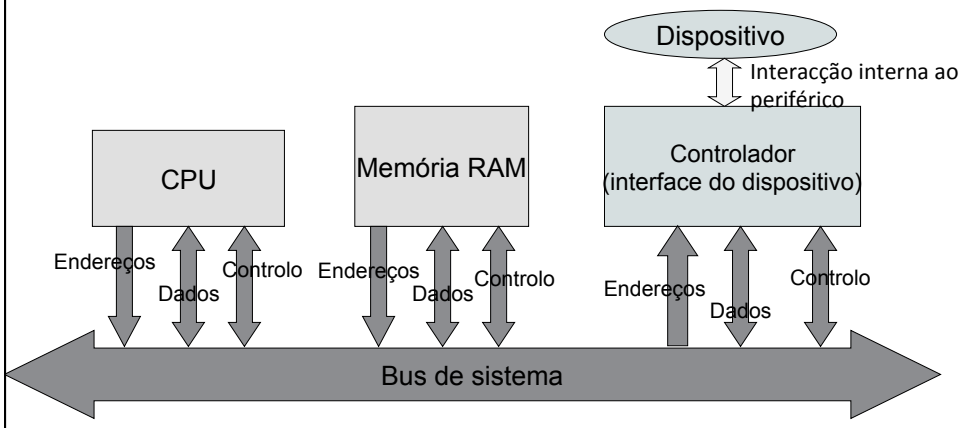
- Fios numa placa (*board*) de circuito impresso ou num cabo
- Fichas (*sockets*) na placa ou para o exterior do computador, ou nos extremos do cabo
  - permitem inserir e remover dispositivos facilmente
- Normas definem:
  - Características físicas anteriores
  - As especificações eléctricas exatas
  - As temporizações dos sinais
  - A forma como os dados são codificados

Organização do "bus"

- Bus está normalmente separado funcionalmente em 3 partes:
  - Controlo – controlar a partilha do bus e para indicar o pedido
  - Especificação do endereço (origem/destino dos dados)
  - Dados – envio ou recepção dos dados



Controlador de dispositivo periférico

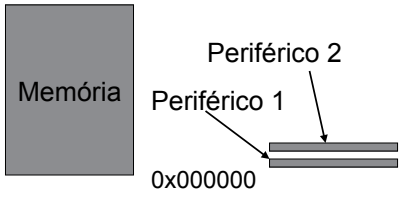


- Como distinguir endereços I/O de endereços de Memória?

Espaço de endereços separado

- Dois espaços de endereçamento separados
  - Um para a memória
  - Um para os controladores dos dispositivos, os **portos** ou **portas** (*ports*)
- Cada controlador é configurado para responder a uma faixa de endereços diferente
- A faixa de endereços da memória é sempre disjunta da dos periféricos
- O CPU necessita de **instruções diferentes** para separar os acessos a memória dos acessos aos periféricos

Entidade	Faixa de endereços
Memória	0x000000-0xFFFFFF
Periférico 1	0x000000-0x00000B
Periférico 2	0x00000C-0x000017



Instruções dedicadas ao IO

- Espaço separado
  - Há instruções máquina dedicadas ao acesso do espaço de endereços de entrada/saída (chamados *ports*)
  - Exemplo da família Intel x86/Pentium
    - IN *port, regA* (leitura do *port* para o registo EAX/AX/AL)
    - OUT *regA, port* (escrita do EAX/AX/AL para o *port*)

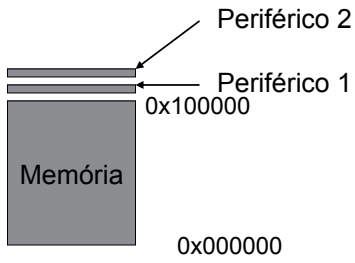
      (Se port > 255 então: IN %DX, *regA* e OUT *regA*, %DX)
- Exemplo (envio de 1 byte—OUT):

```
mov $5, %al
mov $0x3f8, %dx
out %al, %dx
```

Espaço de endereços único (**memory-mapped I/O**)

- Uma faixa de endereços corresponde à memória
- Cada controlador é configurado para responder a uma faixa de endereços fora do espaço da memória e a que nenhum outro controlador responde
- As faixas de endereços da memória e dos controladores são disjuntos

Entidade	Faixa de endereços
Memória	0x000000-0x0FFFFF
Periférico 1	0x100000-0x10000B
Periférico 2	0x10000C-0x100017



Espaço separado vs. Espaço único

- Espaço único
  - Não há instruções máquina especiais para acesso aos controladores. São usadas *LOAD/STORE* (MOV) para endereços que pertencem à faixa de endereços de entrada/saída
- Exemplo (envio de 1 byte—como um OUT):

```
movb $5, 0x10000C
```

Uso do “bus”

- O “bus” só suporta duas operações (via controlo)
  - Leitura
  - Escrita
- No controlo assinala-se se é acesso a memória ou IO
- Todas as operações de I/O são efectuadas efectuando leituras e escritas via “bus” de sistema
- Estas traduzem-se na transferência dos dados de/para as interfaces dos periféricos, como se fossem leituras e escritas em **registos internos** ao controlador do periférico

Acesso em leitura (Load ou Input)

- Usa as linhas de controlo para obter acesso ao “bus”
- Coloca um endereço nas linhas de endereço
- Usa as linhas de controlo para pedir uma operação de leitura, a memória ou IO
- Testa as linhas de controlo para verificar se a transferência já se concluiu
- Lê um valor das linhas de dados (resposta ao pedido)
- Posiciona as linhas de controlo para deixar outra entidade aceder ao “bus”

Acesso em escrita (Store ou Output)

- Usa as linhas de controlo para obter acesso ao "bus"
- Coloca um endereço nas linhas de endereço
- Coloca um valor nas linhas de dados
- Usa as linhas de controlo para pedir uma operação de escrita, e indicar se memória ou IO
- Testa as linhas de controlo para verificar se a transferência já se concluiu
- Posiciona as linhas de controlo para deixar outra entidade aceder ao "bus"

Ponto de vista do CPU

- A interface do dispositivo fornece uma interface de programação, que tem apenas duas operações:
  - Memória: ler e escrever
  - Periféricos: entrada e saída
- Nas entradas/saídas com periféricos, cada periférico possui um controlador com uma faixa de endereços **distinta**
- Os valores escritos em cada *port* podem ter significados especiais para o controlador
  - Definem uma linguagem de operação/controlo do periférico
- **Para o programador, o bus é invisível**

Ponto de vista da interface do dispositivo

Seja R o conjunto de endereços associado à interface

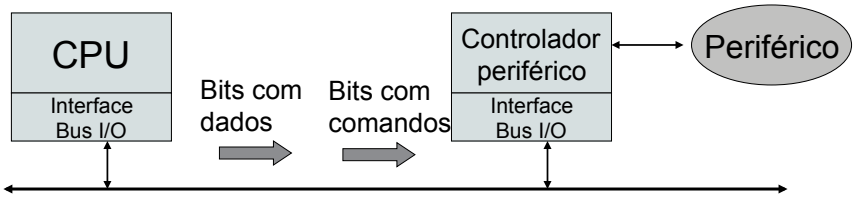
```
while(1){
  monitorizar o bus até aparecer um pedido
  if( pedido refere um endereço pertencente a R )
      aceitar o pedido
      eventualmente responder
  else
      ignorar o pedido
}
```


Erros

- Podem ocorrer dois tipos de erros:
  - Conflito de endereços: mais de um periférico tenta responder
  - Endereços não atribuídos: nenhum responde (*timeout*)

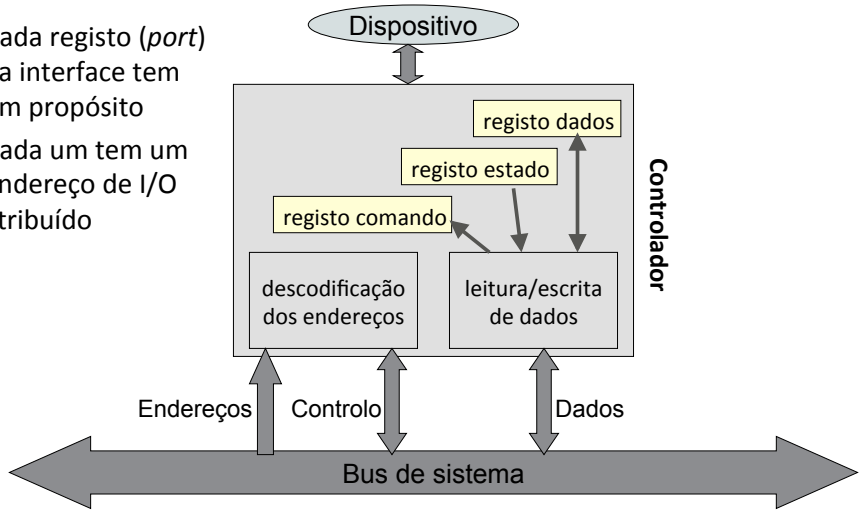
Ler/escrever e Entrada/Saída

- O bus apenas fornece um meio de passar bits de uma entidade para outra
- Os bits que passam no bus de dados podem ser:
  - Dados que se pretende trocar com o periférico
  - Representar operações de controlo sobre o periférico; cada configuração de bits é interpretada pelo controlador do periférico
- I/O: Depois de dado o comando o CPU e o controlador/periférico executam autonomamente



Interface genérica de periférico

- Cada registo (*port*) da interface tem um propósito
- Cada um tem um endereço de I/O atribuído



Controlador genérico

- Em geral, as interfaces têm três tipos de registos (portos ou portas):
  1. Porta de dados: para guardar os dados enviados pelo CPU ou recebidos do exterior pelo dispositivo periférico
  2. Porta de estado: indica informação sobre o estado corrente do periférico
     - existência de dados para ler
     - disponível para receber dados
     - erros e outras informações
  3. Porta de controlo: para receber comandos vindos do CPU (inicializar interface, posicionar a interface em estados para receber ou enviar, e outros comandos que desencadeiam a operação do periférico)

Exemplo de uma operação

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 para reg. DADOS (OUT)
     - o CPU escreve o comando para enviar no reg. 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 fica a transferir dados de/para o periférico

Programação dos controladores de I/O

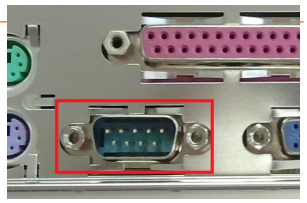
- Cada controlador oferece uma interface de operação:
  - Uma "linguagem" específica desse *hardware*
- Controladores podem ser mais ou menos autónomos em relação ao CPU
  - Controladores com pouca "inteligência" precisam de grande interacção com o CPU
  - Controladores com grande "inteligência" oferecem funcionalidades mais potentes. Exemplo: só necessitam de atenção do CPU no início e fim da transferência
- Sincronização entre o CPU e os controladores
  - Periféricos são muito mais lentos do que o CPU
  - O CPU tem de interactuar com o periférico para saber se este já está disponível para receber novo comando, ou se já tem o dado pretendido ...

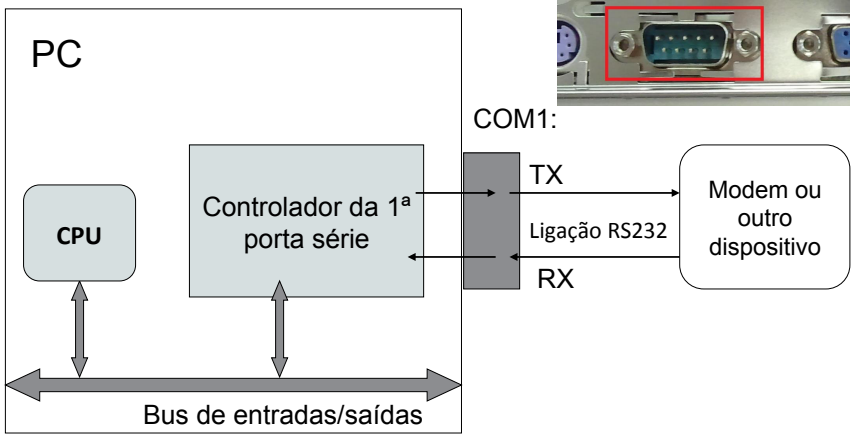
Programação por espera ativa (*busy waiting* ou *polling*)

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 executar outro código, voltando a testar mais tarde

- O controlador fica a efectuar a operação
- O controlador posiciona os seus bits de estado (reg. ESTADO)
  - O controlador de I/O não informa o CPU diretamente e não interrompe o CPU

Exemplo: Porta série do PC

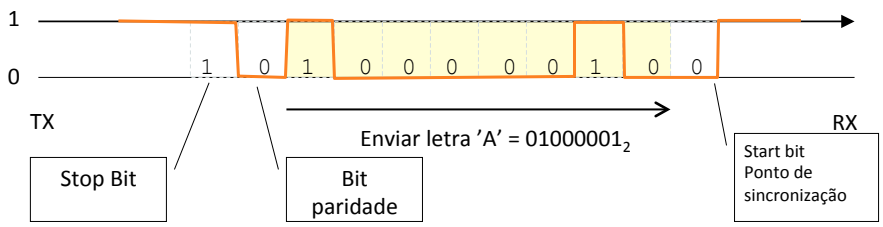




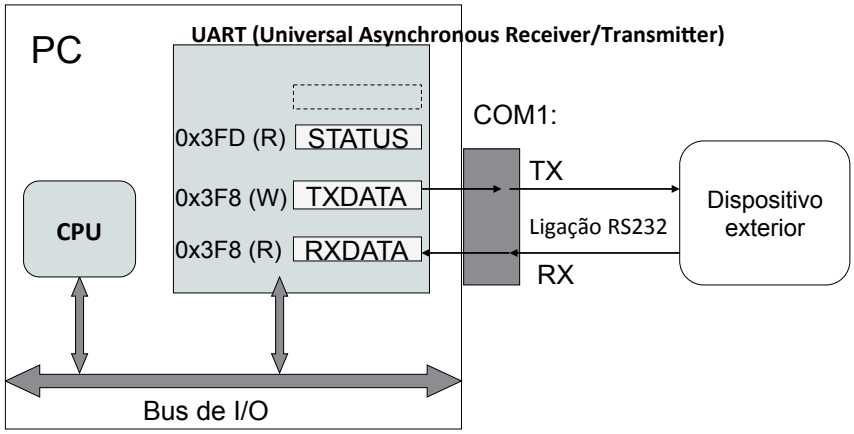
RS-232 ou V.24

- A norma inclui aspectos "físicos": fichas, cabos, tensões (Volts), etc…
- Características da comunicação:
  - Taxa de transferência, sincronização , detecção de erros, etc…
  - Exemplo simples:

  Grupo de 8 bits, 1 start bit, 1 stop bit e 1 bit paridade par

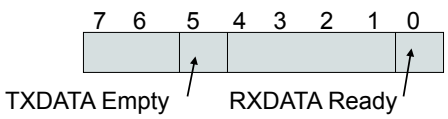


Interface do controlador



Interface da UART

- A interface de programação oferecida esconde os detalhes da comunicação série
  - Interacção com o controlador por bytes de dados
  - Suporta *full-duplex*: 1 reg dados a enviar (TXDATA) e 1 reg. de dados recebidos (RXDATA)
  - Registo de estado (STATUS)



  - Vários registos de comando/controlo para programar as características da comunicação

Porta série: envio

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

```
unsigned char in(unsigned short port);
void out(unsigned int port,unsigned char byte);
```

- Para enviar um byte pela porta série teremos:

```
void send_serial( unsigned char b )
{ unsigned char s;
  do {
      s = in(0x3fd); /* polling STATUS*/
  } while ((s & 0x20) == 0);

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