Arquitetura de Computadores

MIEI – 2021/22

DI-FCT/UNL

Programação de Dispositivos de Entradas/Saídas (I/O)

Sumário

- Dispositivos de Entradas/Saídas
- Bus
- Endereçamento de Controladores de Dispositivos de E/S
- Organização do Software de E/S
- Programação de de Controladores de Dispositivos de E/S
  - Espera Ativa
  - Interrupções
- Periféricos orientados ao bloco
  - Discos Magnéticos e SSD
  - Acesso Direto à Memória (DMA)

Bibliografia:

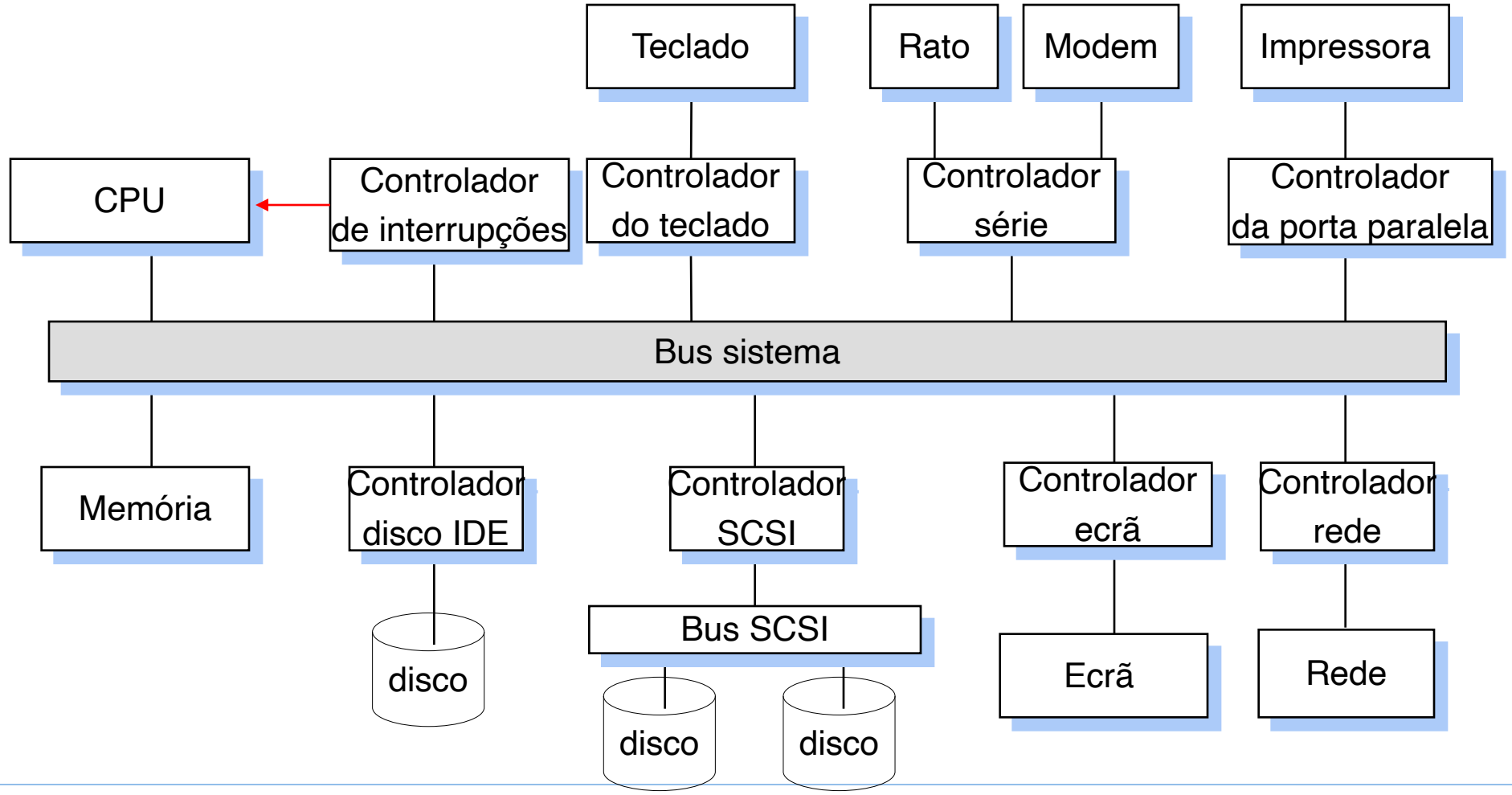
Remzi H. Arpaci-Dusseau and Andrea C. Arpaci-Dusseau. Operating Systems: Three Easy Pieces. Cap 36 *(https://pages.cs.wisc.edu/~remzi/OSTEP/file-devices.pdf).*

Bryant, O'Hallaron Computer Systems: A Programmer's Perspective, 2nd ou 3rd Ed, secções 6.1.2, 6.1.3 e 6.1.4

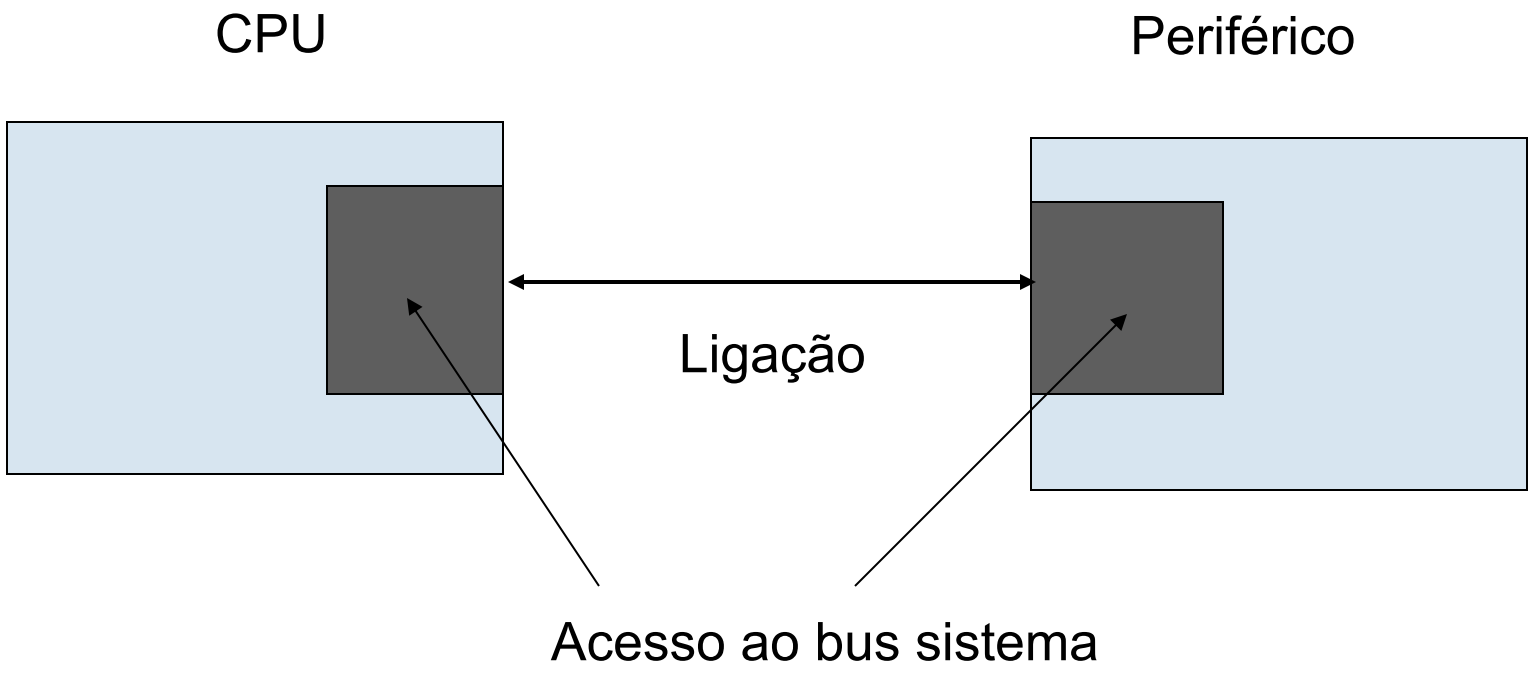
Dispositivos de entrada/saída

- **Dispositivos de comunicação**:
  - **Dispositivos de entrada**: permitem introduzir no sistema dados gerados por um agente exterior que são transformados em formato binário de forma a poderem ser processados;
    - teclado, rato, controlador de rede (entrada)
  - **Dispositivos de saída**: recebem dados do CPU/memória em formato digital e tornam-nos acessíveis a um agente exterior
    - ecrã, impressora, controlador de rede (saída)
- **Dispositivos de armazenamento**: mantêm internamente ao sistema dados em forma não volátil. Podem ser de entrada/saída ou só de entrada
  - Discos magnéticos, discos óticos, discos de estado sólido, “tapes”

Hardware


Teclado
Rato
Modem
Impressora
CPU
Controlador de interrupções
Controlador do teclado
Controlador série
Controlador da porta paralela
Bus sistema
Memória
Controlador disco IDE
Controlador SCSI
Controlador ecrã
Controlador rede
disco
Bus SCSI
Ecrã
Rede
disco
disco

Interligação CPU-Periférico



Para transferir dados entre o CPU e o periférico

Ligação em série ou em paralelo

- Interface **paralelo**
  - Composta por muitos fios
  - Num dado instante de tempo, cada fio transporta um só bit
  - “Largura” da interface é o nº de fios (8bits, 16, ...)

- Interface **série**
  - Só um fio (mais a referência ou massa)
  - 1 bit enviado de cada vez
  - Mais lento que a interface paralela
  - A vantagem é que usa menos fios

Visão do CPU em relação às entradas/saídas

- O CPU não faz acesso aos periféricos diretamente
- Usa uma interface programável para passar pedidos ao controlador de interface e para receber as repostas a esses pedidos
- O controlador de interface converte esses pedidos em sinais elétricos
- Esses sinais são interpretados pelo controlador do lado do periférico
- Este é que interage diretamente com o dispositivo físico.

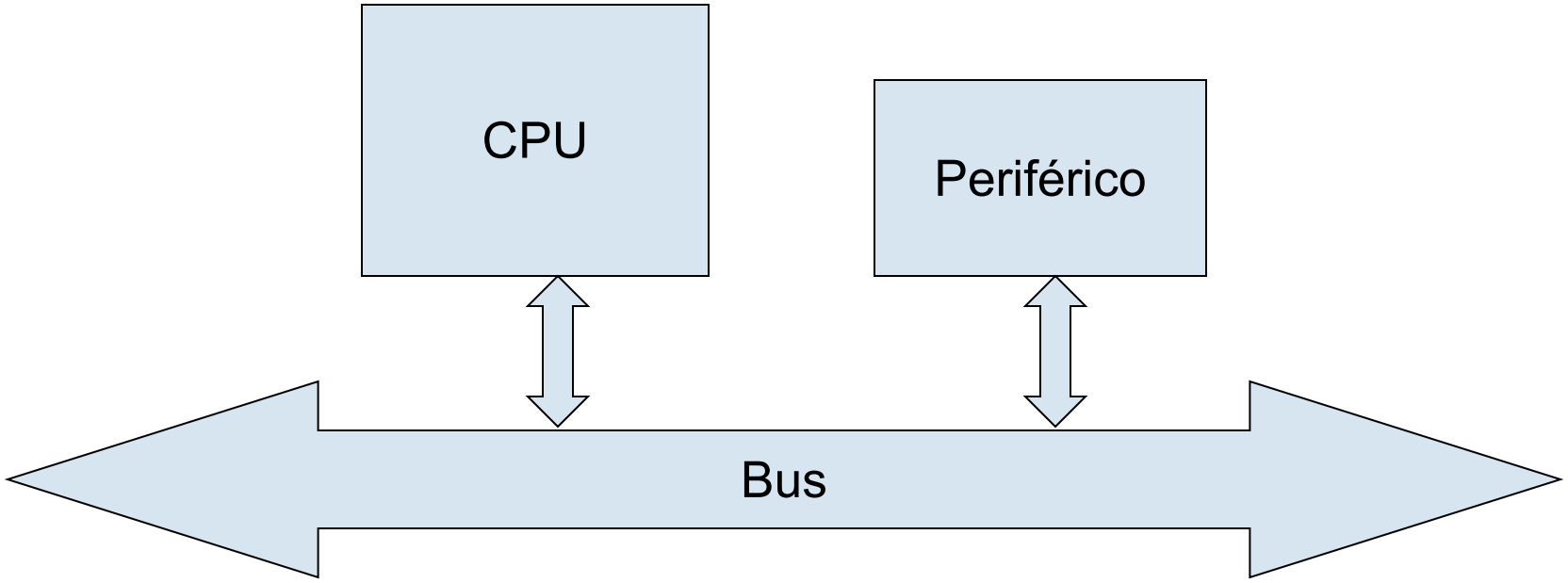
Bus (barramento)

- Um “bus” é um mecanismo de comunicação digital que permite a duas ou mais unidades funcionais transferir dados e sinais de controle

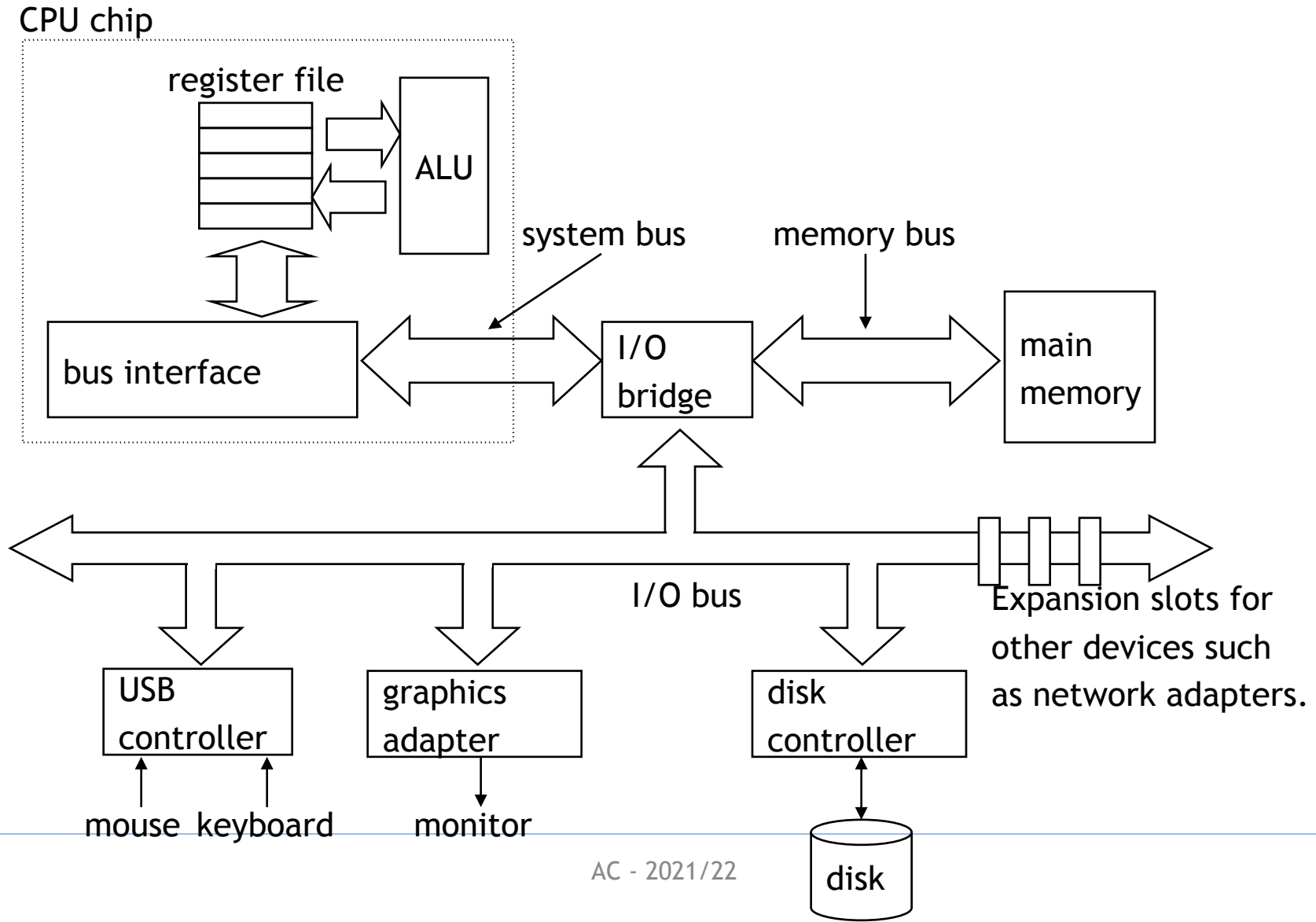
- Um computador pode conter vários “buses” que estão otimizados para um determinado ojectivo
  - “bus” de **memória**: liga o CPU ao subsistema de memória
  - “bus” de **entrada/saída** (I/O bus) interliga o CPU a um conjunto de dispositivos de entrada/saída

Bus (barramento)

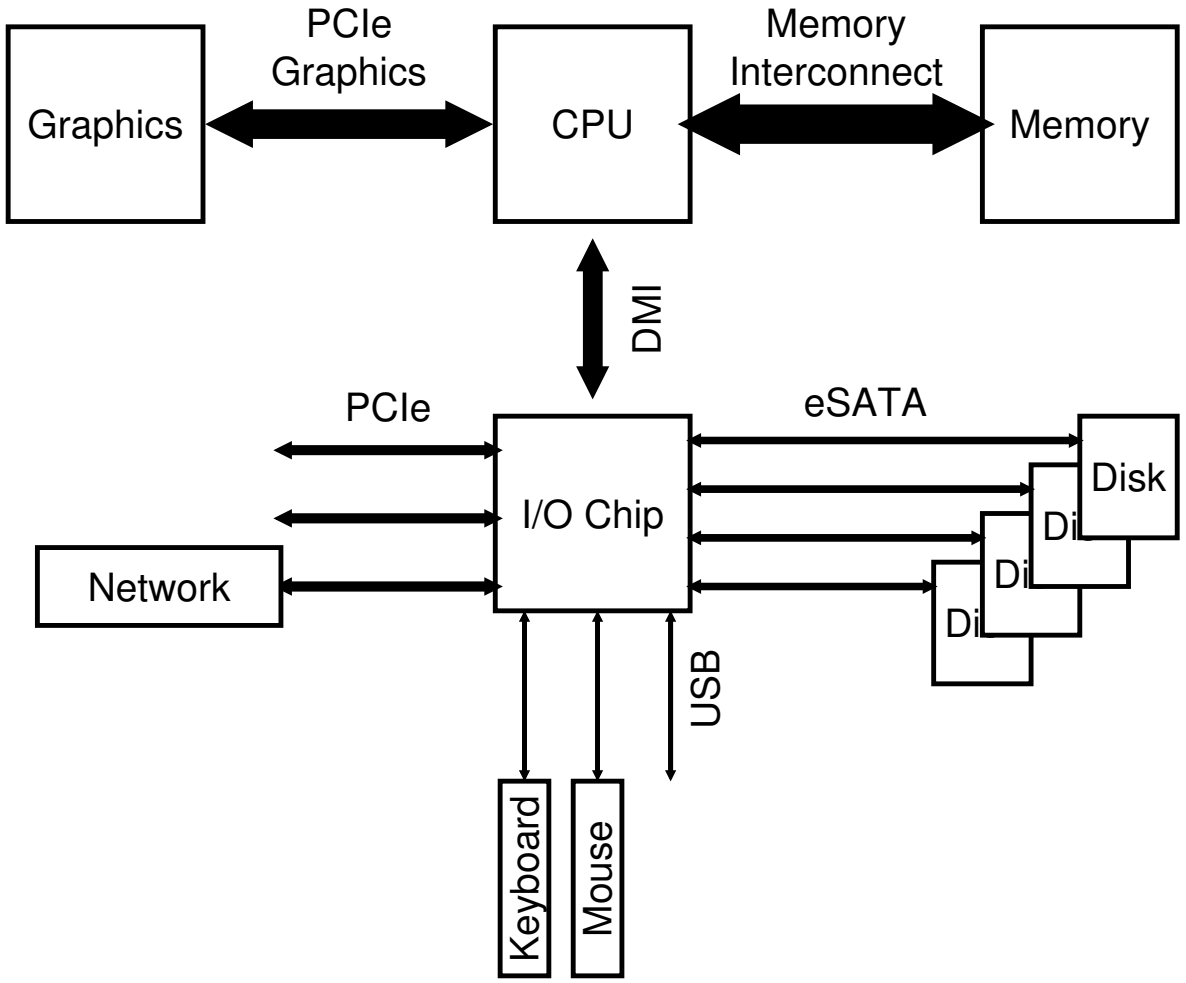
- Um “bus” suporta as ligações entre o CPU e os periféricos referidas anteriormente. Na maior parte dos casos transfere múltiplos bits em simultâneo (paralelo)



Hardware típico



Máquinas Atuais



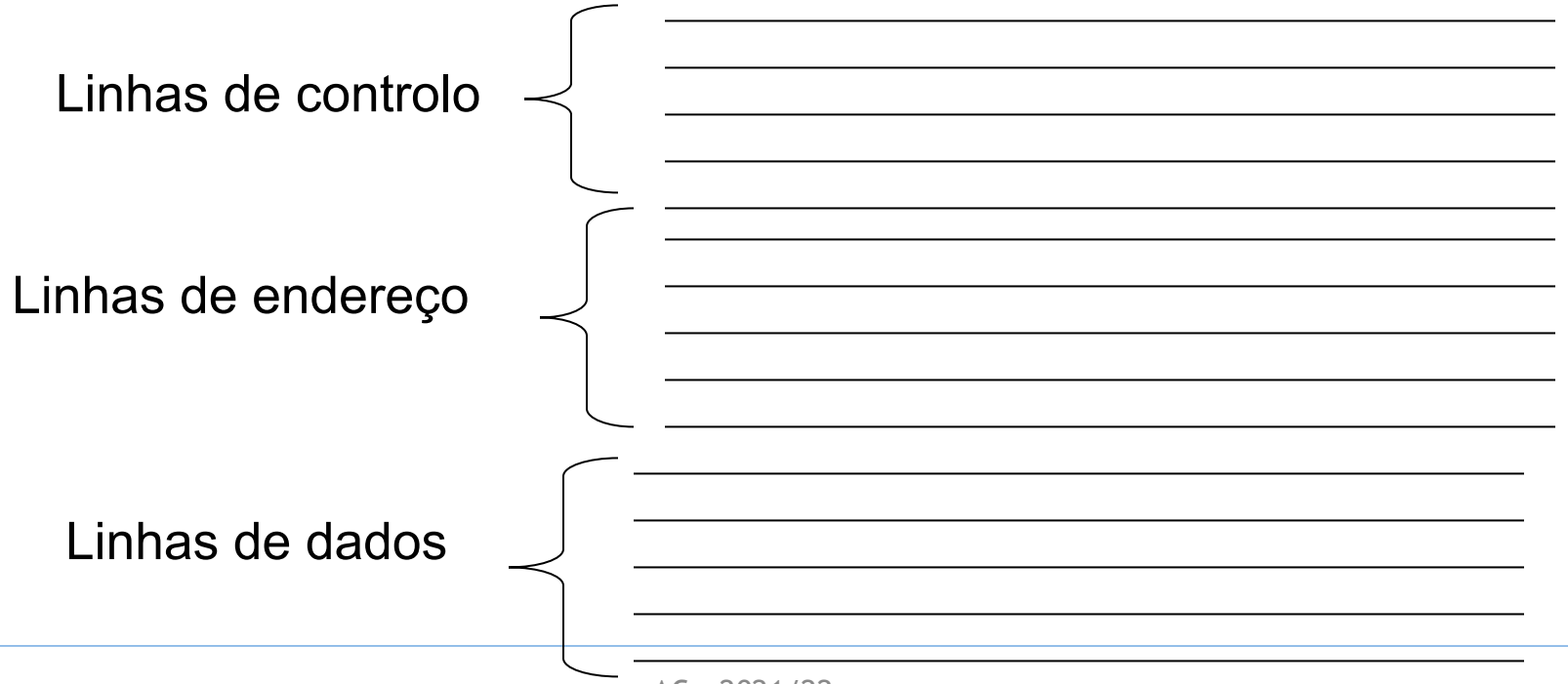
Características do “bus”

- Transferência de dados em paralelo
  - Pode transferir múltiplos bits em simultâneo
  - Largura típica: 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

Organização do “bus”

- Bus está separado funcionalmente em 3 partes
  - Controlo
  - Especificação da localização do endereço
  - Dados a transferir



Acesso ao “bus”

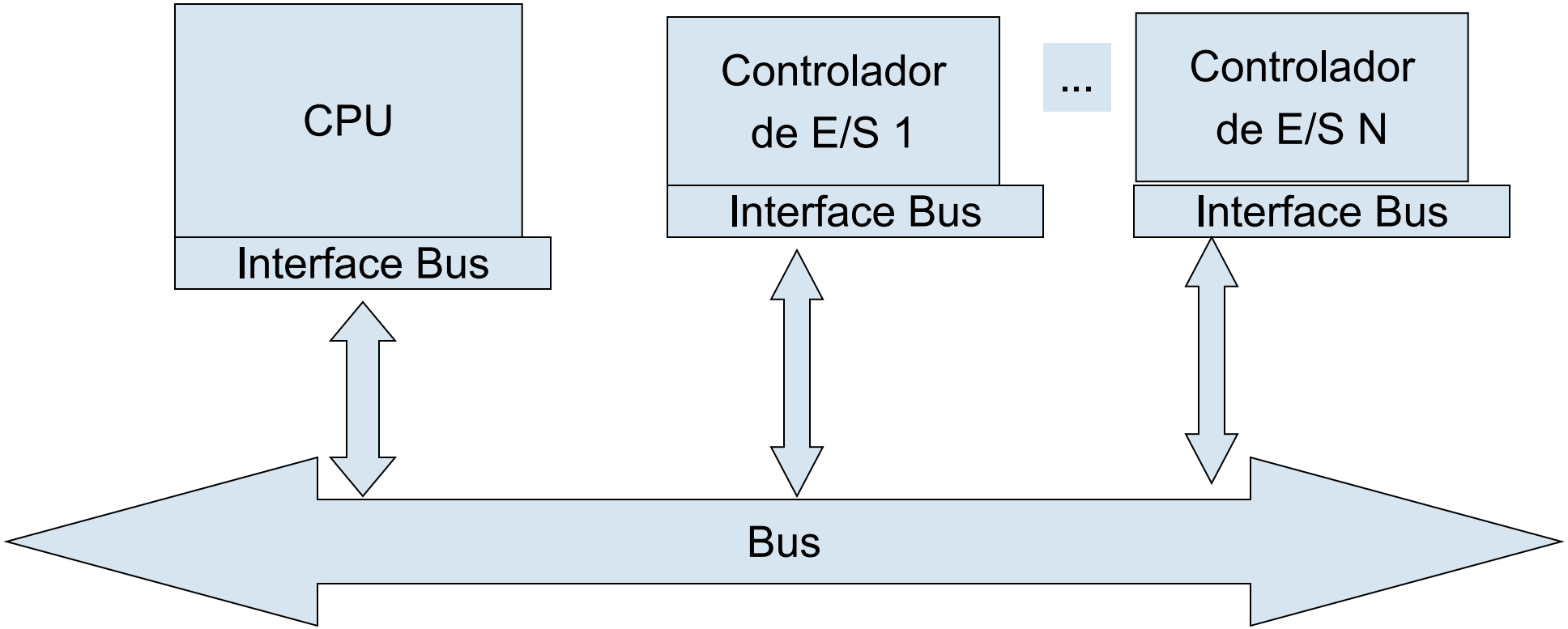
- O “bus” só suporta duas operações
  - Leitura
  - Escrita

- O modo de acesso é o mesmo da memória

- Todas as operações de E/S são realizadas efetuando leituras e escritas no “bus”

Endereçamento no “bus”

- Um “bus” define um espaço de endereçamento



Endereçamento no “bus”

- A interface de bus implementa o protocolo do “bus” e trata de todas as transferências
- Usa as linhas de controlo para fazer acesso ao bus
- Posiciona as linhas de dados e endereços
- Embora receba todos os pedidos que passam no “bus”
  - A interface só participa nas transferências que contêm os endereços para os quais a interface foi configurada

Ponto de vista do CPU

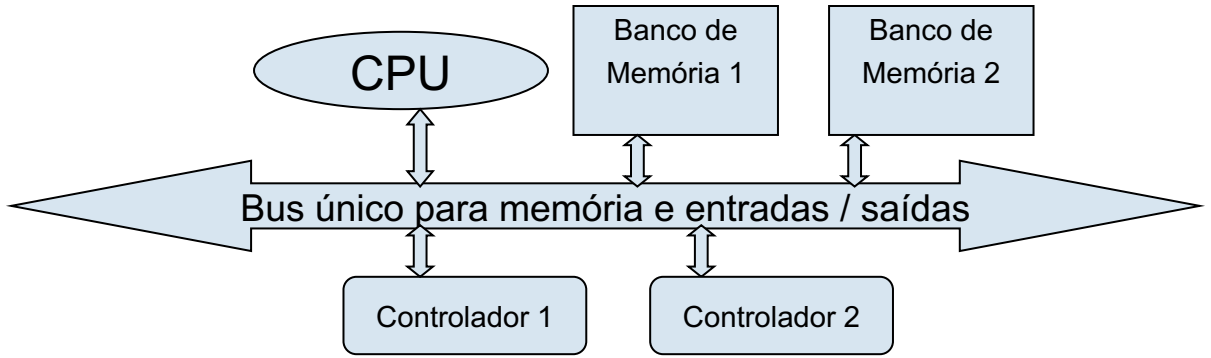
- A interface do bus fornece uma interface de programação, que tem apenas duas operações:
  - Ler e escrever
- Quando há uma referência à memória o CPU deixa tudo a cargo da interface do bus:
  - Leitura – o CPU pede à interface para efctuar a leitura
  - Escrita – idem
- Do ponto de vista do programador, a interface de bus é invisível: ela define um **espaço de endereçamento, em que cada controlador corresponde a uma faixa de endereços distinta**

A Interação com os Controladores de E/S

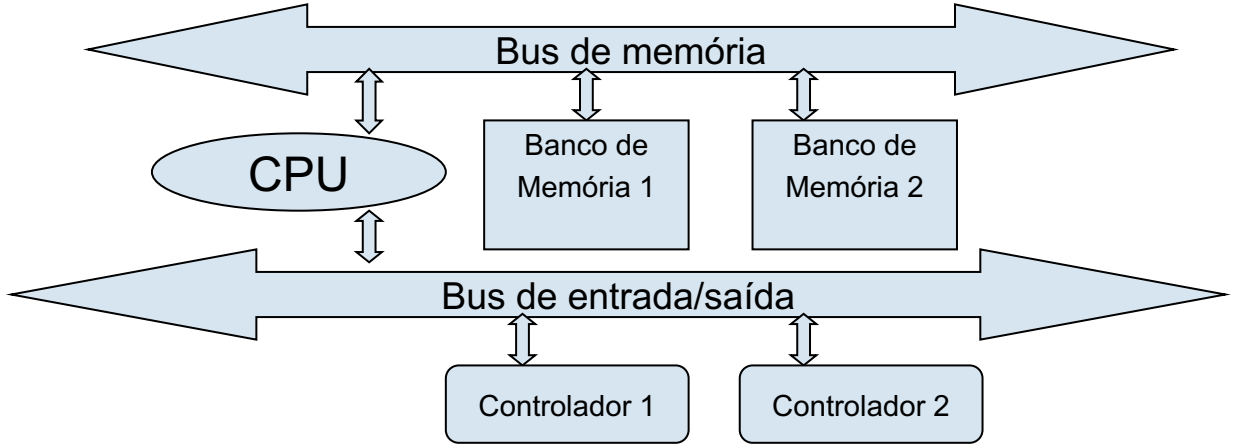
- Como endereçar os vários controladores
- Controlador genérico
- Interação usando espera ativa
- Interação usando interrupções

Buses de memória e E/S unificados ou separados

Bus único



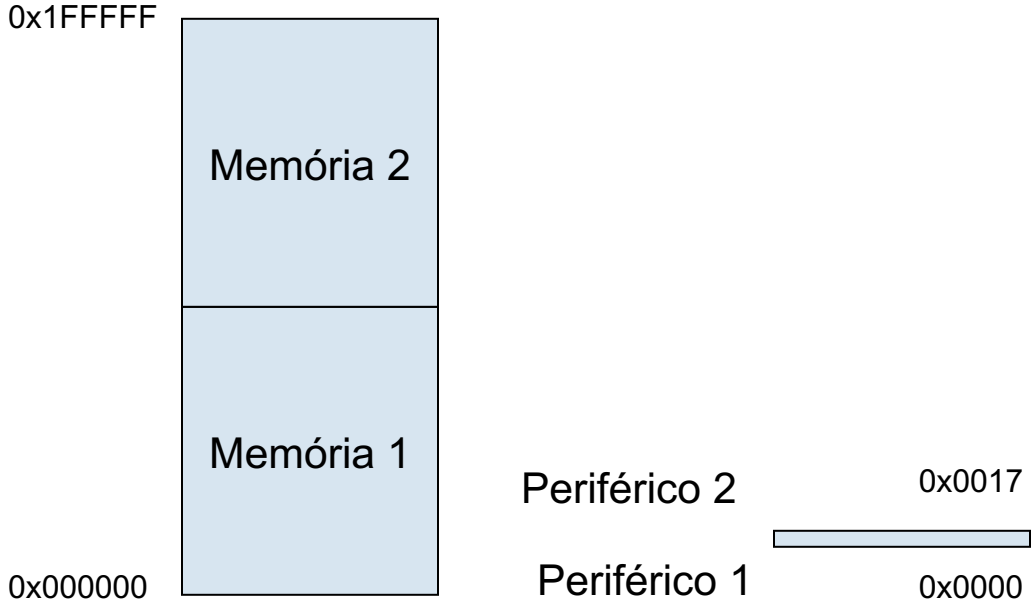
Bus separado



“Buses” separados

- Dois espaços de endereçamento separados
  - Um para a memória
  - Um para os controladores dos dispositivos
- Cada banco de memória é configurado para uma faixa de endereços que não tem intersecção com a faixa de endereços de nenhum outro banco
- Cada controlador é configurado para responder a uma faixa de endereços a que nenhum outro controlador responde

Entidade	Faixa de endereços
Memória 1	0x000000-0x0FFFFF
Memória 2	0x100000-0x1FFFFF
Periférico 1	0x0000-0x000B
Periférico 2	0x000C-0x0017



Acesso em leitura

Instrução máquina do CPU

```
in endereço, registo do CPU
```

- Transfere o conteúdo de um endereço de E/S (**porta de E/S**) para um registo do CPU
  - Usa as linhas de controlo para obter acesso ao “bus”
  - Coloca o endereço **endereço** nas linhas de endereço
  - Usa as linhas de controlo para requerer uma operação de leitura
  - Testa as linhas de controlo para verificar se a transferência já se concluiu
  - Lê um valor das linhas de dados
  - Transfere para o registo registo do CPU

Acesso em escrita

Instrução máquina do CPU

```
out registo do CPU, endereço
```

- Transfere o conteúdo de um registo do CPU para um registo de um controlador de E/S (**porta de E/S**)
  - Usa as linhas de controlo para obter acesso ao “bus”
  - Coloca o endereço `endereço` nas linhas de endereço
  - Coloca o conteúdo de registo do CPU nas linhas de dados
  - Usa as linhas de controlo para indicar uma operação de escrita
  - Testa as linhas de controlo para verificar se a transferência já se concluiu

Programação dos controladores de dispositivos de E/S

- Controladores podem ser mais ou menos autónomos em relação ao CPU
  - Controladores com *pouca* inteligência precisam de grande interação com o CPU
  - Controladores com *grande* inteligência 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 interatuar com o periférico para saber se este já está disponível para receber novo comando, se já tem o dado pretendido ...

Registos de uma interface

- Uma interface ocupa uma série de endereços (normalmente consecutivos) no espaço de endereçamento
  - Registo(s) de dados (leitura): onde se lêem 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 , ...

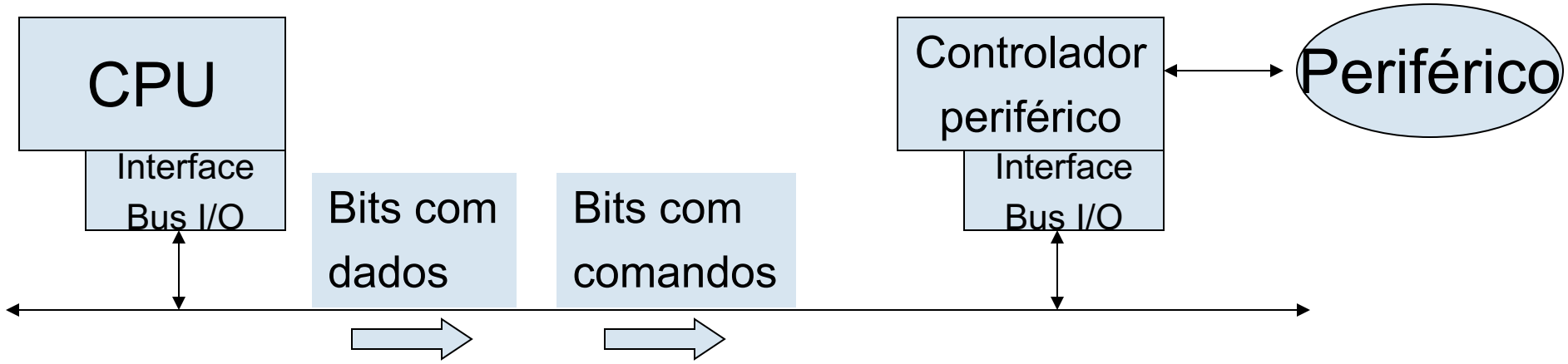
Registos de uma interface

- 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 um rato se move, o movimento relativo em relação à última posição fica guardado num registo de dados
    - Quando o CPU lê esse registo de dados, desencadeia automaticamente um novo processo de medição de deslocamento, cujo resultado virá para o registo de dados.

Como é que operações de leitura e escrita manipulam periféricos ?

- O bus apenas fornece um meio de passar bits de uma entidade para outra
- Os bits que passam no bus podem ser:
  - Dados que estão a ser transferidos
  - Representar operações de controlo sobre o periférico; uma determinada configuração de bits é interpretada pelo interface do hardware



Ponto de vista da interface do dispositivo

- Seja R o conjunto de endereços associado à interface

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

- Podem ocorrer dois tipos de erros (bus errors):
  - Conflito de endereços: mais do que um responde
  - Endereços não atribuído: nenhum responde (timeout)

Exemplo: um “display” de luzes

- Periférico faz as seguintes ações:
  - Ligar/desligar o display
  - Mudar o brilho
  - Luz nº i on e off
- Controlador ocupa endereços de 100 a 103; bus tem 16 bits de dados

Endereço	Operação	Acções associadas
100	Escrita	Dados a zero desligam o display; dados não nulos ligam o display
101	Leitura	Retorna 0 se o display está desligado; retorna um valor não nulo se está ligado
102	Escrita	Muda o brilho. Os 4 bits menos significativos especificam o brilho de 0 (menor) a 15 (maior)
103	Escrita	“Bitmap” das luzes 0 a 15: bit a 1 luz acesa, bit a 0 luz apagada

Organização do software de Entradas/Saídas

Objetivos da gestão de dispositivos de entrada/saída

- Apresentar uma visão mais abstrata dos dispositivos de comunicação e armazenamento, escondendo aos seus utilizadores (processos) detalhes sobre o funcionamento

- Permitir o uso eficiente dos dois tipos de dispositivos: comunicação e armazenamento

- Suportar a partilha dos dispositivos de comunicação e armazenamento

Partilha dos dispositivos de entrada/saída

- Dispositivos partilháveis
  - Podem ser usados em concorrência por vários processos
  - **Exemplo**: discos

- Dispositivos dedicados
  - Atribuído em exclusivo a um processo por um (longo) período de tempo
  - Acesso aos dispositivos não é só uma questão de proteção, mas também é preciso gerir os acessos de acordo com as características do dispositivo

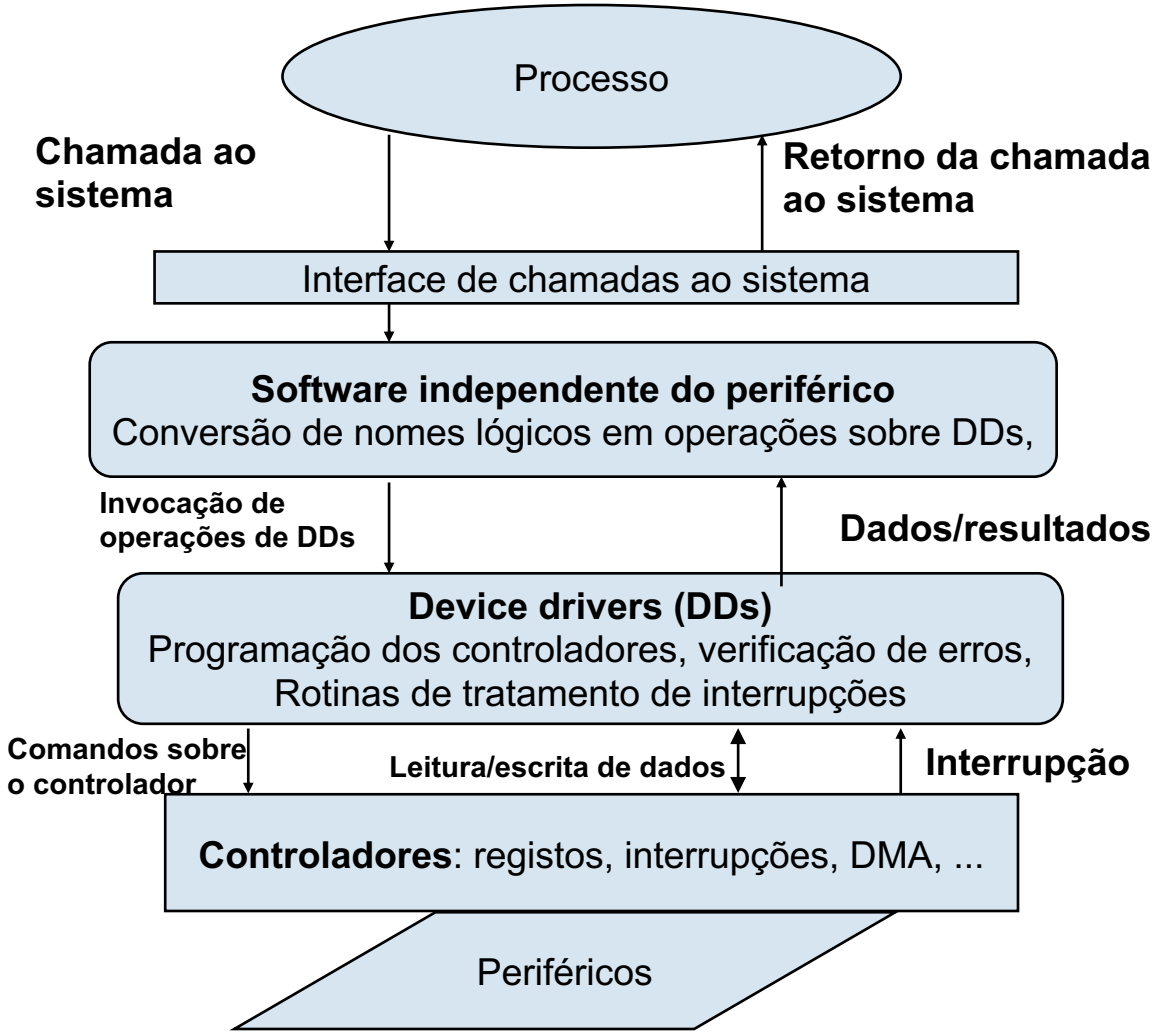
Acesso aos dispositivos de entrada/saída

- Apenas permitido ao sistema operativo
  - Manipulação dos periféricos por instruções privilegiadas
  - Código de acesso direto aos periféricos concentrado nos **device drivers** (supervisores de periféricos que pertencem ao SO)
  - Utilizadores fazem acesso indireto através de chamadas ao sistema

Organização do sistema na parte relacionada com E/S

- Dispositivos de E/S integrados no sistema de ficheiros:
  - Designação (“/dev/ttys0”, “COM1:”, …)
  - Chamadas ao sistema com modelo semelhante ao dos ficheiros (open/read/write/close), com algumas extensões para acomodar certas classes de periféricos

Organização do software de E/S



Programação de Controladores de Dispositivos de Entradas/Saídas

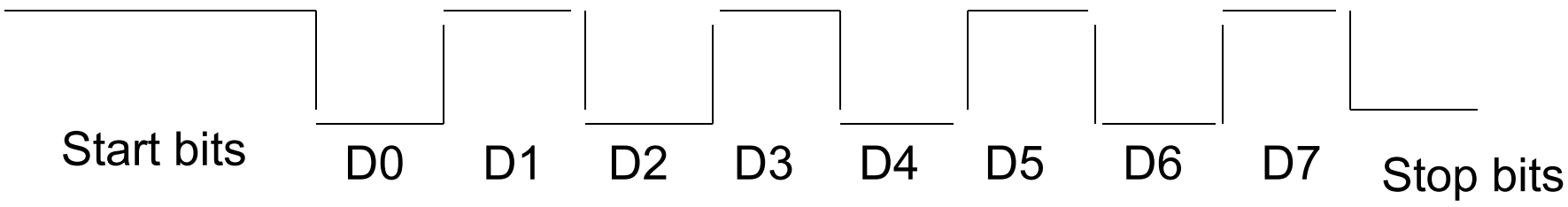
Programação por **Espera Activa** (Polling)

- O CPU interroga repetidamente o controlador do periférico para saber se a operação anteriormente especificada já terminou

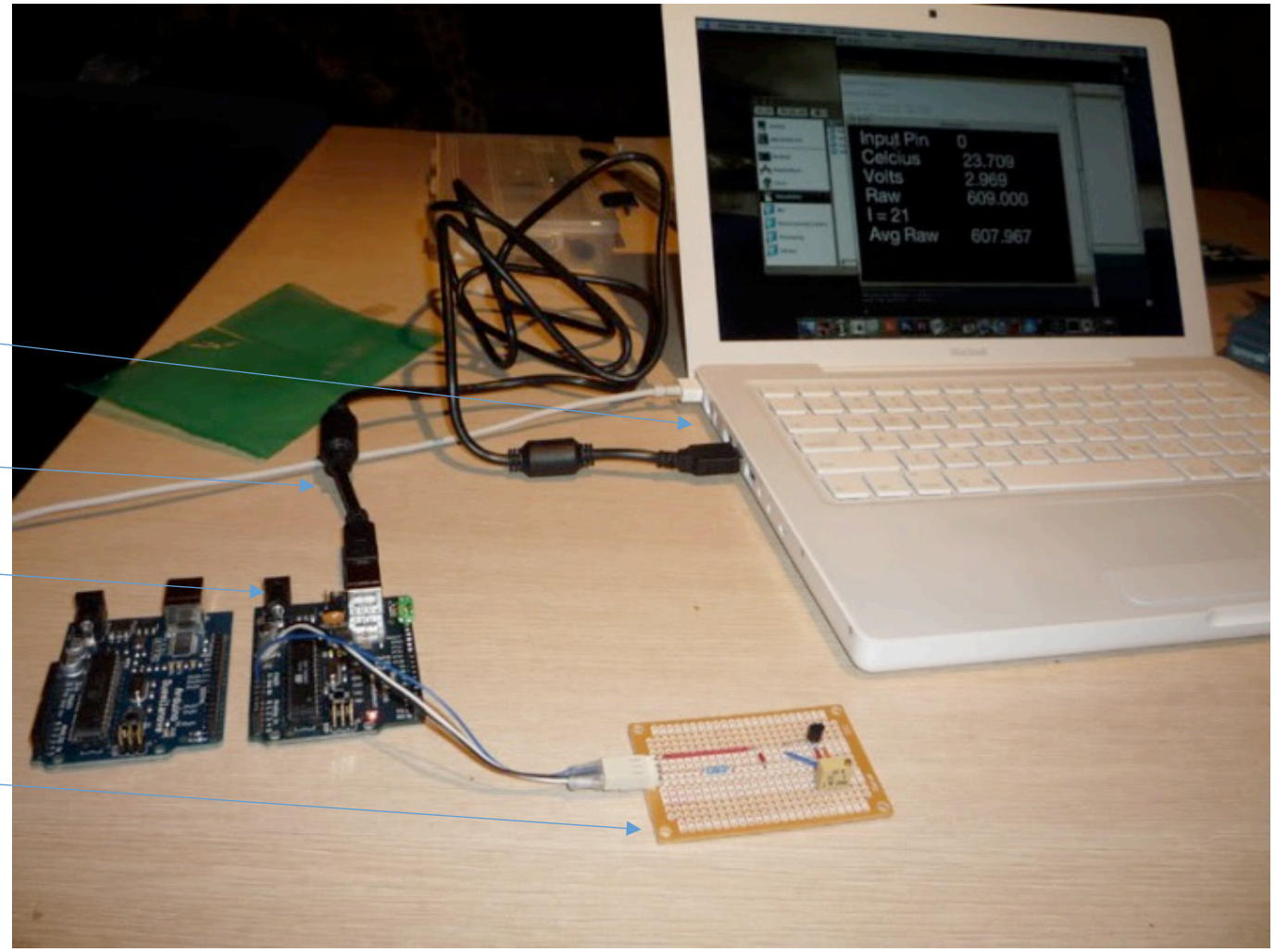
- **Exemplo:** interacção com uma porta série

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 síncrona
  - 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 recetor



Comunicação série usando porta USB entre PC e Arduino



Ficha USB

Cabo série

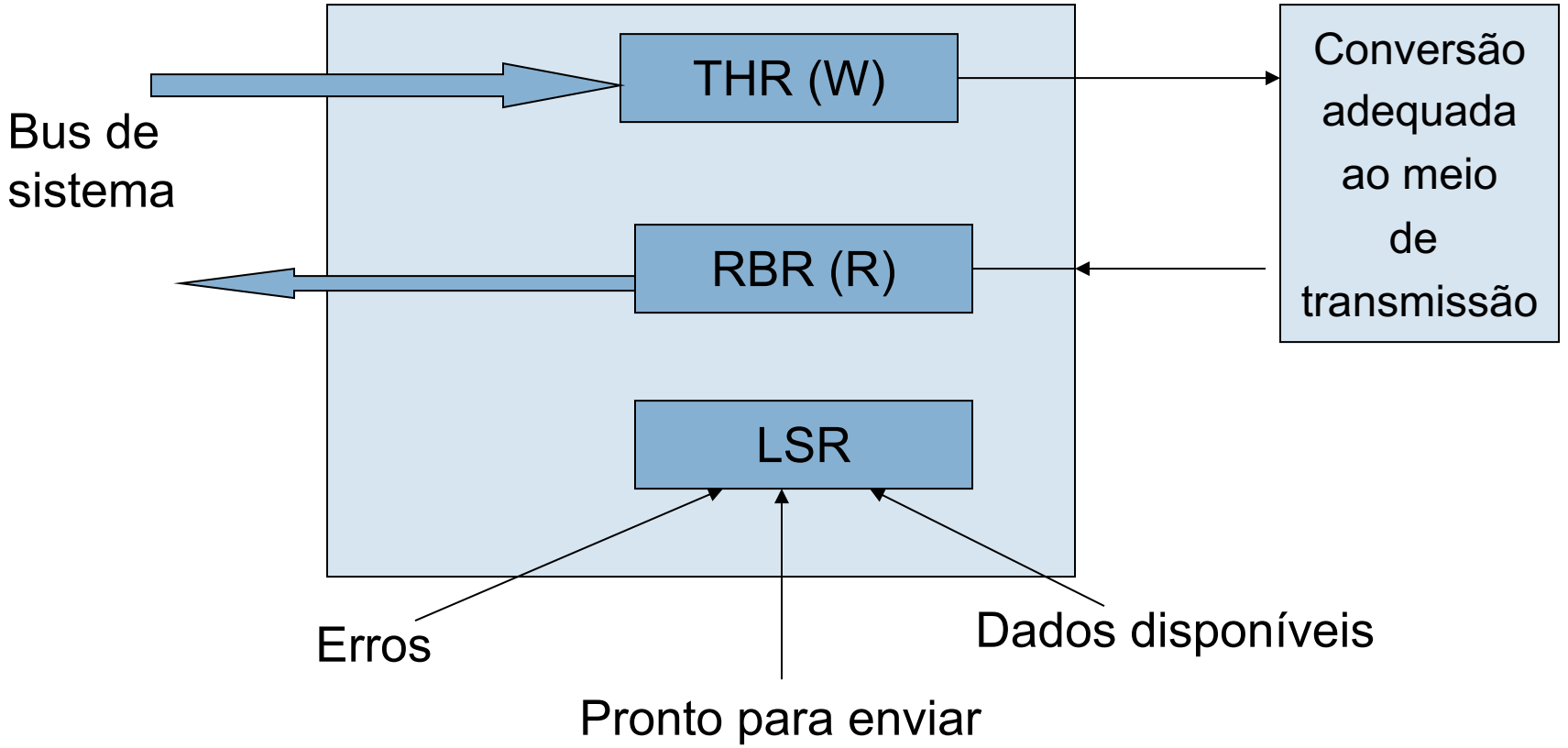
Arduino

Sensor de luminosidade

Programação por espera activa de um controlador série

- Porta série:
  - Programação de vários parâmetros:
    - Baud rate (inverso da “duração” de um bit)
    - Paridade (deteção de erros)
    - Número de bits de dados: ASCII (7 ou 8)
    - Número de “start bits” e “stop bits”

Porta série

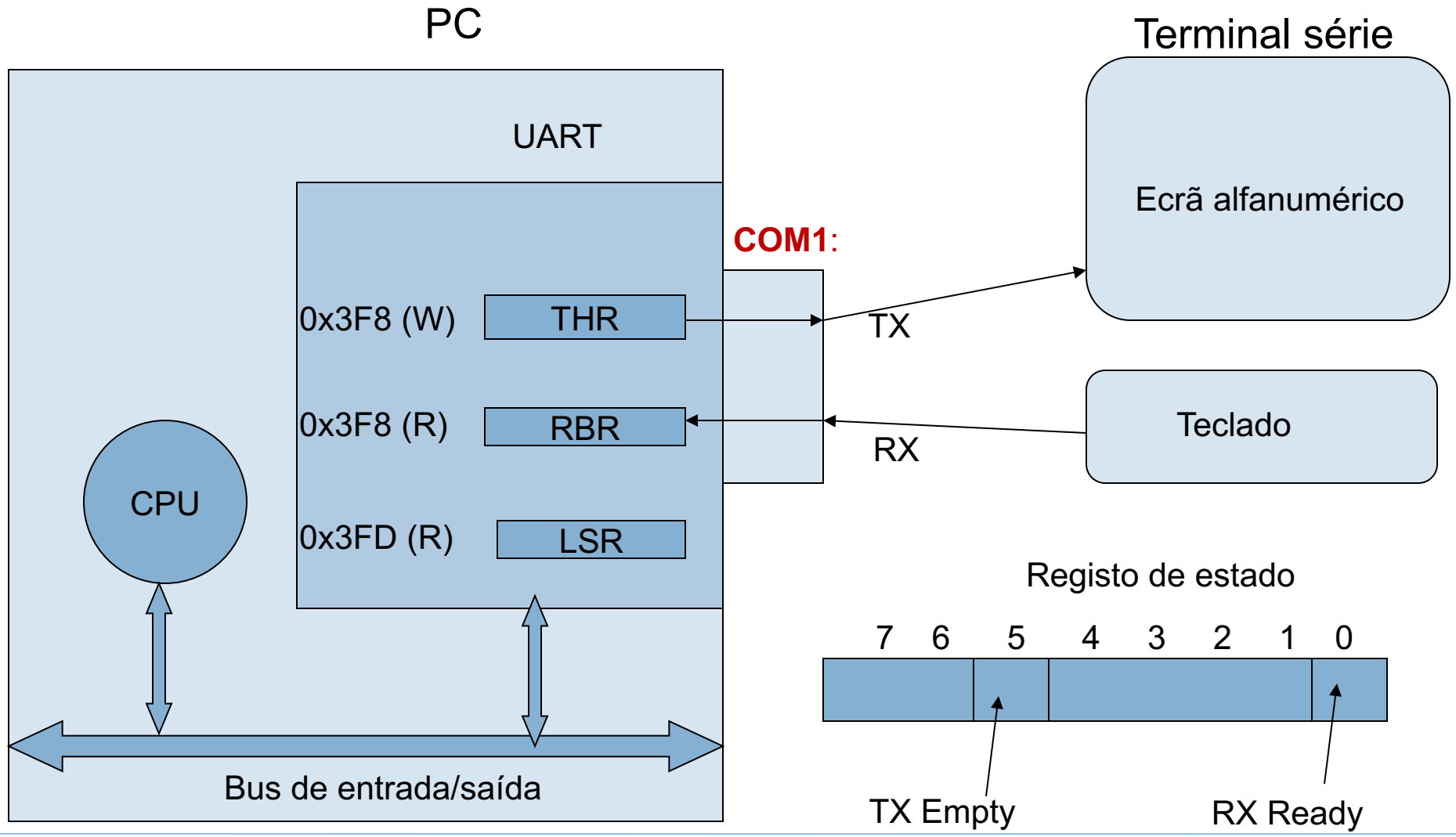


No PC, o controlador série é constituído por um único circuito integrado, chamado UART (Universal Asynchronous Receiver Transmiter).
O nome no Windows é COM1, no Linux /dev/ttys0

Porta série do PC

- Registo de dados de saída (THR – Transmission Hold Register)
  - Endereço: 0x3F8
  - A escrita neste registo desencadeia a serialização dos dados e saída para o exterior
- Registo de dados de entrada (RBR – Receive Buffer Register)
  - Endereço: 0x3F8
  - Este registo acumula um sequência de bits recebida em série
- Registo de estado (LSR – Line Status Register)
  - Endereço: 0x3FD
  - Bit 0 — a 1 indica que há um dado para ler em RBR; 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 THR passa a 0
  - Bits 1,2,3 — vários tipos de erros

Porta série do PC



Programação Porta série – Espera Ativa

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

```
unsigned char inportb( unsigned short endereço);
void outportb( unsigned short endereço, unsigned char valor);
```

- Para enviar um byte teremos

```
void send_serial( unsigned char b ){
  unsigned char s;
  do {
    s = inportb(0x3fd);
  } while(s & 0x20) == 0;
  outportb( 0x3f8, b);
}
```

Programação Porta série – Espera Ativa

- Para receber um byte teremos

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

  return inport( 0x3f8);
}
```

Inconvenientes do uso da Espera Ativa

- Os controladores que só suportam espera ativa são muito simples do ponto de vista hardware, mas implicam um grande desperdício de tempo de CPU
- Os dispositivos de E/S são muito lentos e o CPU vai passar grande parte do tempo nos ciclos do exemplo anterior
- O tempo de espera é fixo (depende do periférico) e independente da velocidade do CPU
- Enquanto se espera há potencial para o CPU executar muitas instruções

O mecanismo de interrupções aumenta a taxa de uso do CPU

- A invenção do mecanismo de interrupções (~1960) permitiu resolver a questão da desadequação das velocidades do CPU e dos periféricos
- Permite que o CPU continue a efetuar computações enquanto espera que a transferência de dados acabe
- O controlador atua autonomamente depois do CPU iniciar a operação de transferência; quando esta termina o controlador envia uma interrupção ao CPU.

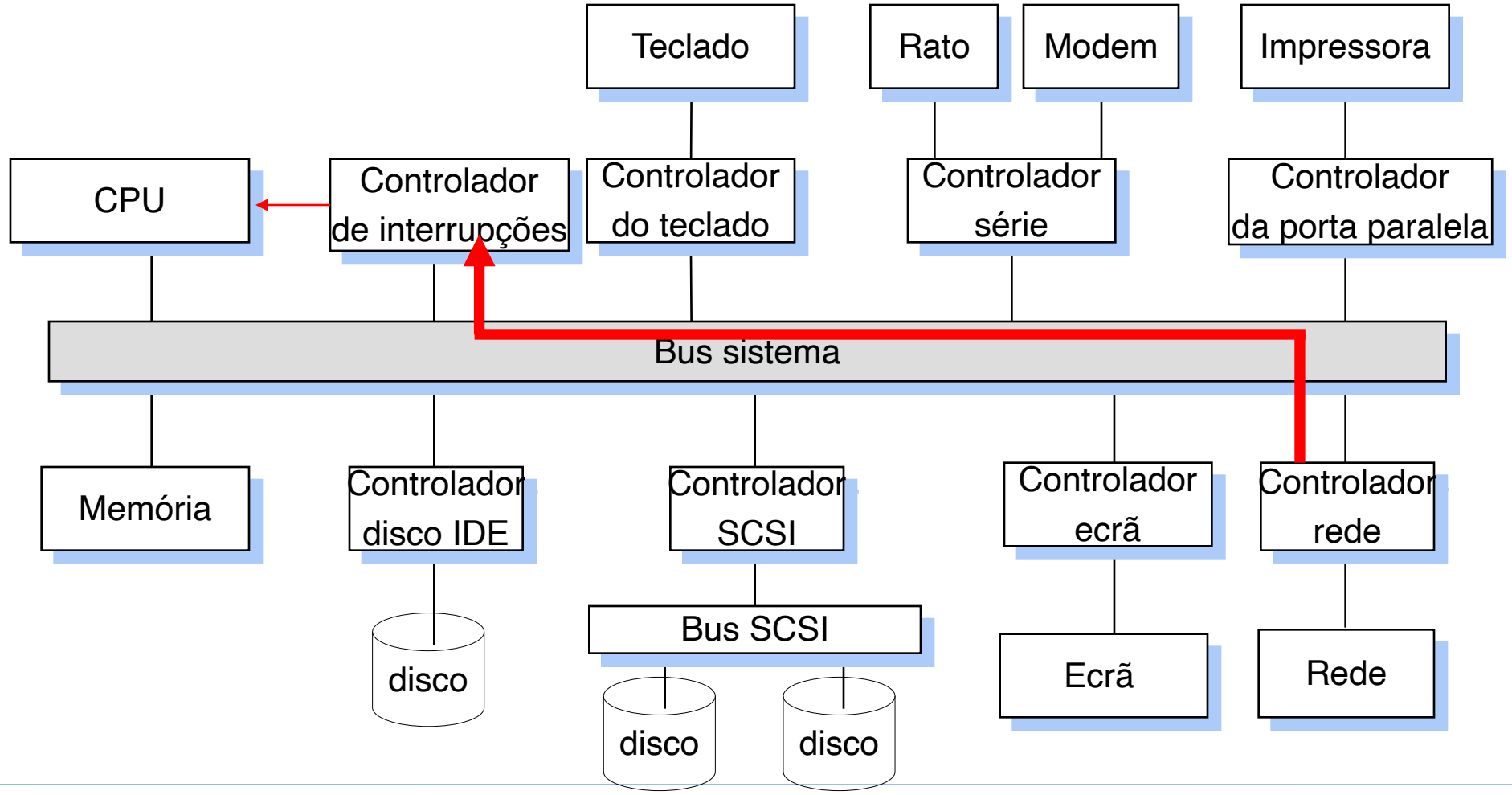
Entradas/saídas controladas por interrupções

- Resolvem o problema do desperdício do CPU
- O CPU não interroga periodicamente o controlador para saber o seu estado
- O controlador de E/S (I/O) interrompe o CPU quando terminou a transferência

Linhas gerais de uma entrada de dados controlada por interrupções

- CPU emite um comando de leitura
- O controlador de E/S obtém dados do periférico enquanto o CPU faz outra coisa
- O controlador de E/S interrompe o CPU
- CPU faz acesso ao controlador e transfere o dado recebido para um registo do CPU

Interrupções assinalam conclusão da operação ou erro



Linhas gerais de uma saída de dados controlada por interrupções

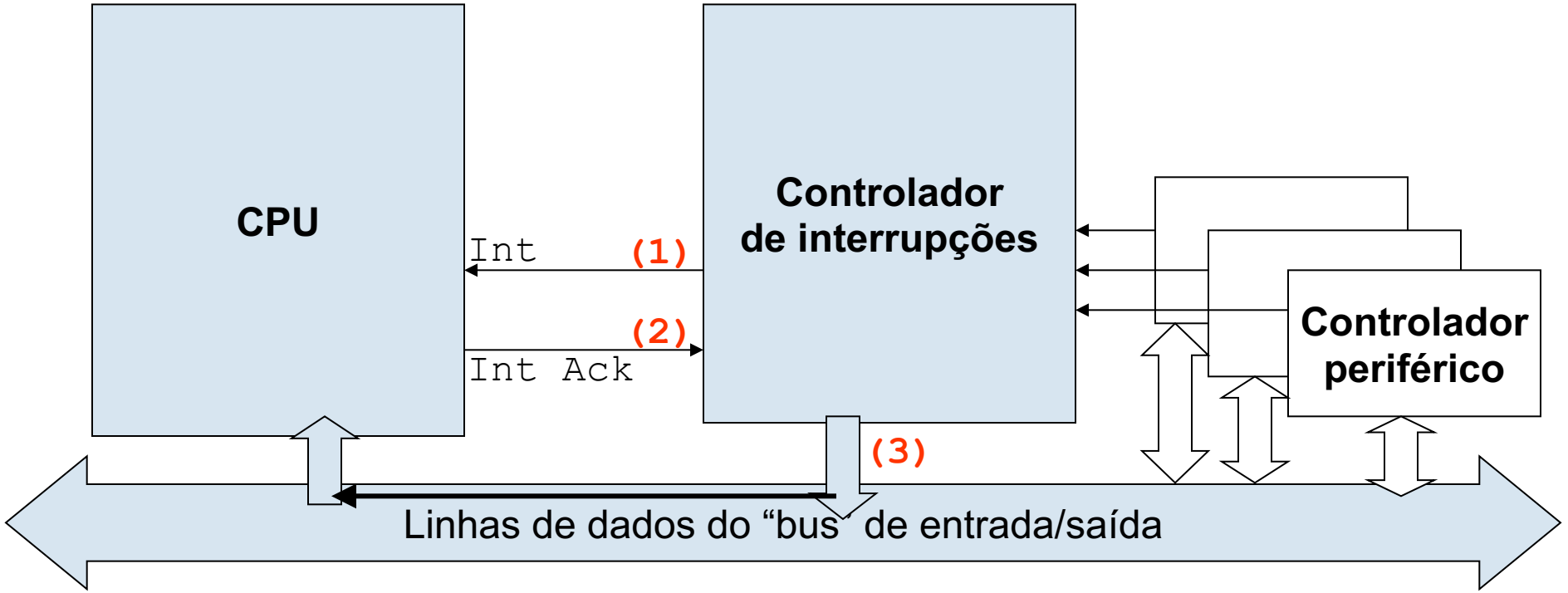
- CPU está a executar código em modo Utilizador
- Quando acaba um envio de dados, o controlador de E/S interrompe o CPU
- O CPU faz acesso ao controlador colocando no registo de dados de saída o próximo elemento a enviar
- CPU dá ordem para iniciar a transferência (pode ser implícito no ponto anterior)

Como identificar o controlador que fez a interrupção?

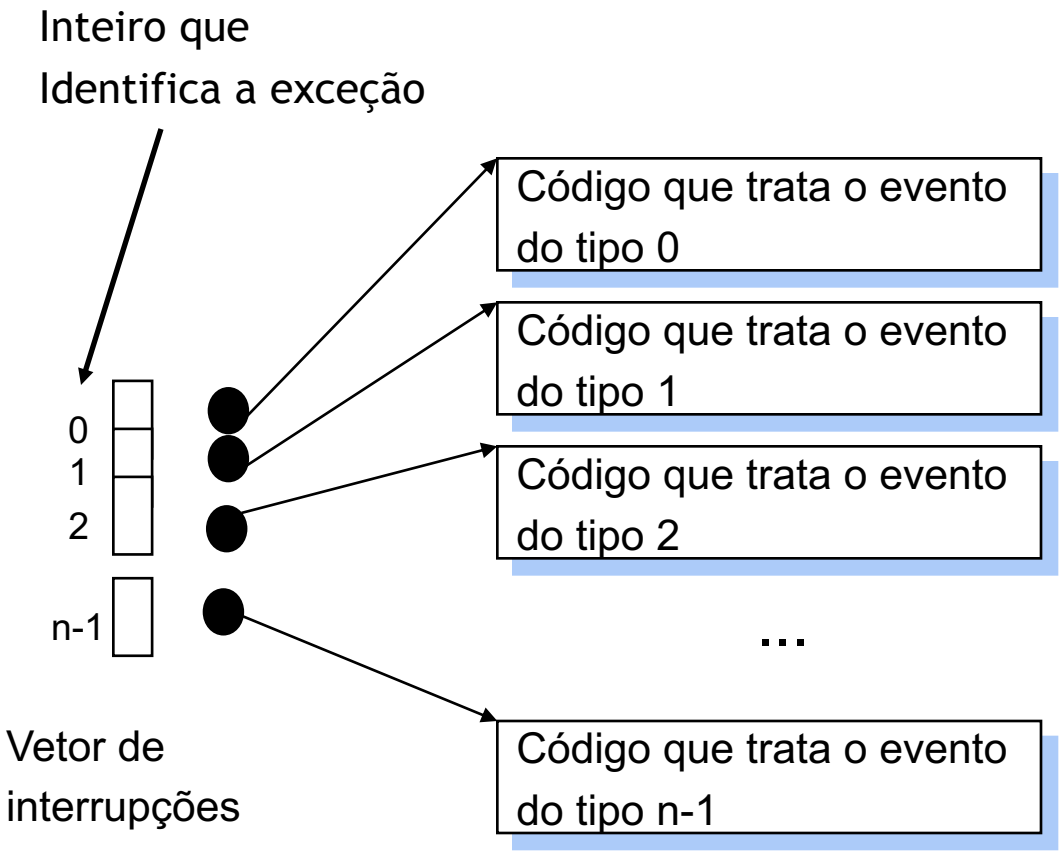
- Uma linha diferente para cada controlador?
  - Limita o número de controladores (e dispositivos)
- Interrogação por software?
  - O CPU consulta o registo de estado de todos os controladores
  - Lento
- Identificação por hardware?

Controlador de Interrupções

- Protocolo entre CPU e Controlador
  - (1) Controlador ativa a linha de interrupção
  - (2) CPU aceita a interrupção e ativa a linha de "interrupt acknowledge"
  - (3) O controlador coloca um valor no "bus" de dados; o CPU usa esse valor para identificar a rotina de tratamento a chamar



Vetor de interrupções (Relembrar)



- Cada controlador de periférico terá associado um tipo distinto

Inicialização do vetor de interrupções

- O software de sistema inicializa o vetor de interrupções antes de ocorrer a 1a. Interrupção
- Os controladores só começam a lançar interrupções depois de instruídos pelo CPU para o começarem a fazer
- Por segurança, quando o CPU começa a funcionar o sistema de interrupções está desligado

O estado do sistema de interrupções é guardado numa “flag”

- O estado do CPU contém uma “flag” Interrupt Flag (IF) que indica se as interrupções estão ou não habilitadas
- Inicialmente a IF está a 0 – interrupções não permitidas

- Há instruções máquina para ligar e desligar as interrupções
  - Enable interrupts; EI ; IF ← 1
  - Disable interrupts; DI ; IF ← 0

- São naturalmente instruções privilegiadas

Ciclo de Execução do CPU (Atualização)

```
while (1){
   Fetch : obter a instrução na posição de memória apontada pelo PC; atualizar PC
   Execute : executar a instrução
   if (Interrupt Flag == 1) {
            if ( interrupção pendente) {
                salvar o PC e as "flags" (usualmente no "stack")
                mudar para modo supervisor
                identificar a origem da interrupção
                carregar o PC com o endereço da rotina que
                        faz o tratamento da situação (interrupt handler)
            }
   }
}
```

Ligar/desligar interrupções

- Instruções privilegiadas
  - Enable Interrupts (IF ← 1)
    - No IA-32 é **STI** (Set Interrupt Flag)
  - Disable Interrupts (IF ← 0): no ciclo de fetch/execute não é testado se há interrupções pendentes
    - No IA-32 é **CLI** (Clear Interrupt Flag)
  - Inicialmente quando é aplicada energia ao CPU, IF está a 0

- Para tornar a escrita das rotinas de interrupção mais simples:
  - IF é colocada a 0, quando o teste de que há uma interrupção pendente tem sucesso

Impedir que a rotina de tratamento de interrupções seja interrompida

- Quando se entra na rotina de tratamento, IF está a 0
- Se nada for feito, só será colocada a 1 quando houver o “interrupt return” (**iret**)
- Mas as interrupções devem estar fechadas o menos tempo possível – se isso acontecer podem-se perder dados ou tornar a sua saída mais lenta.
- A rotina de tratamento deve fazer EI (IF ← 1) assim que for seguro.

Interrupções de múltiplos controladores

- As necessidades dos controladores diferem:
  - Tempo máximo ao fim do qual têm de receber atenção
  - Duração do processamento da interrupção
- O processamento de uma interrupção de um periférico lento pode levar mais tempo do que o tempo máximo de espera de um periférico rápido
- Para satisfazer estas diferenças são atribuídas prioridades diferentes aos controladores

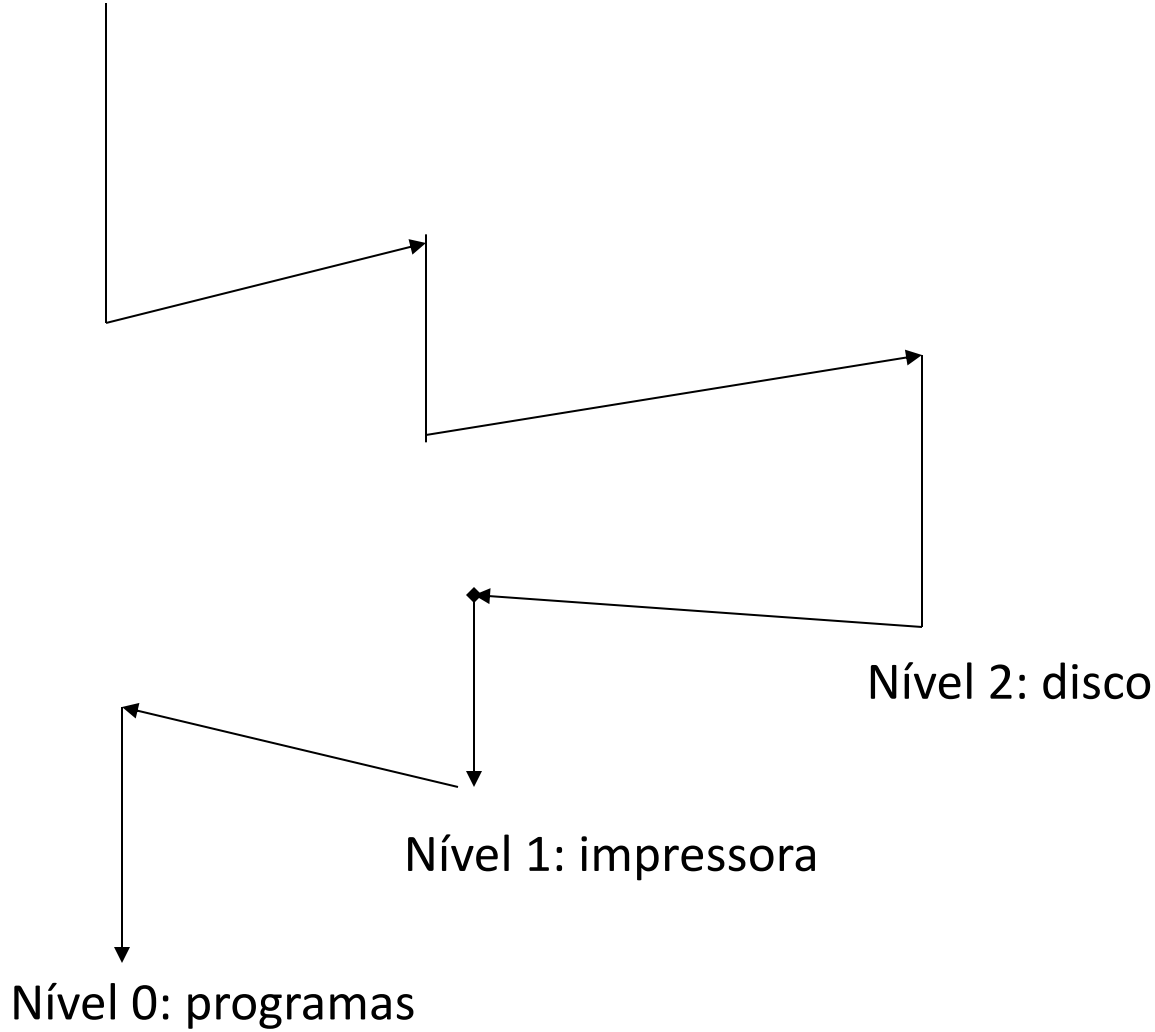
Prioridade das interrupções

- Múltiplos níveis de interrupção ou múltiplas prioridades para as interrupções
- O CPU (e o seu controlador de interrupções) definem vários níveis para as prioridades (8, 16 ou mais)
- A cada periférico é atribuída um nível N (N>0)
  - N-1 : prioridade máxima
  - 0: prioridade mínima
- A rotina de atendimento de interrupções do nível I pode ser interrompida para executar a rotina de atendimento de um periférico de nível J se J > I

Prioridade das interrupções

Exemplo:

- Periférico rápido (disco) prioridade/nível 2
- Periférico lento (impressora) prioridade/nível 1
- Se estamos a executar a rotina de atendimento da impressora e chega uma interrupção do disco: suspende-se a rotina da impressora e vai-se executar a do disco.
- Quando esta acaba voltamos à rotina da impressora; quando esta acaba volta-se ao programa que estava a correr (nível 0)



Prioridade das interrupções

- Se o nível 0 é para os programas e há mais K-1 níveis pode haver k-1 rotinas de atendimento em curso
  - só pode haver uma de cada nível

- Note-se que o nível nada tem a ver com o índice do controlador no vetor de interrupções

Interrupções múltiplas

- Cada linha de interrupção tem uma prioridade

- Uma linha de alta prioridade chega ao CPU mesmo que outra linha de interrupções (menos prioritária) esteja ativa

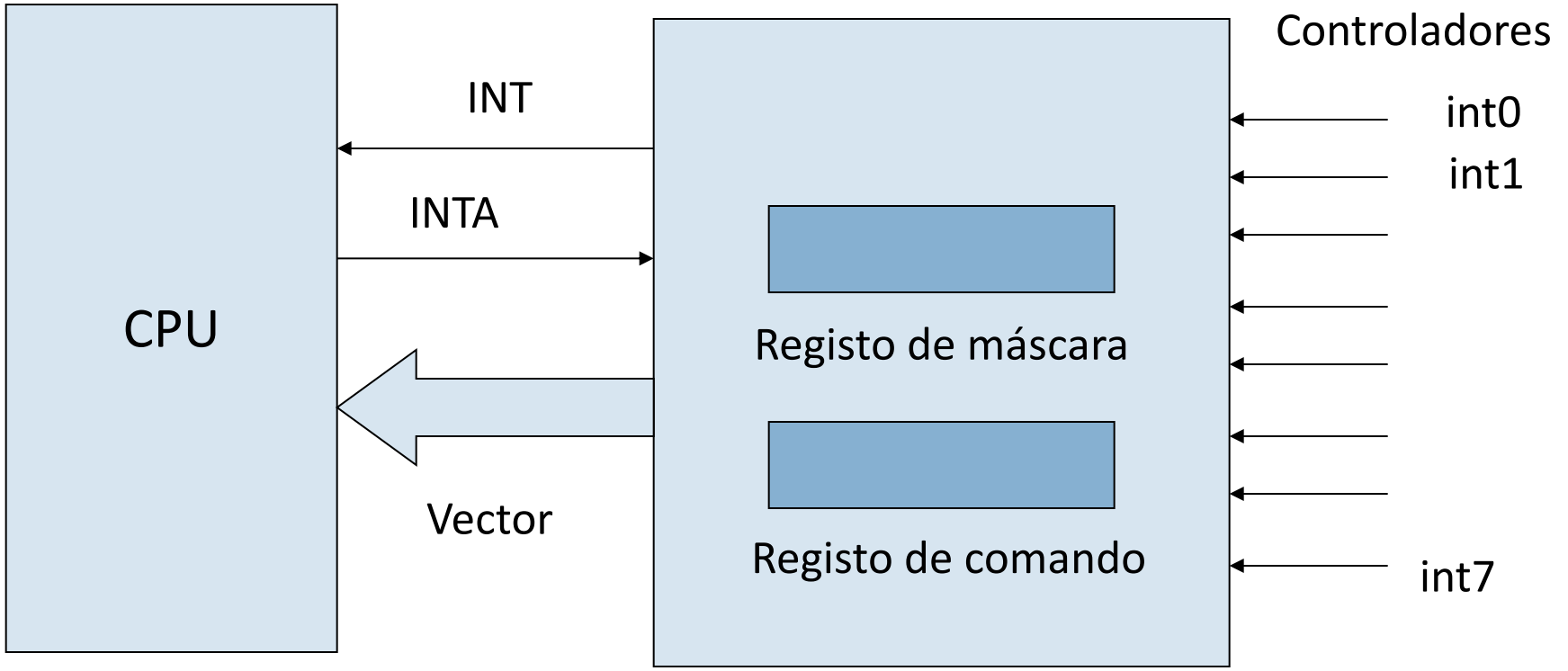
Exemplo – Bus PC

- Pentium tem uma linha de interrupções
- O *chipset* tem um controlador de interrupções 8259A
- O 8259A tem 8 linhas de interrupção

Sequência de eventos

1. Periférico envia interrupção
2. 8259A aceita a interrupção
3. 8259A determina a prioridade
4. 8259A alerta CPU (ativa o pino INTR)
5. CPU faz “acknowledge”
6. 8259A coloca o vetor correspondente no bus de dados
7. CPU salta para a rotina de tratamento de interrupções

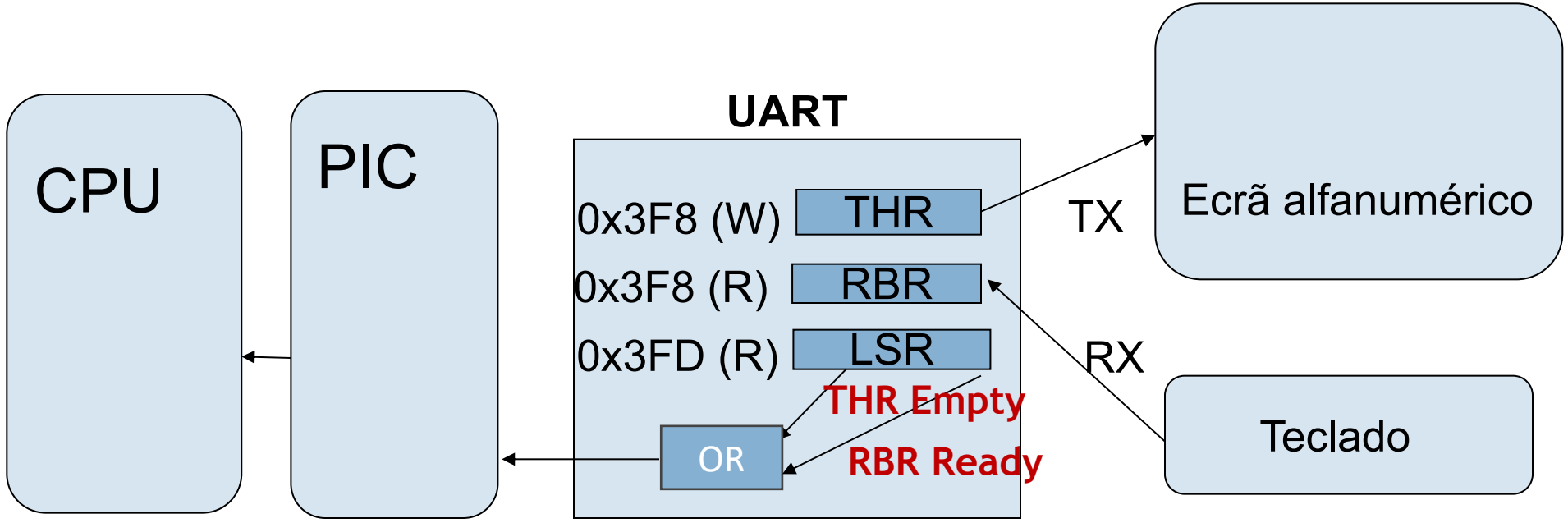
Controlador 8259A (PIC)



- Registo de máscara (endereço 0x21): a 0 indica que a linha i está activa
- Registo de comando (endereço 0x20): permite programar a forma de atendimento
  - End of Interrupt (EOI – valor 0x20) informa o PIC que a rotina de atendimento acabou

Interrupções na UART

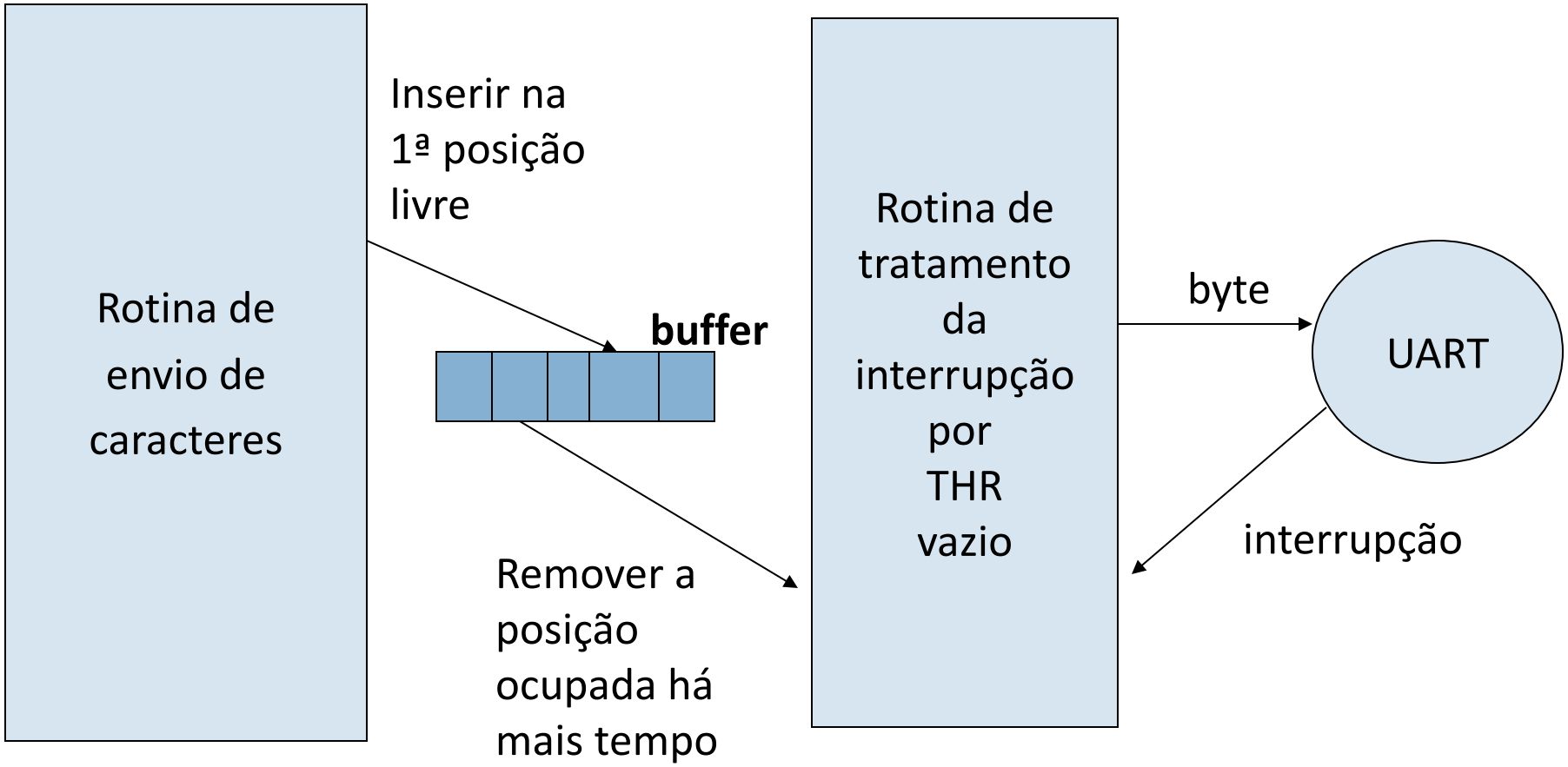
- COM1:
  - IRQ4 (linha 4)
  - índice 12 do vector



Interrupções na Porta Série

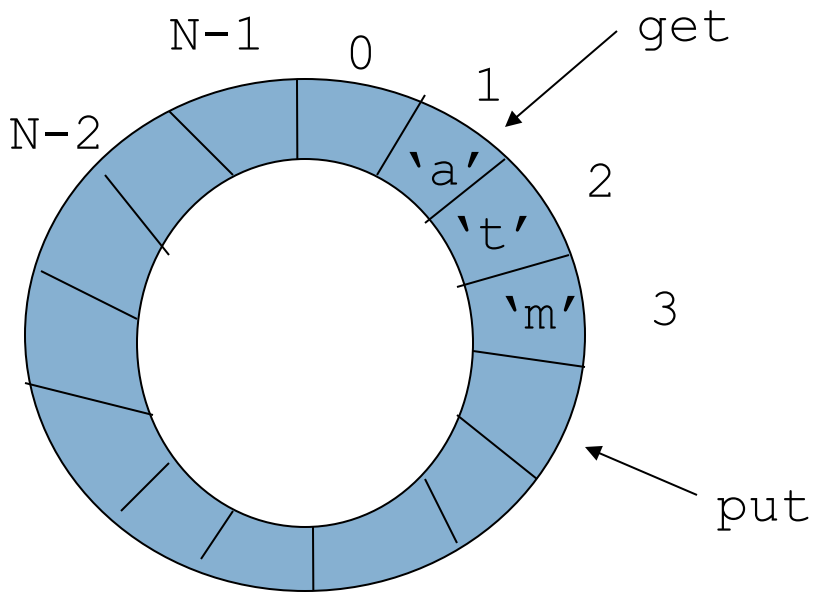
- Programar o PIC para deixar passar as interrupções da linha 4
  - Colocar o bit 4 do registo de máscara a 1
- Para a UART gerar interrupções é preciso:
  - Colocar a 1 o bit 3 do registo MCR (Modem Control Register) - 0x3FC
- Depois é necessário indicar quando é que queremos que sejam geradas:
  - Para tal modificamos o registo IER (Interrupt Enable Register) – 0x3F9:
  - Interrupção quando chega um carácter (RBR READY):
    - → Colocar a 1 o bit 0
  - Interrupção quando o buffer de transmissão está vazio (THR EMPTY), ou seja, o controlador está pronto a enviar
    - Colocar a 1 o bit 1

Transmissão com interrupções



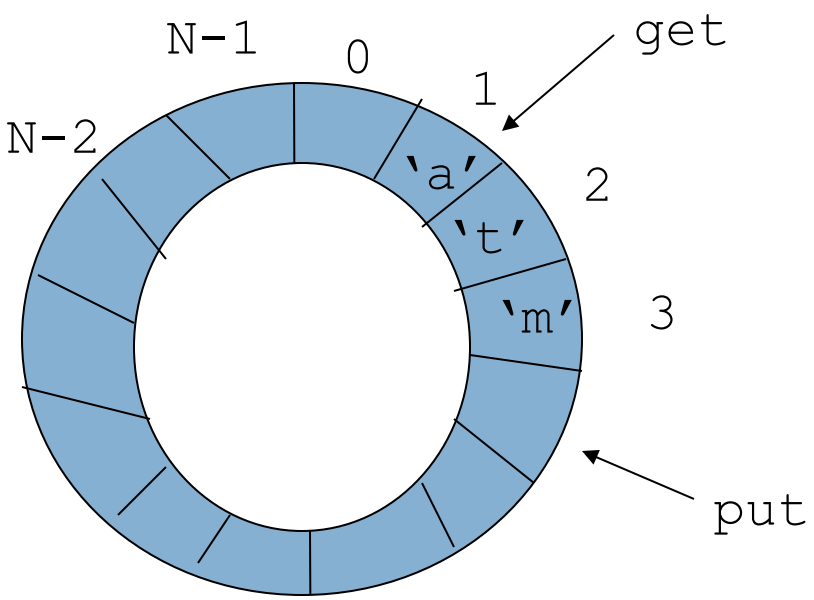
Inicialização do “buffer” circular

```
#define BUFSIZE 128
unsigned char buf[BUFSIZE];
int put;  // 1ª casa livre
int get;  // casa ocupada há mais
          // tempo
int nc;   // nº de bytes no buffer
put = get = nc = 0; // inicial
```



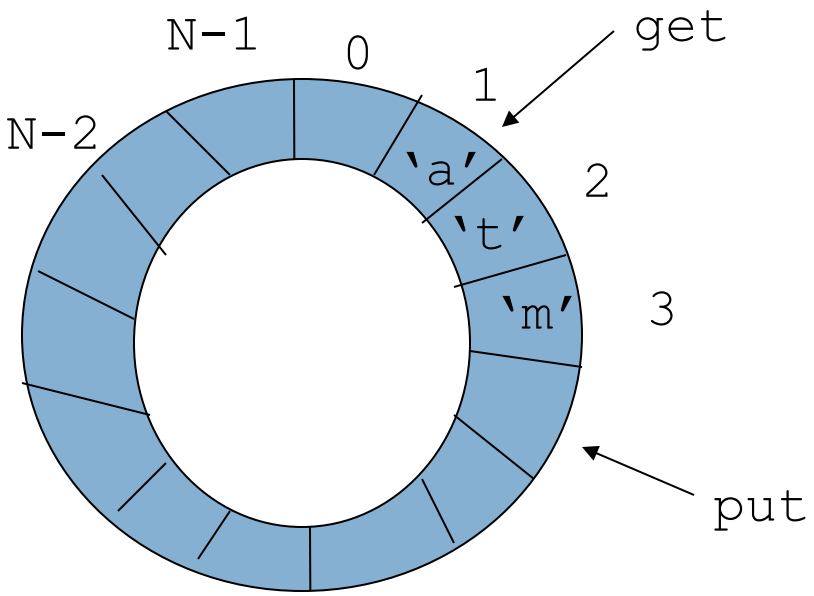
Colocar um byte no buffer circular

```
void bufPut( unsigned char c){
/* assume que buf não está cheio */
  buf[put] = c;
  put = (put + 1) % BUFSIZE;
  nc ++;
}

int bufFull(){
  return (nc == BUFSIZ);
}
```



Obter um byte do“buffer” circular

```
unsigned char bufGet(void){
/* assume que buf não está vazio */
  unsigned char x = buf[get];
  get = (get + 1) % BUFSIZE;
  nc --;
  return x;
}

int bufEmpty(){
  return (nc == 0);
}
```



Transmissão com interrupções

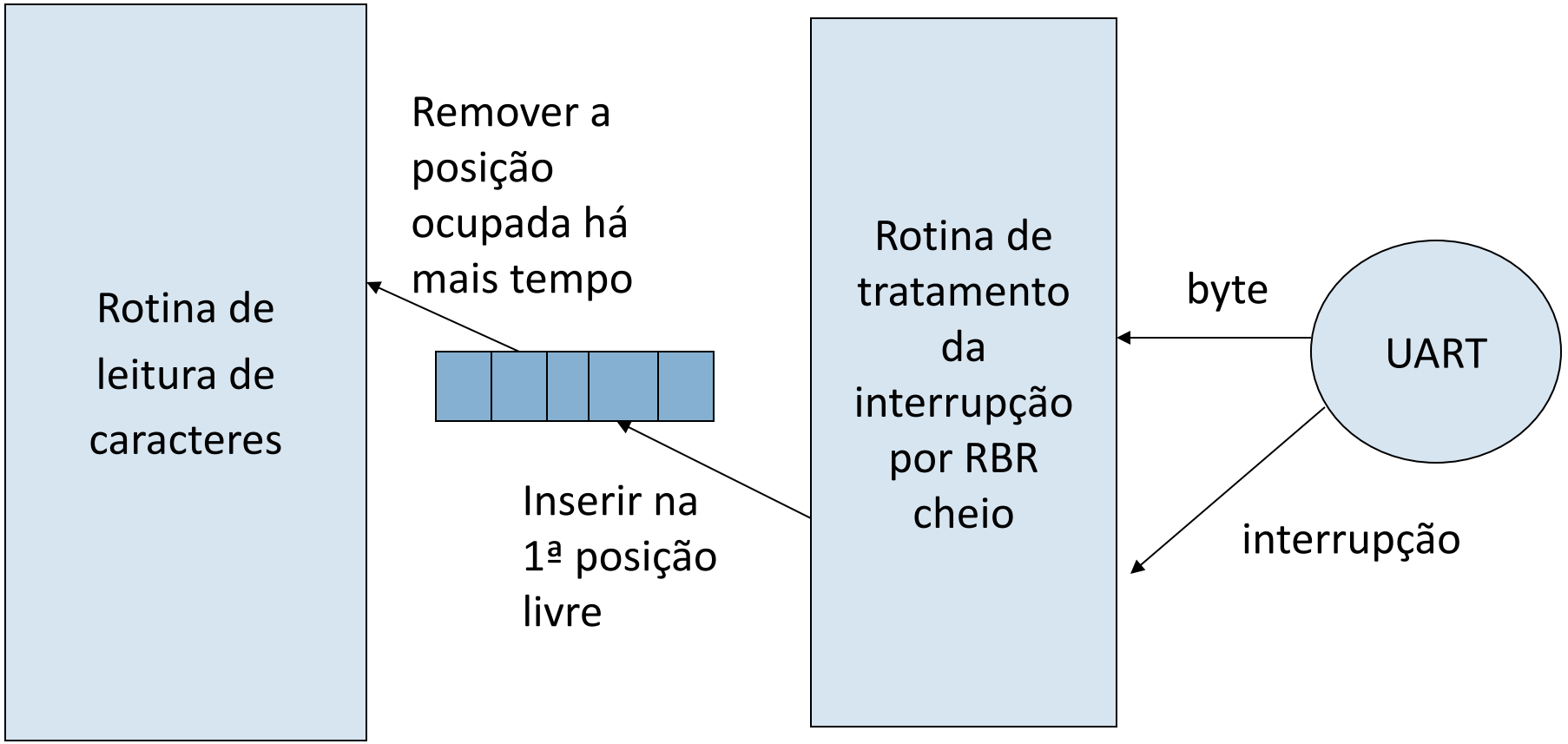
```
enviar_serie(ch){
    if bufFull()
        assinala erro
    if bufEmpty()
        bufPut(ch)
        ligar interrupç. THR_EMPTY
    else
        bufPut(ch)
}
```

```
Rotina de tratamento de interrupções
THR_EMPTY{
    ch = bufGet();
    outportb( THR , ch );
    if bufEmpty()
        desligar as interrupções THR_EMPTY

    outportb( 8259_COMMAND, EOI)
}
```

Notas importantes:

- Quando o hardware é inicializado as interrupções por THR EMPTY estão desativadas
- A rotina de interrupções desliga as interrupções da UART por THR EMPTY se o "buffer" fica vazio
- A rotina **enviar_serie** verifica se o "buffer" está vazio
  - se tal acontecer deposita o carácter no "buffer" e liga as interrupções na UART por THR EMPTY

Receção com interrupções



Receção com interrupções

```
unsigned char receber_serie(){
    while (bufEmpty())
        ; // espera
    disable();
    ch = bufGet();
    enable();
    return ch;
}
```

```
Rotina de tratamento de interrupções RBR_
READY{
    ch = inportb( RBR );
    if bufFull()
        // byte perdido, não há espaço no
        buffer mas não se pode esperar
    else bufPut(ch);
    outportb( 8259_COMMAND, EOI)
}
```

Notas importantes:

- As interrupções por RBR READY estão sempre ligadas
- A rotina de interrupções pode encontrar o “ buffer” cheio
  - se tal acontece não pode ficar em espera ativa
- Ver a razão da necessidade disable() – CLI e enable() – STI a seguir

Atualização do nº de bytes no “buffer”

- A rotina **enviar_série** incrementa o número de bytes no buffer
  - nc ++
  - O compilador traduz para

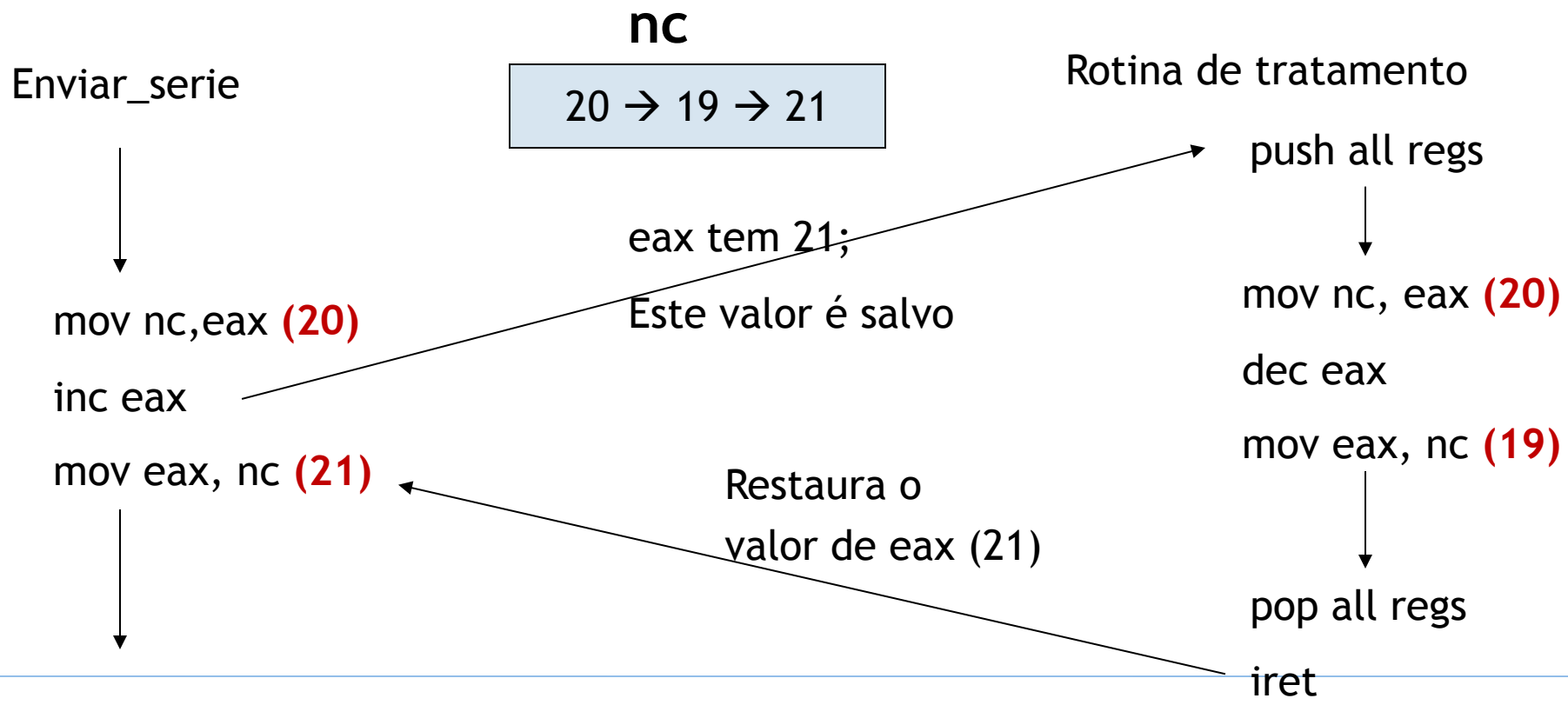
    movl nc, %eax
    incl %eax
    mov %eax, nc

- A rotina de atendimento de interrupções de transmissão decrementa o número de bytes no buffer
  - nc --
  - O compilador traduz para

    movl nc, %eax
    decl %eax
    movl %eax, nc

Atualização do nº de bytes no “buffer”

- Suponhamos que nc é 20 e que enquanto a rotina escrever série deposita um carácter, ocorre uma interrupção motivada pelo facto do registo THR ter ficado vazio



Atualização do nº de bytes no “buffer”

- Após inserir um carácter e remover outro nc deveria ter ficado com 20
- Ficou com 21 o que é um erro que vai provocar problemas para o futuro nas chamadas de bufEmpty() e bufFull()
- Isto aconteceu porque a ação de atualização da variável foi interrompida a meio; antes de voltar à atualização foi chamada uma rotina que atualiza a mesma variável
- A variável *nc* é partilhada pela rotina **enviar_série** e pela rotina de tratamento de interrupções.
  - A sua atualização não pode ser interrompida – constitui o que se chama uma **secção crítica**

Atualização do nº de bytes no “buffer”

- Para resolver isto é preciso alterar a rotina enviar_serie

  disable(); // execução da instrução máquina CLI: colocar a Int Flag do CPU a 0
  bufPut();
  enable(); // execução da instrução máquina STI: colocar a Int Flag do CPU a 1

- Assim garante-se que nc é atualizado corretamente

- A mesma alteração também tem de ser feita na função receber_serie()

Vantagens da interação CPU <-> Controladores por interrupções

- CPU deixa de fazer ciclos de espera ativa
  - **Receção**
    - Quando os bytes chegam são colocados no buffer pela rotina de tratamento de interrupção; o CPU continua a executar o programa
    - Quando precisa de ler bytes
      - Ou não há bytes disponíveis e dá erro
      - É retornado o byte que está há mais tempo no buffer
  - **Transmissão**
    - Os bytes a transmitir são colocados no buffer, sem que o CPU fique à espera que a transmissão efetiva termina; se não há espaço no buffer é devolvido um erro
    - A rotina de tratamento de interrupções vai enviando os bytes pela ordem pela qual foram depositados no buffer

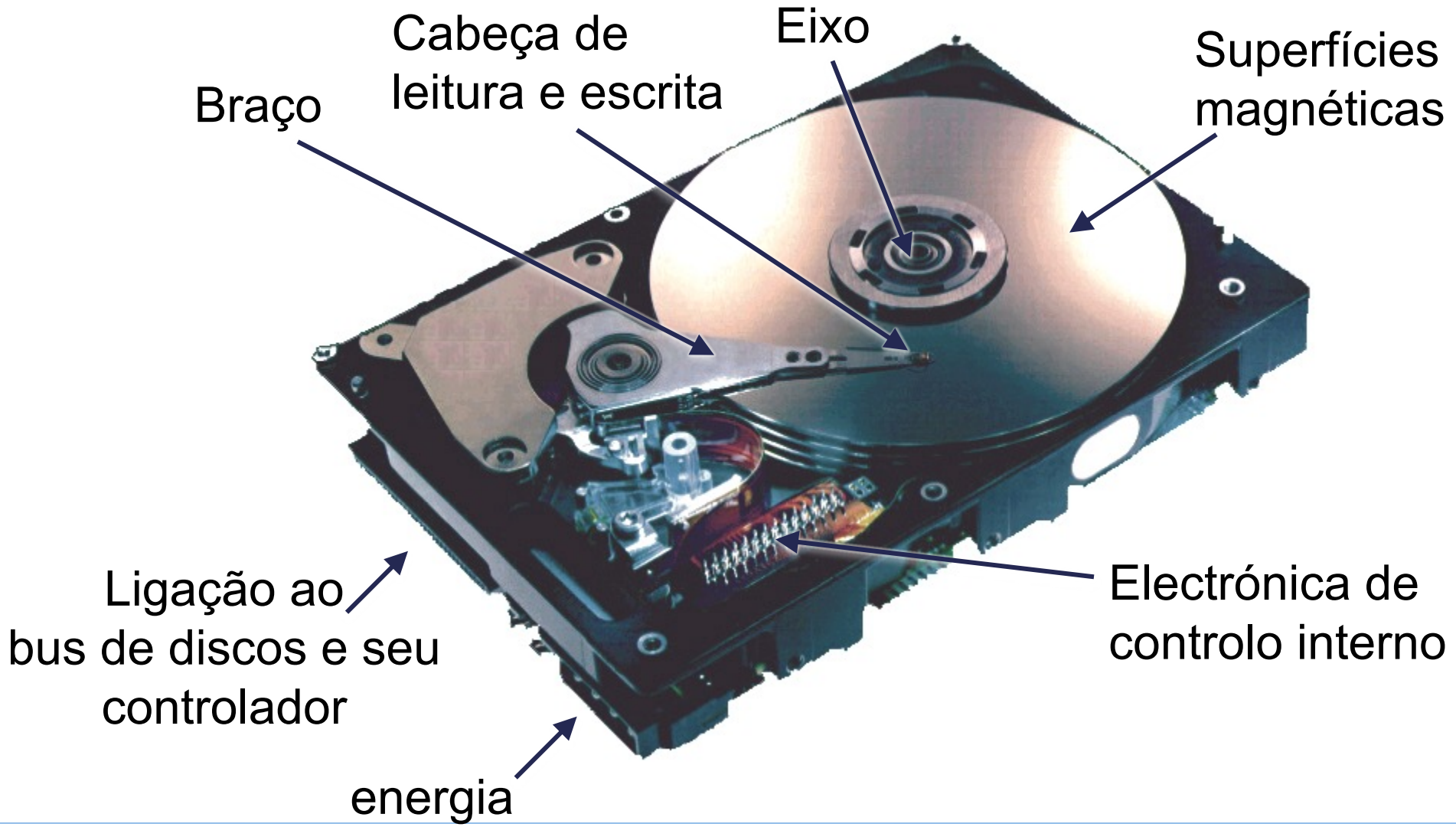
Unidade de transferência: byte-a-byte

- Exemplos: porta série (RS232), teclados, etc.
- Um teclado, disponibiliza um byte com o carácter premido; a porta série transfere um único byte
- O software tem de manipular cada um desses bytes individualmente
- Podemos “ver” estes periféricos para leitura e escrita de sequências de bytes
- Podemos “ver” estes periféricos como “ficheiros” acedidos via streams (ou canais) de bytes
  - stdin, stdout, ...

Unidade de transferência: bloco

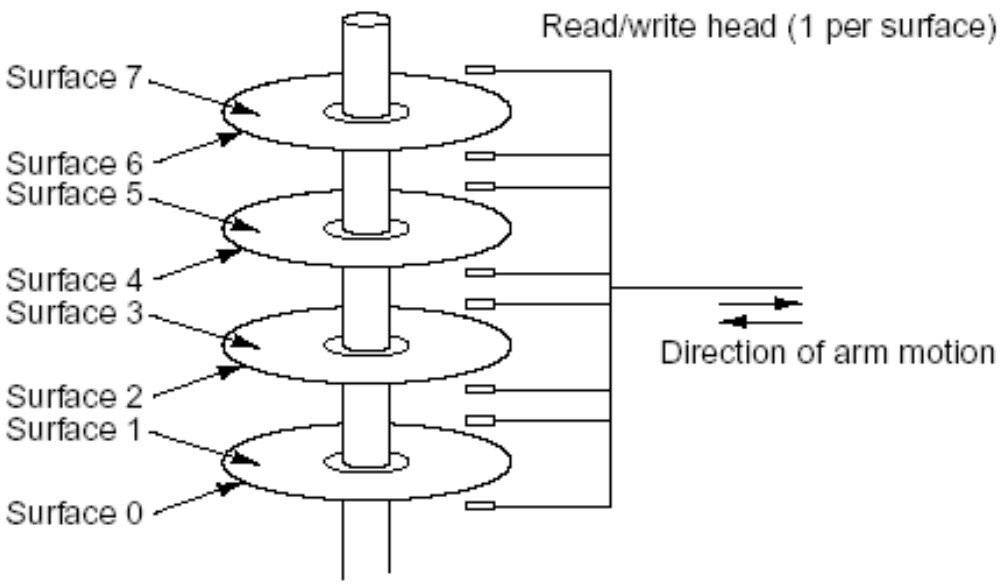
- Exemplo: disco, DVD, pen, etc
- O periférico só permite leitura/escrita de um bloco de bytes (512 bytes, 1K, etc)
- Podemos “ver” estes periféricos como endereçáveis mas orientados a blocos
  - Cada endereço representa um bloco
- Escrever sequencias de bytes significa escrever num buffer até completar um bloco; então o bloco pode ser escrito
  - Na leitura, lê bloco para buffer e vai lendo desse buffer

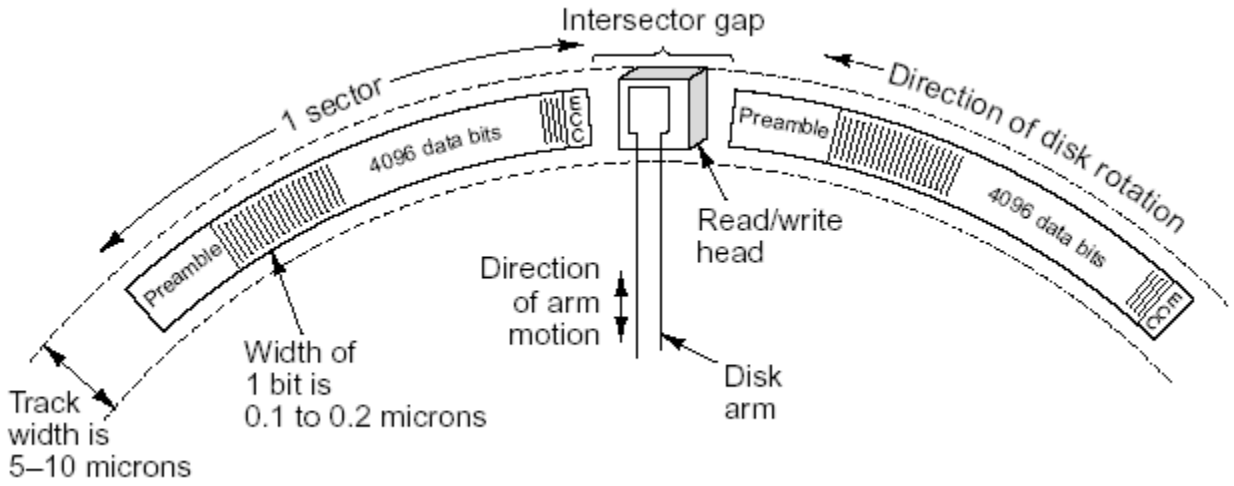
O que há dentro de um disco magnético?



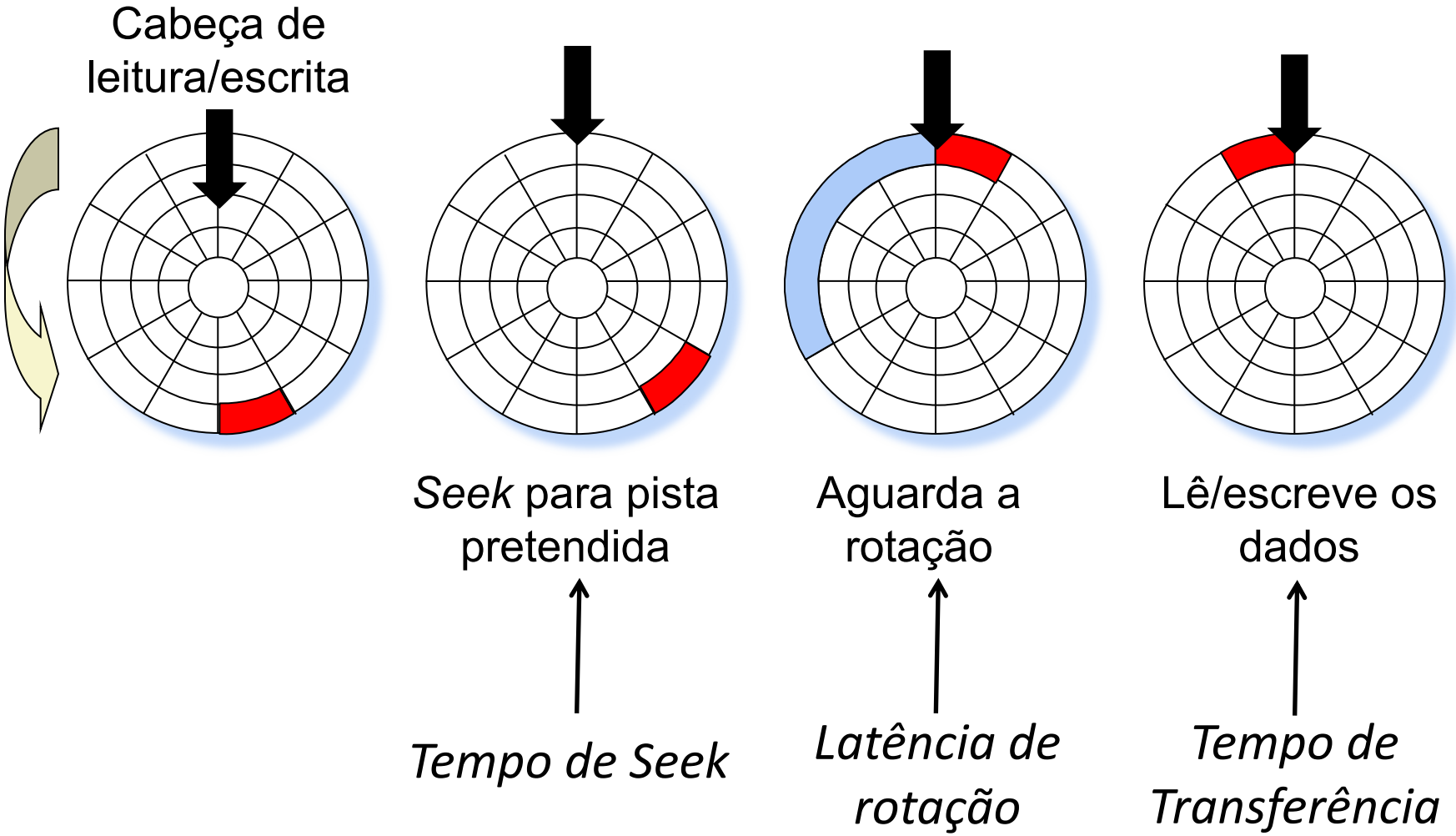
Discos – periféricos orientados ao bloco

- A unidade de transferência é o sector (512bytes, 1K, etc)





Componentes do tempo de serviço

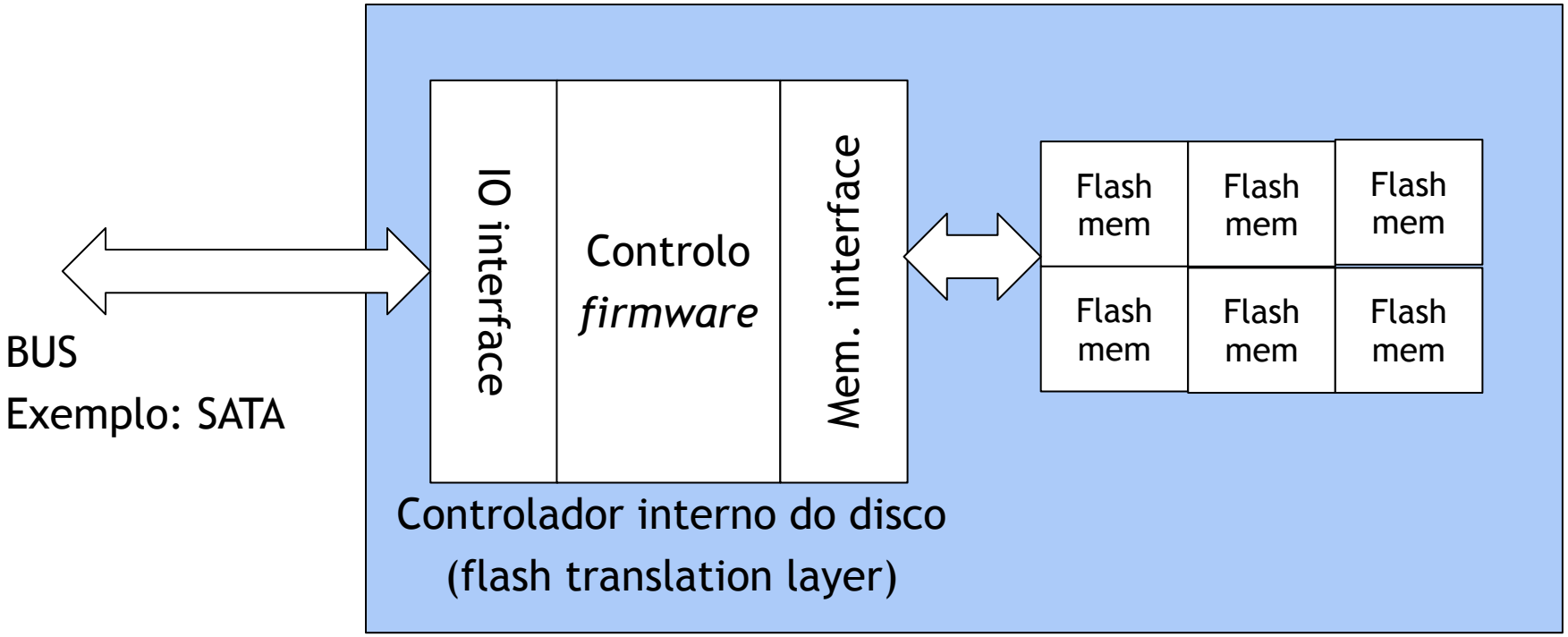


Exemplo de cálculo de tempo de acesso

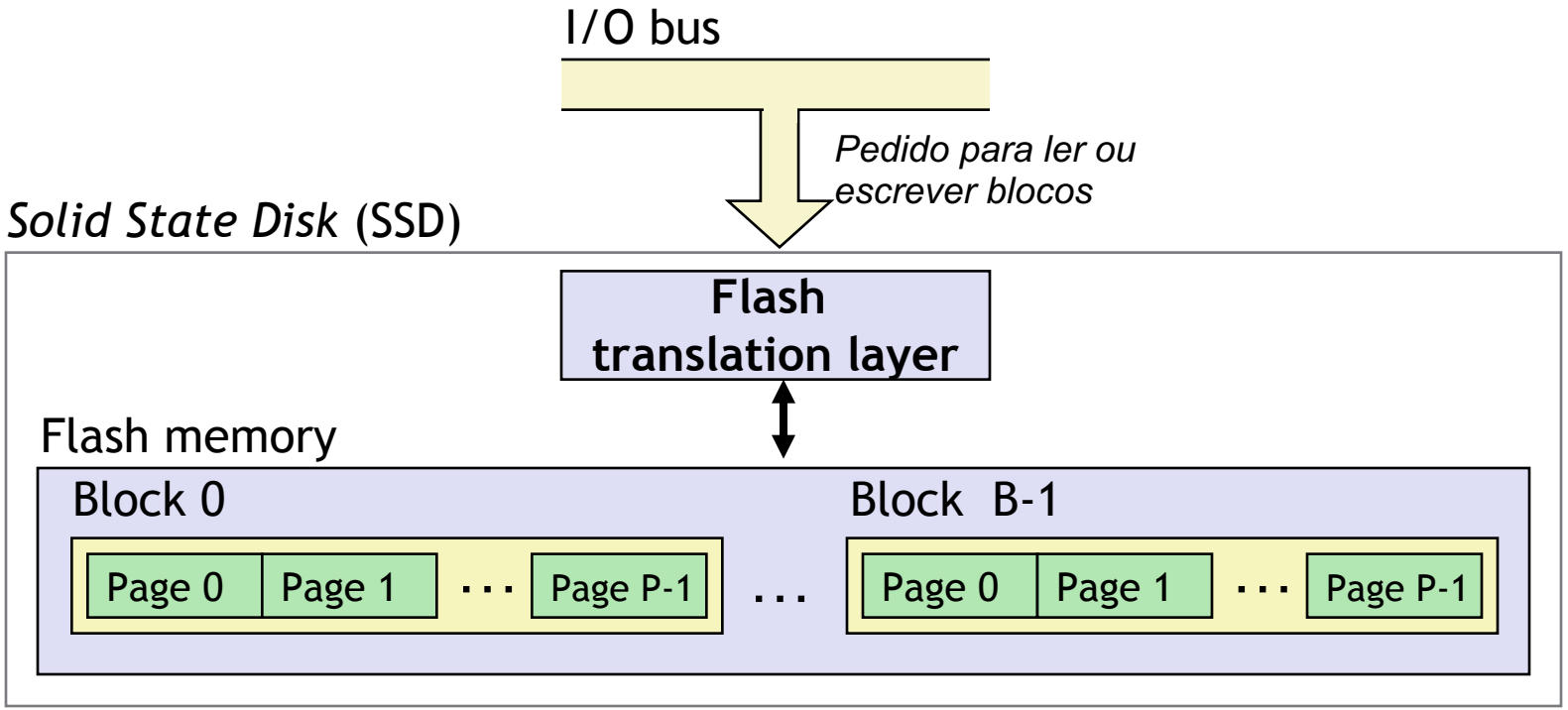
- Dados:
  - Velocidade de rotação = 7200 RPM
  - Tempo médio de seek = 9 ms.
  - Nº médio de setores/pista (spp) = 400.
- Determina-se:
  - Tmédio de rotação = Tmax_rotação/2 = (60 s/7200 RPM)/2 * 1000 ms/s = 4 ms.
  - Tmédio de transferência = (60/7200 RPM)/400 spp * 1000 ms/s = 0,02 ms
  - Tmédio acesso = Tseek + Tmédio de rotação + Tmédio de transferência
    = 9 ms + 4 ms + 0,02 ms = 13,02 ms
- Pontos importantes:
  - Tempo de acesso **dominado pelo seek time e pela latência de rotação**.
  - 1º bit de um sector é o que demora mais, o resto é de “graça”.
  - Acesso a sectores consecutivos poupam no seek e rotação
  - O disco pode ser 100 000x mais lento que a memória central

Solid State Disks (SSDs)

- Discos baseados em memoria FLASH (que também é a base das pens USB, cartões de memória, ...)



Solid State Disks (SSDs)



- Os dados estão em páginas (eq. setores) de 512B a 4KB, dentro de blocos
  - 32 a 128 páginas por bloco
- As escritas só são possíveis em páginas "apagadas"
- Uma página só pode ser apagada, apagando todo o bloco
- Um bloco fica inutilizado após ser apagado (antes ~100 000 vezes, atualmente cerca de 2000 a 3000 vezes)

Solid State Disks (SSDs)

- A escrita de uma página **pode** exigir apagar o bloco
- Apagar um bloco é lento (cerca de 1 ms)
- A escrita de uma página pode desencadear a cópia de todas as páginas em uso no bloco:
    1. Escolher um bloco livre e apagá-lo se necessário
    2. Copiar as páginas a manter do bloco antigo para o novo
    3. Escrever a página nesse bloco
    4. Marcar o bloco antigo como livre
- Esta gestão é feita pelo *firmware* interno ao disco

Comparação entre SSD e discos magnéticos

- Vantagens
  - Memórias sem partes móveis→ maior rapidez e robustez, menos consumo
  - Mais rápidos
    - HDD 7200 RPM → 80-160 MB/s
    - SSD → 200 a 500 MB/s

- Desvantagens
  - Tem o potencial para se gastarem
    - Minimizado pela "*wear leveling logic*" na *flash translation layer*
    - E.g. Intel X25 garante 1 petabyte de escritas aletórias até que o disco se torne inútil
  - Preço mais caro por byte (tem vindo a diminuir)

Unidade de transferência: bloco (p.e. setor)

- Exemplo: disco, DVD, pen, etc
- O periférico só permite leitura/escrita de um bloco de bytes (512 bytes, 1K, etc)
- Podemos “ver” estes periféricos como cada endereço representando um bloco
- Como o software copia esses blocos de memória para o respectivo controlador e vice-versa?
  - Em bytes/words etc por espera ativa?
  - Em bytes/words etc usando interrupções por cada um?

Transferência de blocos

- Exemplo: enviar 1000 bytes usando interrupções:

```
Enviar_Bloco()
    por_bloco_num_buffer()
     ligar_transf_intr()
    aguarda_transferencia_pelo_controlador()
    manda_controlador_escrever_bloco()

RotinaServiço:
    envia prox. dword (ou word, ou byte) do buffer p/ controlador
    se último, desliga_transf_intr()
```

O CPU pode ser interrompido 250 vezes (ou até 1000) para executar a RotinaServiço até transferir tudo para o controlador.

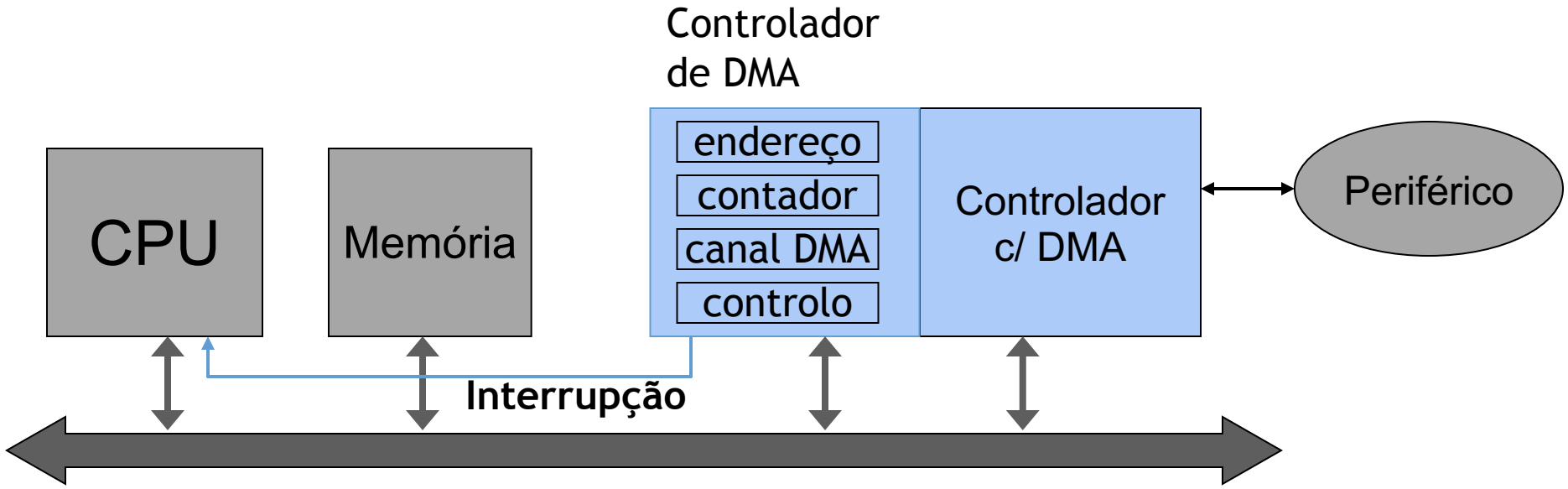
Acesso Direto à memória
Direct Memory Access (DMA)

- O I/O usando interrupções ainda requerem uma intervenção ativa do CPU nestes periféricos
  - Para um bloco de dados o CPU precisa de intervir muitas vezes
  - A resposta a estes problemas é dispensar, nestes casos, a necessidade de IN e OUTs ou interrupções a cada byte/word/dword

- Pretende-se uma nova funcionalidade hardware que permita transferências entre Memória e Controladores sem intervenção do CPU: Acesso Direto à Memória

Controlador de DMA. **Exemplo**

- Registos de um controlador genérico:
  - Endereço – endereço de memória
  - Contador – número de bytes a transferir
  - Controlo – comandos, por exemplo: sentido da transferência



Operação com DMA

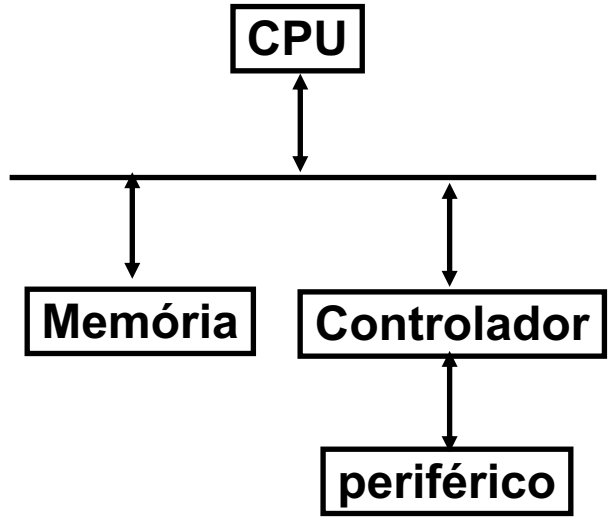
- O CPU programa o controlador de DMA com a transferência pretendida:
  - Leitura da memória ou escrita em memória
  - Endereço do bloco de memória dos dados a receber/enviar
  - Número de bytes a transferir
- O CPU fica livre para outro processamento
- O controlador de DMA trata da transferência com o controlador do periférico
- O controlador de DMA compete com o CPU pelo uso do BUS para aceder à memória
- O controlador de DMA envia uma interrupção quando termina toda a transferência (ou em caso de erro)

Transferência de dados baseada em interrupções sem DMA

- Os processos são parados durante a transferência (o CPU executa código para a transferência)

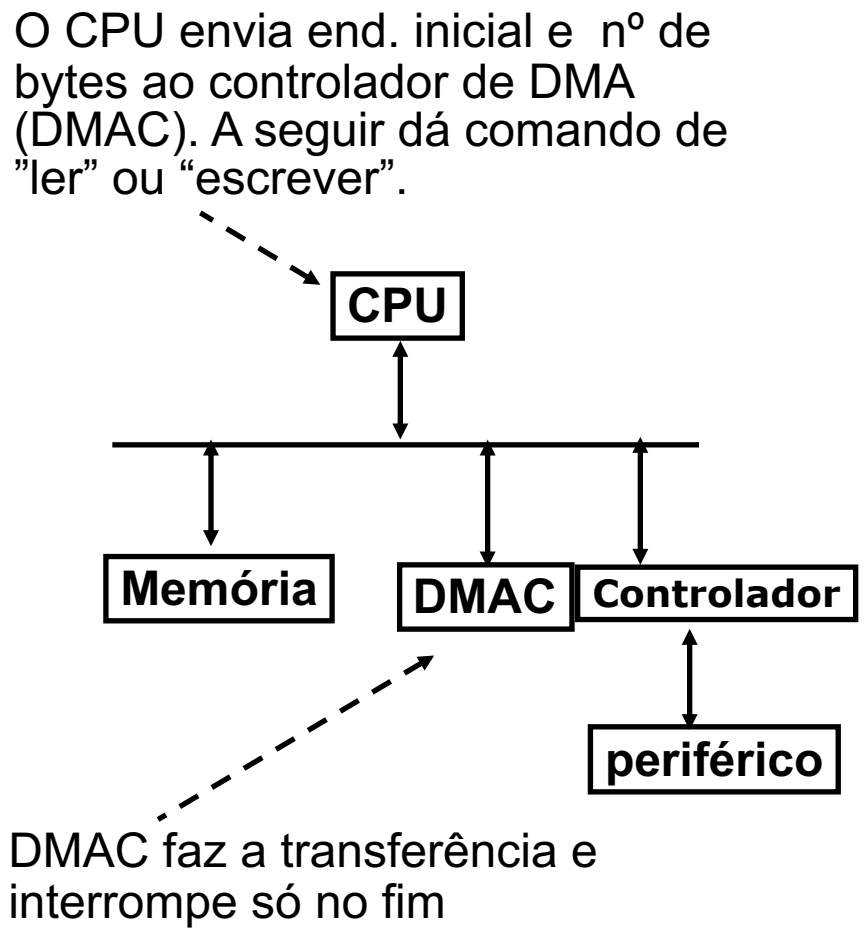
- Se CPU a 1GHz e rotina de tratamento com 100 instruções:
  - 1 GHz → 10^9 ciclos por segundo
    → 10^9 instruções por segundo
  - 1000 interrupções ➔ 1000 x a rotina de serviço custa ao CPU : 1 000x100 insts ou 100 000 Hz

- No melhor dos casos transfere $10^9/100 = 10^7$ = 10 Mbytes/s (mas depende também da velocidade do bus, periférico e memória)



Direct Memory Access

- Suponhamos que a preparação do DMA e o fim também "custa" 100 instruções cada.
  - A transferência de 1000bytes custa ao CPU: 100+100 = 200 instr. (200 Hz)
- O CPU está livre para executar os programas.
- A taxa máxima de transferência depende do bus, memória, do controlador de DMA e do periférico.



Periféricos orientados ao bloco

- Os controladores organizados em blocos (p.e. para discos) incluem o controlador de DMA
  - O software (CPU) só está envolvido para dar os comandos da transferência pretendida
    - Depois fica livre enquanto a transferência decorre
  - A leitura de 1 bloco decorre diretamente do periférico para memória
  - A escrita de 1 bloco decorre diretamente de memória para o periférico
  - Pode enviar interrupção para notificar o fim e poder ser efetuado outro pedido ao periférico