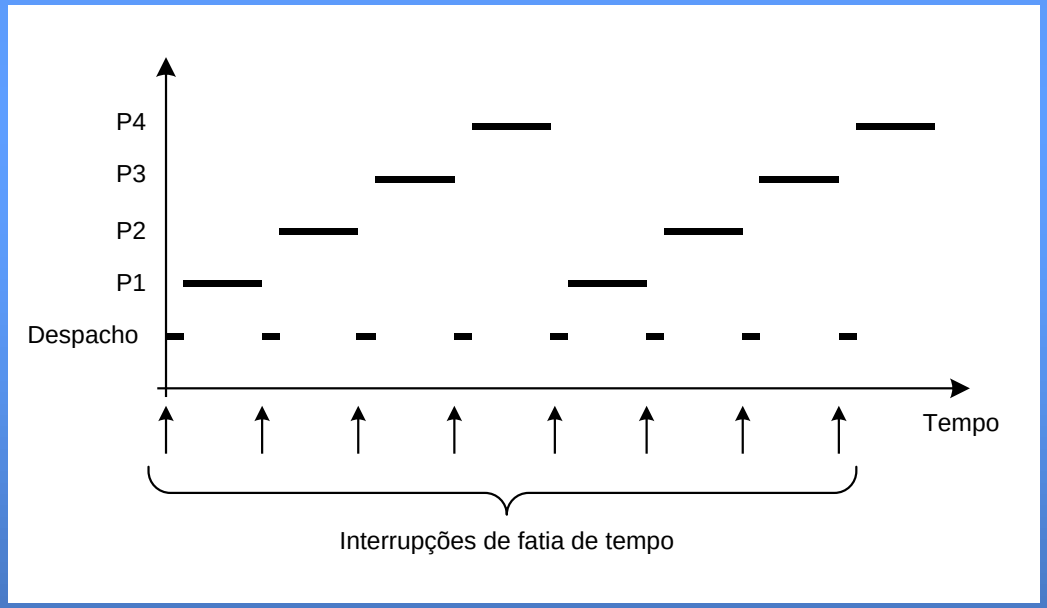
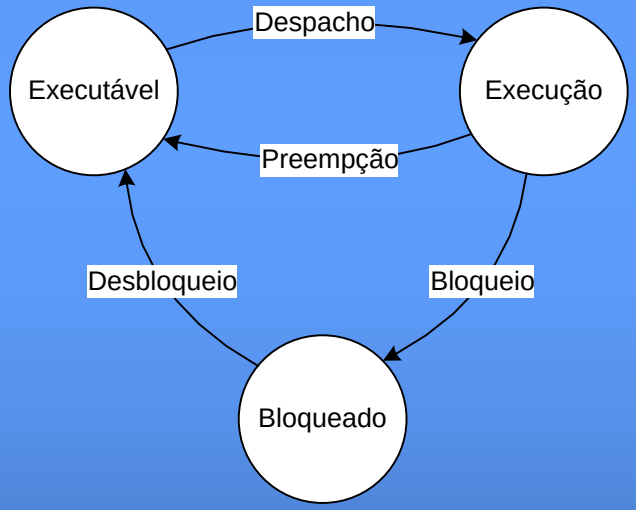
# Introdução à arquitetura de sistema

- Programação cooperativa
- Exceções
- Interrupções
- Periféricos
- Microcontroladores
- Desempenho

# Processos com multiprogramação

- Processos – forma de programar várias atividades de forma independente
- Mudança de processos por meio de interrupção (assíncrona)



# Processos cooperativos

- Mudança de processo por iniciativa do próprio processo
- Implementação de um processo que precise de manter estado entre invocações sucessivas:

```
function ProcessoX () {
   switch estado {
      case 1:
         . . .                  /* instruções deste estado */ ;
         estado = ...;          /* indica qual o estado seguinte */
         break;                 /* sai do processo */
      case 2:
         . . .                  /* instruções deste estado */ ;
         estado = ...;          /* indica qual o estado seguinte */
         break;                 /* sai do processo */
      . . .                     /* cláusulas dos outros estados */
   }
   return;
 }
```

# Espera não bloqueante

- Não se deve fazer:

```
; rotina que implementa o processo 1
proc1:    Lê posição de memória
          CMP valor pretendido      ; vê se valor é o esperado
          JNZ  proc1                ; se ainda não é, vai tentar de novo
          . . .                     ; a fazer caso o valor seja o esperado
          RET                       ; acabou, regressa
```

- O que se deve fazer:

```
; rotina que implementa o processo 1
proc1:    Lê posição de memória
          CMP valor pretendido      ; vê se valor é o esperado
          JNZ  fim                  ; se ainda não é, vai tentar de novo
          . . .                     ; a fazer caso o valor seja o esperado
fim:      RET                       ; acabou, regressa. Há de voltar na próxima
                                    ; iteração do ciclo
```

- A espera deve ser externa à rotina, e não interna

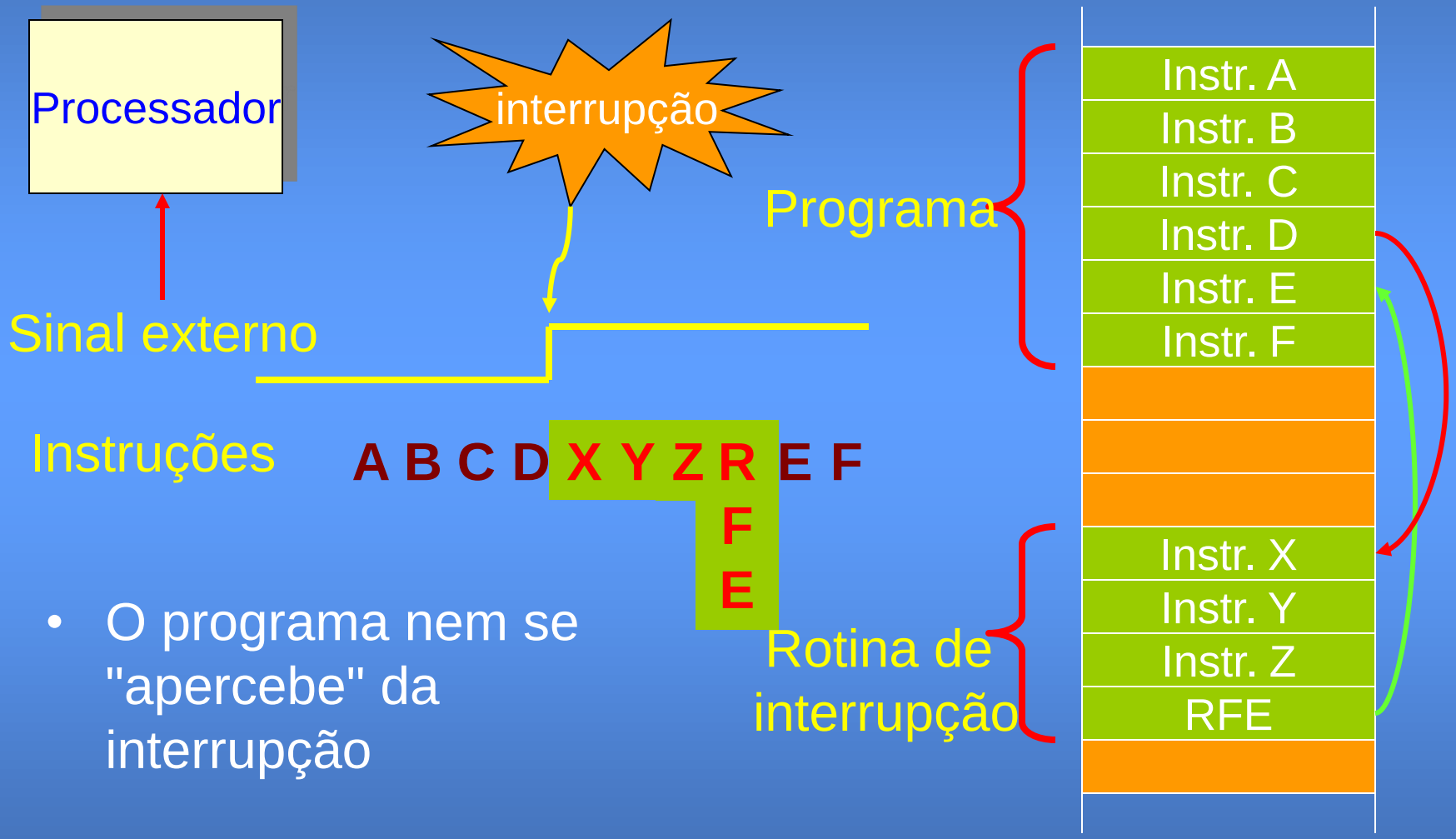
# Estrutura em *assembly*

```
; rotina que implementa um processo (proc1)
proc1:      MOV   R0, estado
            MOV   R1, [R0]          ; Lê variável com o estado do processo

estado0:    CMP   R1, 0             ; vê se é o estado 0
            JNZ   estado1
            ...                     ; faz o processamento do estado 0
            MOV   R1, novo valor    ; indica qual o próximo estado
            JMP   fim               ; acabou execução do estado 0

estado1:    CMP   R1, 1             ; vê se está no estado 1
            JNZ   estado2
            ...                     ; faz o processamento do estado 1
            MOV   R1, novo valor    ; indica qual o próximo estado
            JMP   fim               ; acabou execução do estado 1

estado2:    CMP   R1, 2             ; vê se está no estado 2
            JNZ   estado3
            ...                     ; faz o processamento do estado 2
            MOV   R1, novo valor    ; indica qual o próximo estado
            JMP   fim               ; acabou execução do estado 2

estadoN: . . .                      ; etc. Outros estados
            ...
            JMP   fim               ; acabou execução do estado N

fim:        MOV   [R0], R1          ; atualiza estado do processo
            RET                     ; sai, para que os outros processos corram
```

# Exceções vs Interrupções

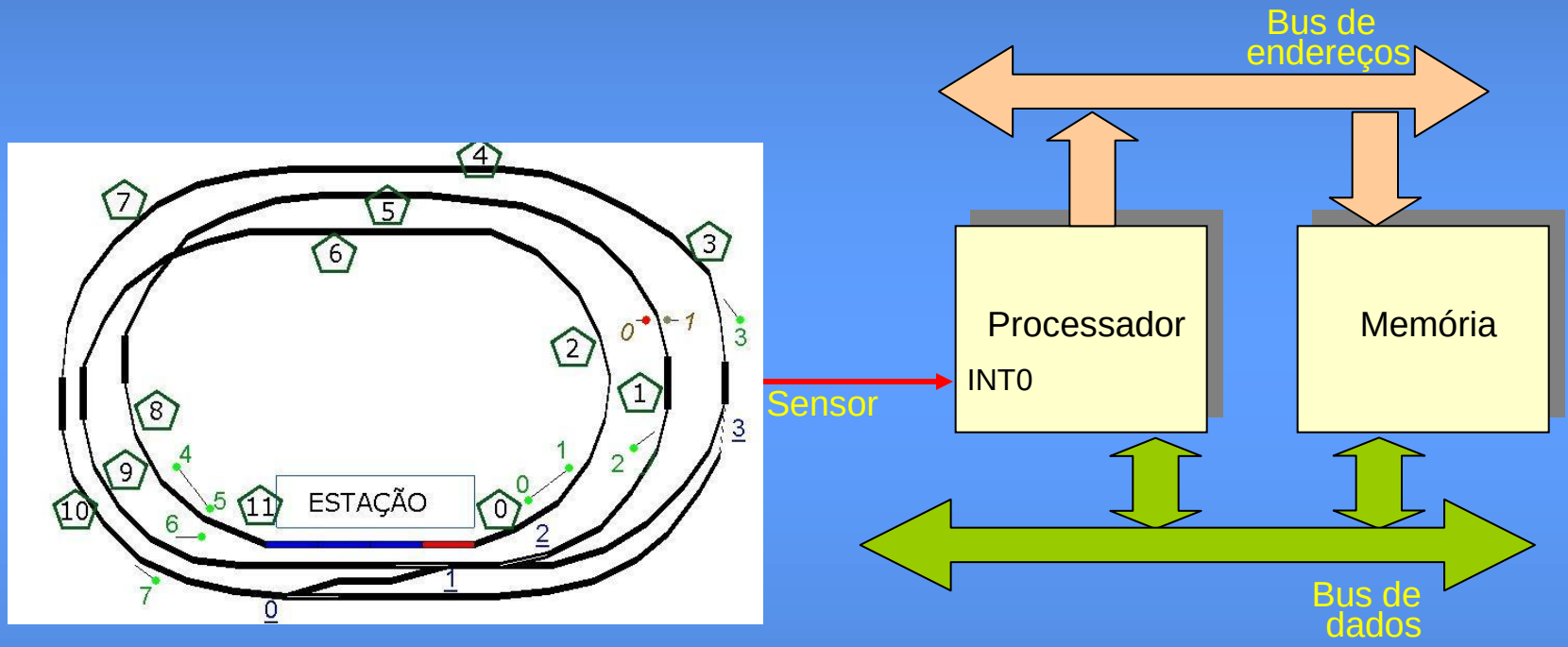
- Exceção - qualquer evento que pode ocorrer mas de que o programa que está a correr nesse instante não está à espera.
- Solução: interromper o programa normal e invocar uma rotina de tratamento da exceção.
- As exceções podem ser causadas:
  - Pelo próprio programa (divisão por zero, falta de página, acesso à memória desalinhado, etc). São síncronas em relação ao programa;
  - Pela ativação de um pino externo (interrupções). São assíncronas face ao programa, sendo imprevisível a instrução em que ocorrem.

# Mecanismo das interrupções

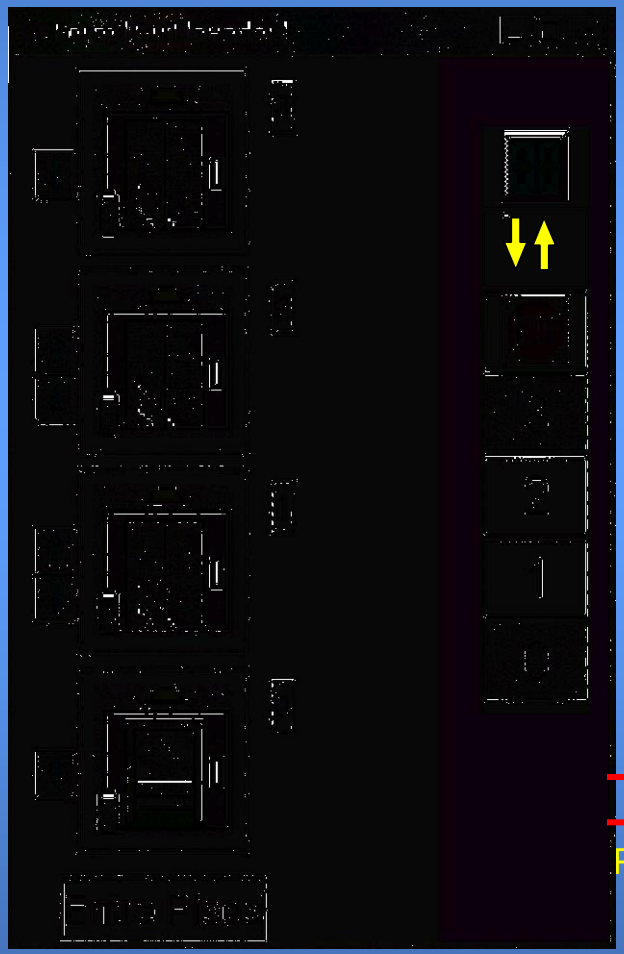


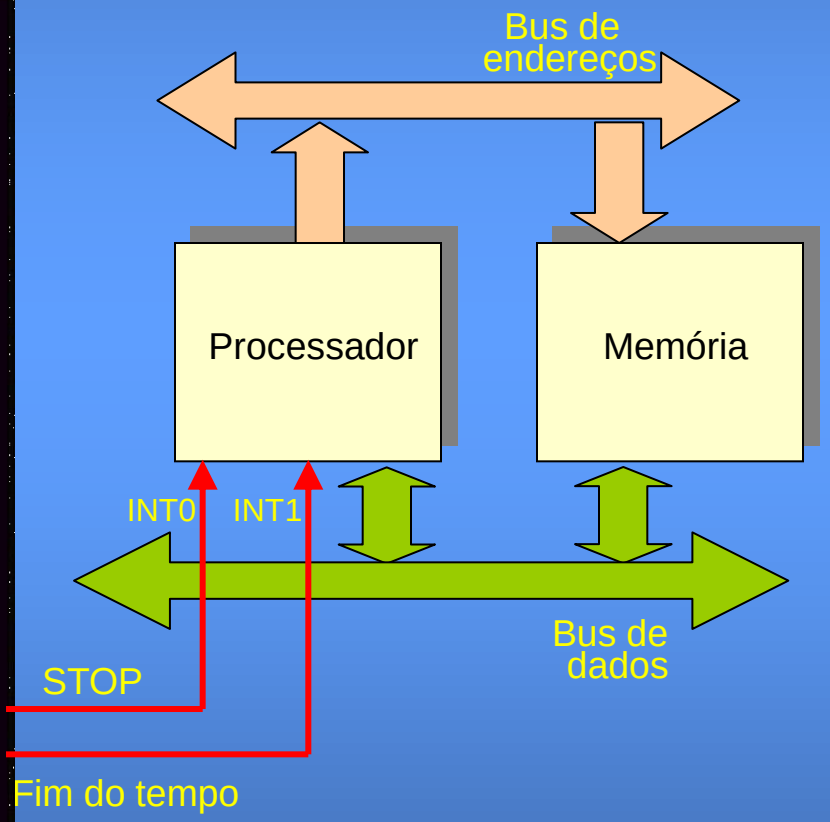

- O programa nem se "apercebe" da interrupção

# Interrupções



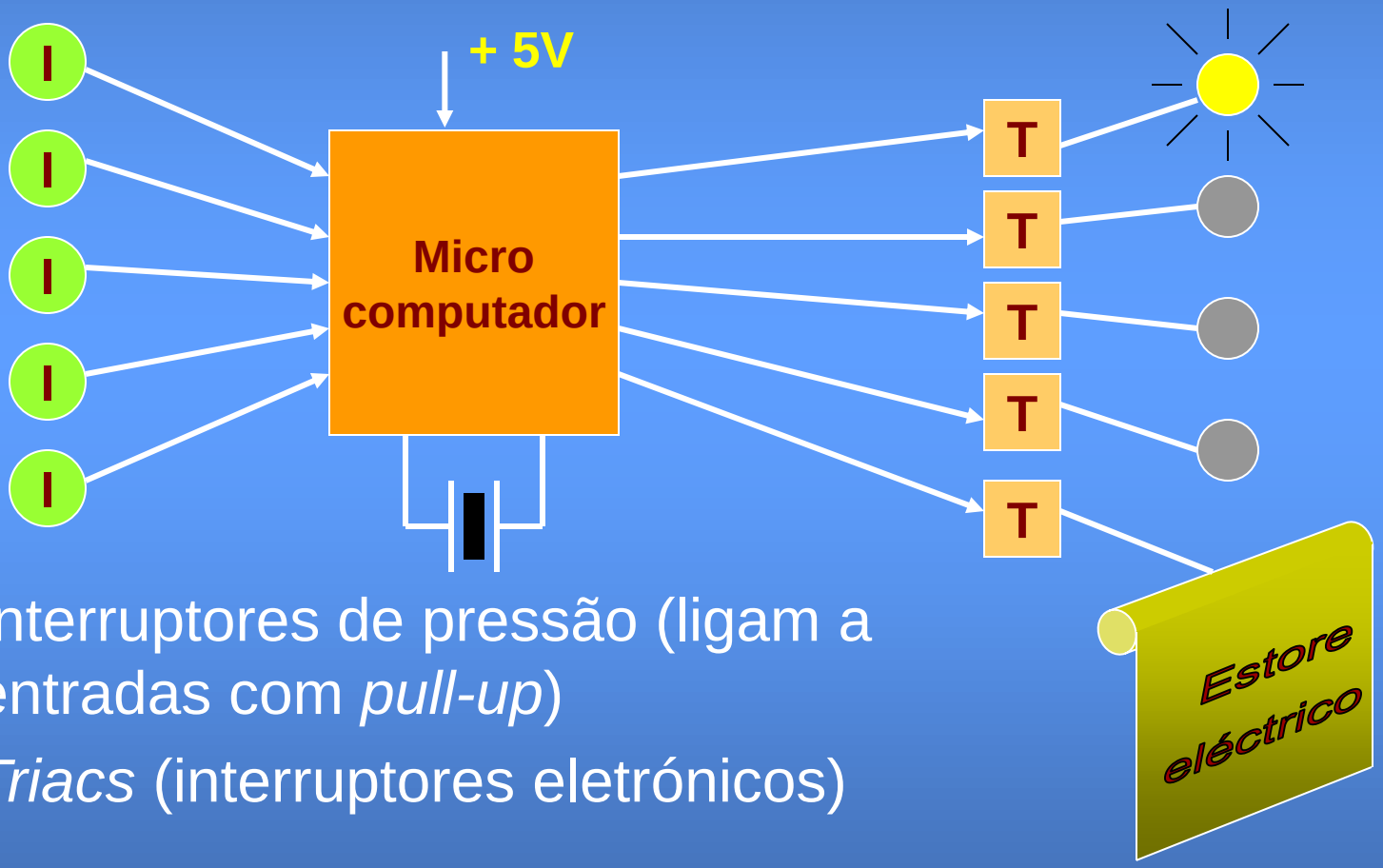
# Interrupções





# Exemplo de aplicação

- Aplicação: controlo de uma casa (domótica)

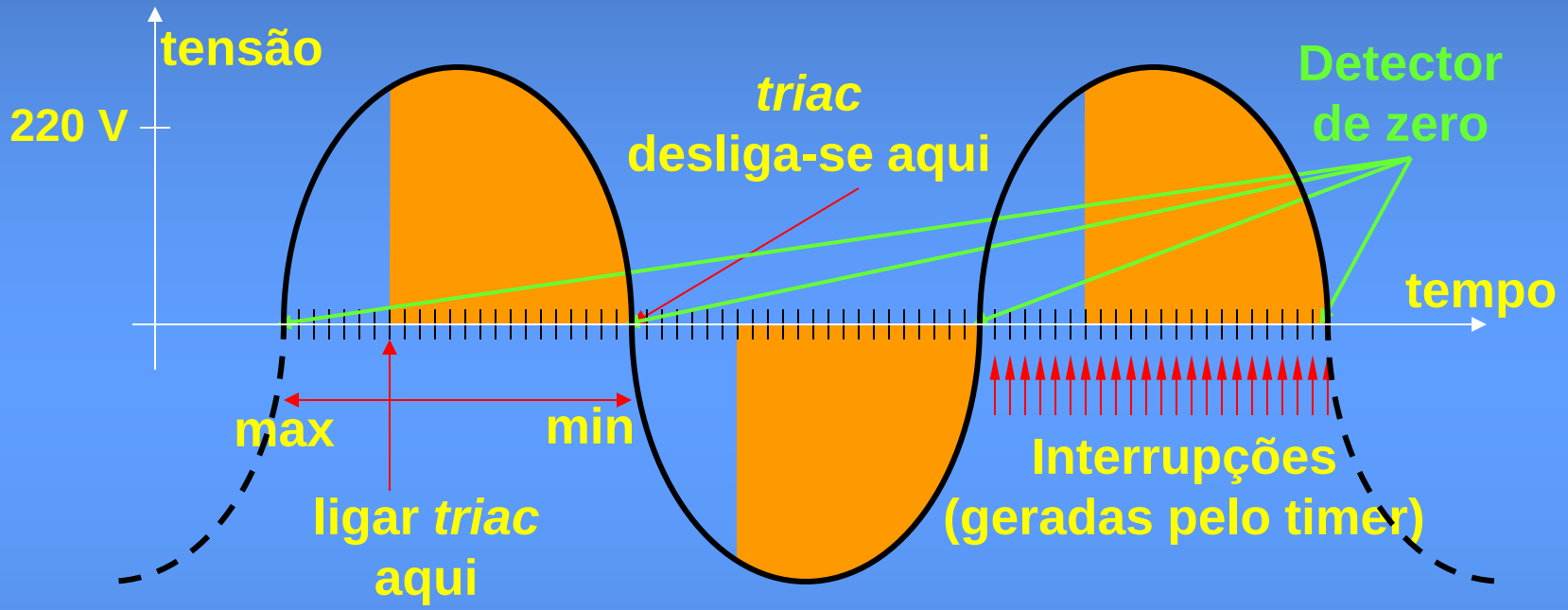



I – Interruptores de pressão (ligam a entradas com *pull-up*)

T – *Triacs* (interruptores eletrónicos)

# Controlo de tempo real

- Controlo da intensidade luminosa das lâmpadas:




- Subir ou descer a intensidade luminosa é mudar o ponto de ativação do *triac* (número de interrupções ocorridas desde a deteção de zero da sinusoide)

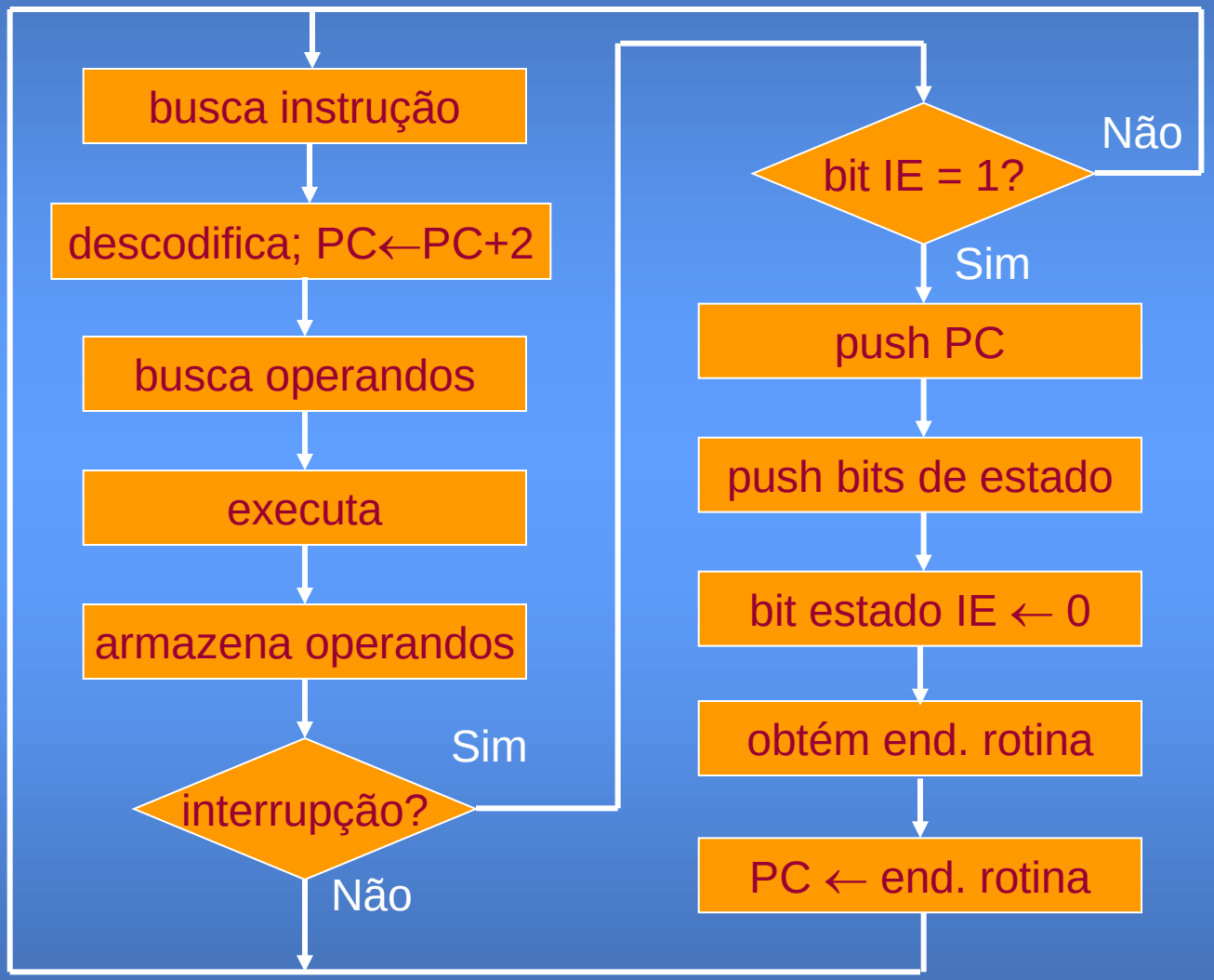
# Rotinas de interrupção

- Invocáveis em qualquer ponto do programa quando um sinal externo (programável):
  - tem um dado valor (nível), ou
  - muda de valor (flanco)
- Não podem alterar rigorosamente nada do estado do processador (nem mesmo os bits de estado).
- A invocação da rotina de interrupção guarda automaticamente na pilha:
  - Endereço de retorno (endereço da próxima instrução na altura em que a interrupção aconteceu)
  - Registo dos bits de estado
- A instrução RFE (*Return From Exception*) faz dois POPs pela ordem inversa (repondo os bits de estado e fazendo o retorno).
- RET e RFE não são intermutáveis!
- Se a rotina de interrupção alterar qualquer registo, tem de o guardar primeiro na pilha e restaurá-lo antes do RFE.

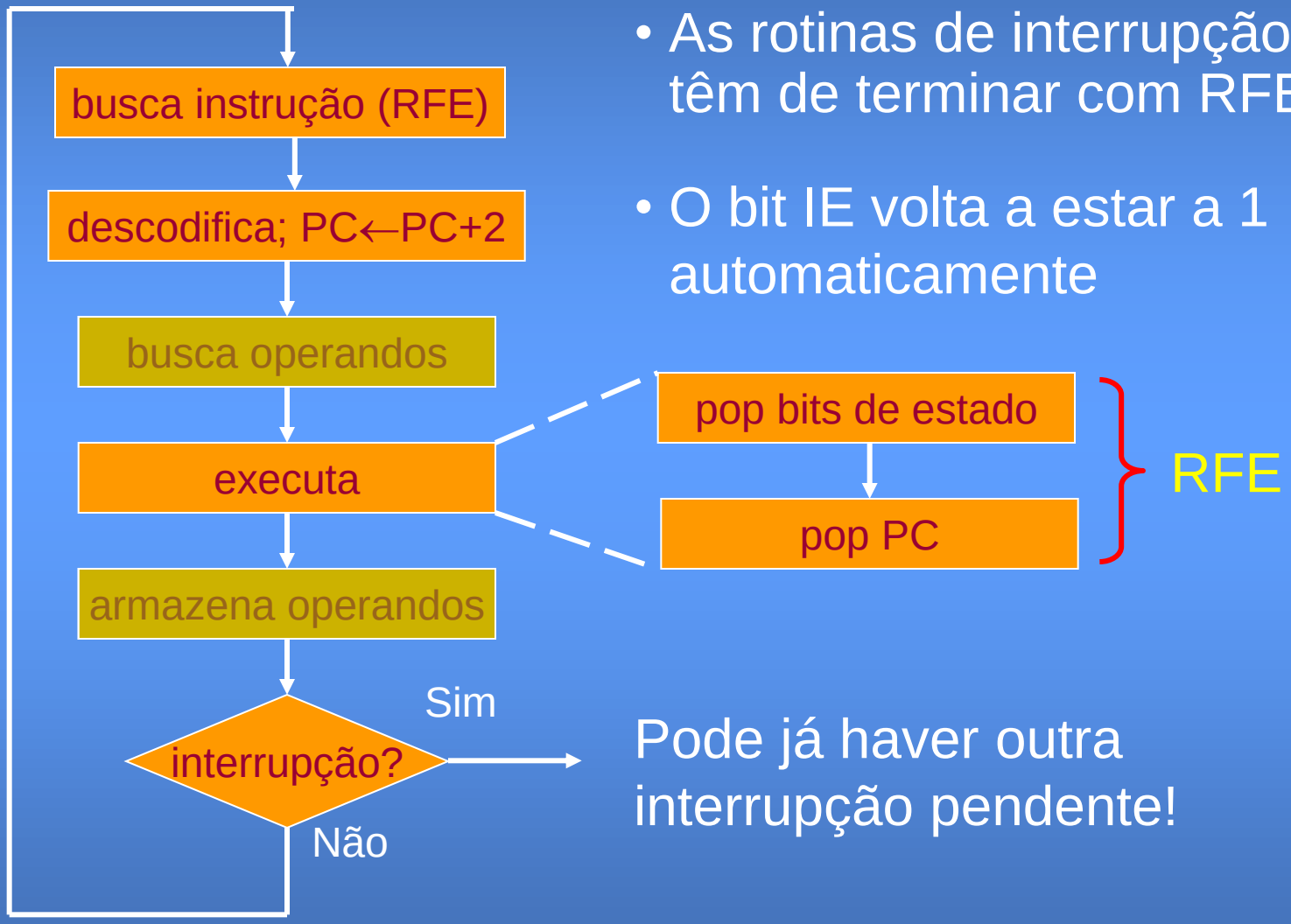
# Bit de estado IE

- Um programa pode estar a executar operações críticas que não devem ser interrompidas.
- Por isso, existe um bit de estado (IE) que quando está a 0 impede o processador de atender interrupções.
- Para manipular este bit existem duas instruções:
  - EI (*Enable Interrupts*). Faz IE ← 1
  - DI (*Disable Interrupts*). Faz IE ← 0
- A própria rotina de interrupção pode ser crítica e não permitir interrupções a ela própria. Por isso, IE é colocado a 0 automaticamente quando uma interrupção é atendida.

# Tratamento de interrupções



# Retorno de interrupções (RFE)




- As rotinas de interrupção têm de terminar com RFE
- O bit IE volta a estar a 1 automaticamente

Pode já haver outra interrupção pendente!

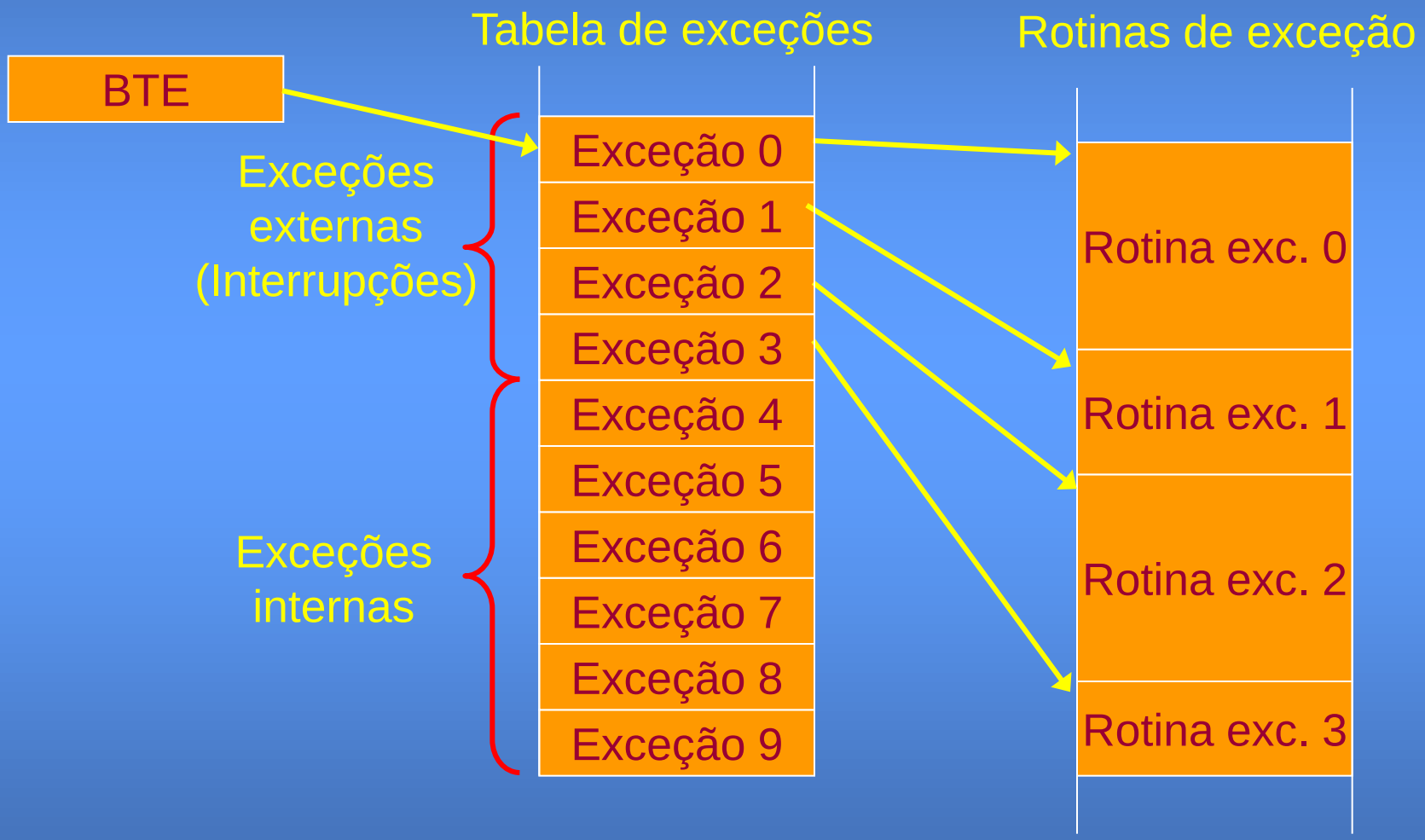
# Rotina de interrupção típica

```
rotina_int:                 ; DI automático, bit IE fica a 0
                            ; (não responde às interrupções)
        push    registos    ; guarda registos que a  rotina vá usar
        instruções críticas ; sequência de instruções que não pode
                            ; ser interrompida
        EI                  ; permite interrupções
        instruções não críticas ; sequência de instruções que pode
                            ; ser interrompida
        pop     registos    ; restaura registos (pela ordem inversa)
        RFE                 ; retorna da rotina (EI automático)
```

# Exemplo: aquisição de dados

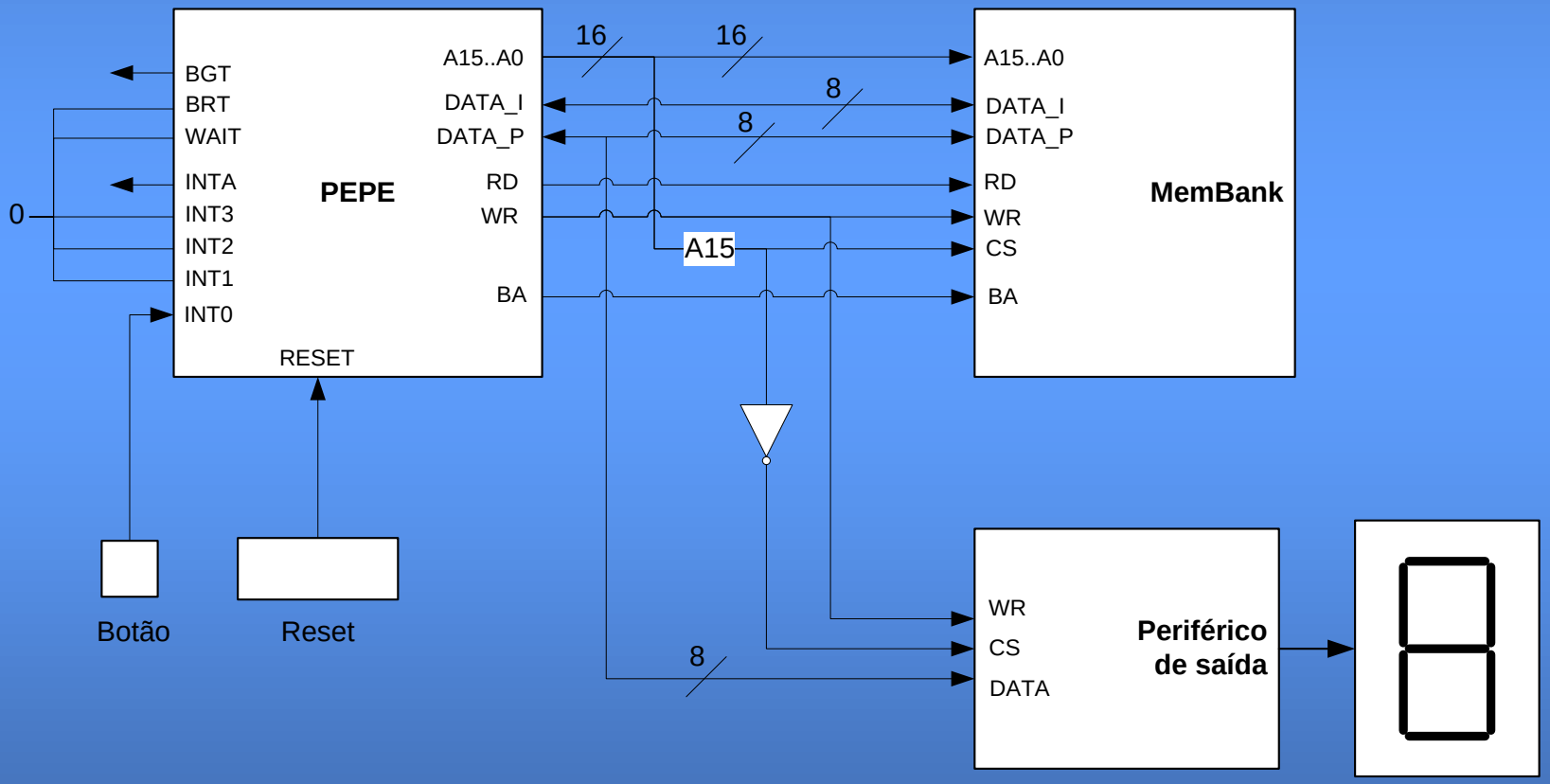
```
LeDados: ; invocada por interrupção quando há valor disponível no periférico
         ; R3 contém endereço do periférico
         ; R4 contém o endereço de base da zona de dados na memória
         ; R5 contém o índice da última posição ocupada na zona de dados
         ; (inicializado com o TAMANHO da zona de dados em palavras)
         PUSH    R1                ; guarda registos que serão alterados
         PUSH    R2
         MOV     R1, [R3]          ; lê valor (16 bits) do periférico
         SUB     R5, 2             ; índice da palavra onde guardar o valor
         MOV     [R4+R5], R1       ; guarda valor na zona de dados
         JNZ     continua          ; ainda não encheu a zona de dados
         MOV     R5, TAMANHO       ; reinicializa o índice (buffer circular)
continua: EI                       ; passa a permitir novas interrupções
         . . .                     ; outro processamento que possa...
         . . .                     ; ...ser interrompido
         POP     R2                ; restaura registos (ordem inversa)
         POP     R1
         RFE                       ; retorna da interrupção
```

# Tabela de exceções



# Exemplo

- Ao carregar no botão, o mostrador de 7 segmentos sobe uma unidade (por software).



# Rotina de interrupção (flanco)

```
PLACE       2000H
tab:  WORD  rot0          ; tabela de interrupções

PLACE       0
      MOV   BTE, tab      ; incializa BTE
      MOV   SP, 1000H     ; incializa SP
      MOV   R0, 0         ; inicializa contador
      EI0                 ; permite interrupções
      EI
fim:  JMP   fim           ; fica à espera


rot0:                     ; rotina de interrupção
      PUSH  R1            ; guarda R1
      MOV   R1, 8000H     ; endereço do periférico
      ADD   R0, 1         ; incrementa contador
      MOVB  [R1], R0      ; atualiza mostrador
      POP   R1            ; repõe R1
      RFE                 ; regressa
```

# Rotina de interrupção (nível)

```
PLACE       2000H
tab:  WORD  rot0          ; tabela de interrupções

PLACE       0
      MOV   BTE, tab      ; incializa BTE
      MOV   SP, 1000H     ; incializa SP
      MOV   R0, 2
      MOV   RCN, R0       ; interrupção 0 sensível ao nível
      MOV   R0, 0         ; inicializa contador
      EI0                 ; permite interrupções
      EI
fim:  JMP   fim           ; fica à espera


rot0:                     ; rotina de interrupção
      PUSH  R1            ; guarda R1
      MOV   R1, 8000H     ; endereço do periférico
      ADD   R0, 1         ; incrementa contador
      MOVB  [R1], R0      ; atualiza mostrador
      POP   R1            ; repõe R1
      RFE                 ; regressa
```

# Exercícios

1. As interrupções são desativadas automaticamente sempre que um processador atende uma interrupção. Porquê?
2. Explique o que sucede se invocar uma rotina de interrupção com CALL.
3. Os chamados programas de “hard real-time”, em que os tempos de execução são críticos e têm de ser escrupulosamente respeitados, não usam interrupções. Se houver necessidade, o processador testa e processa todos os sinais explicitamente e quando entender. Explique a razão de eliminar algo tão útil como as interrupções.

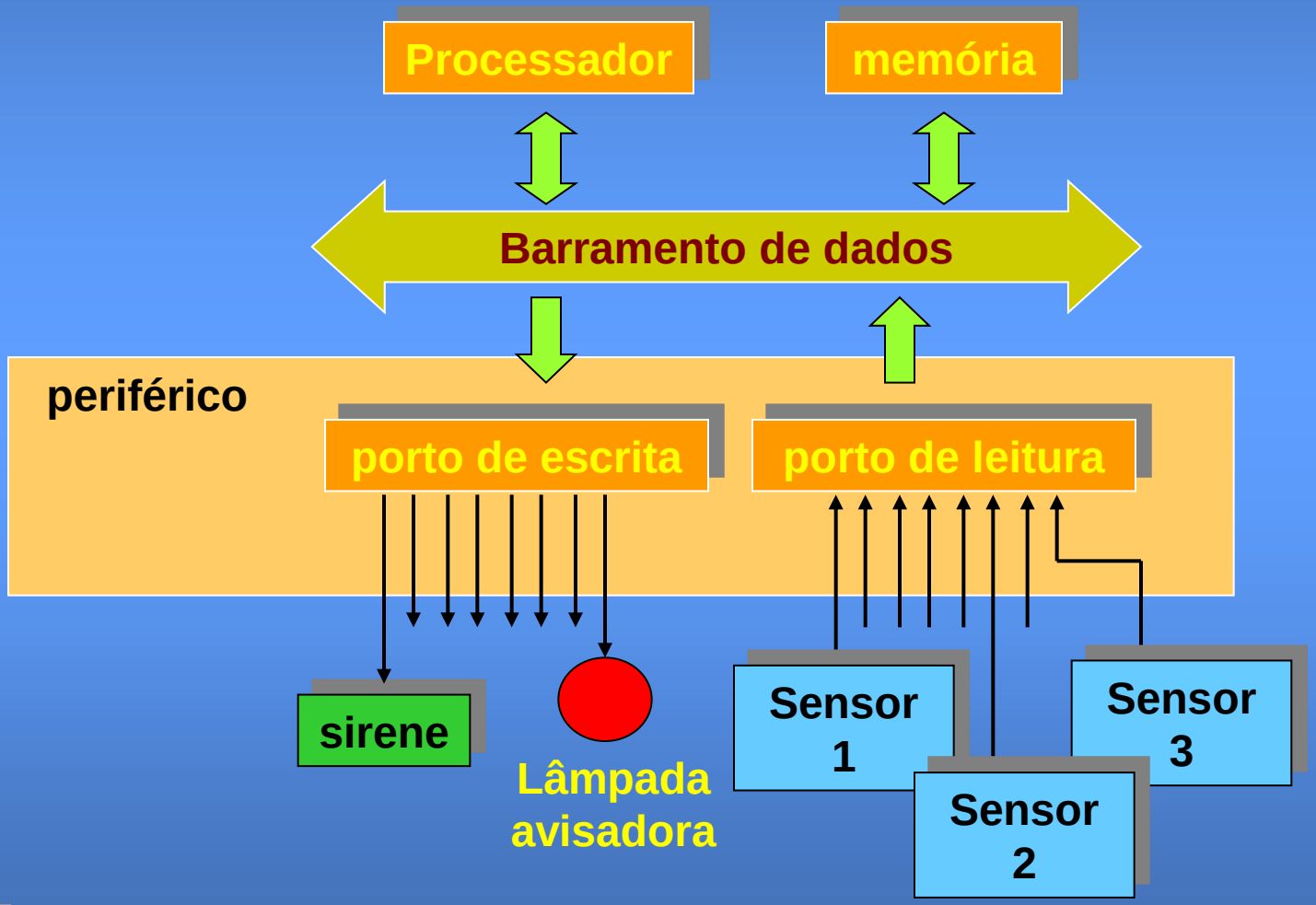
# Exercícios

4. Suponha que a rotina de interrupção A demora 1 milissegundo a executar e tem maior prioridade que a rotina de interrupção B, que demora 10 milissegundos a executar. Nem A nem B voltam a permitir as interrupções explicitamente (com a instrução EI). Suponha que o hardware externo pede 100 interrupções A e 50 interrupções B por segundo.
   a) Quanto tempo tem o processador por segundo para correr o programa principal?
   b) Supondo que no programa principal as interrupções estão sempre permitidas e que o tempo máximo de execução de uma instrução do processador é de 10 microssegundos, indique qual o tempo máximo de espera antes que o processador atenda uma interrupção do tipo A e do tipo B, supondo que o processador acabou de começar a executar (i) uma instrução no programa principal, (ii) a rotina de interrupção A e (iii) a rotina de interrupção B.

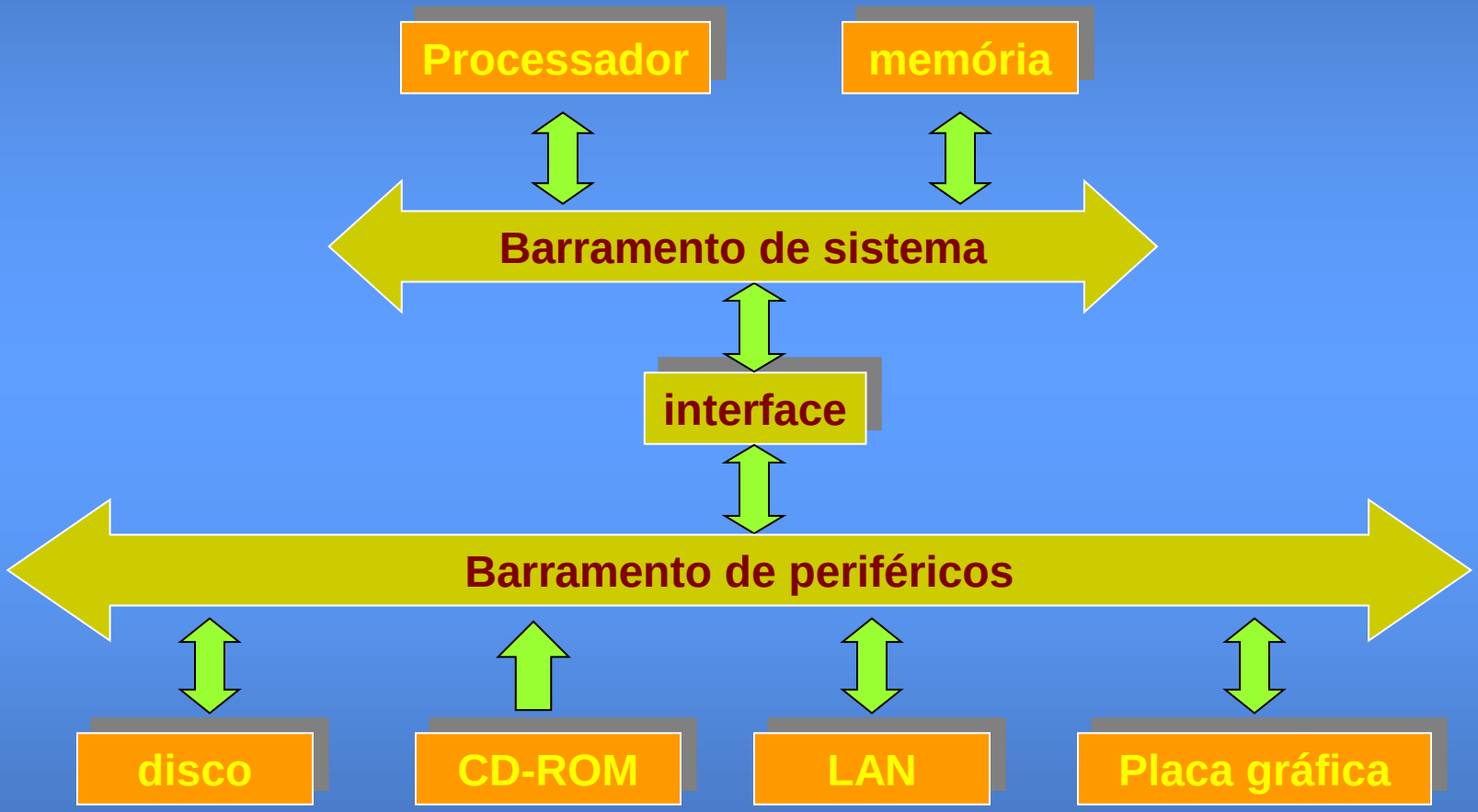
# Exercícios

4. (continuação)
   c) Suponha agora que a cadência de pedidos de interrupções B começa a aumentar. Explique o que se passa em termos do tempo do processador dedicado ao programa principal e às rotinas A e B. Indique para que valores dessa cadência acontecem coisas significativas.
   d) Idem, mas voltando à cadência inicial de pedidos de interrupção B e começando agora a aumentar a cadência de pedidos de interrupção A.
5. Explique o que sucede quando uma interrupção ocorrer e na tabela de endereços das rotinas de interrupção o endereço especificado para essa interrupção for o de uma rotina normal.

# Exemplo de periférico simples



# Barramentos hierárquicos

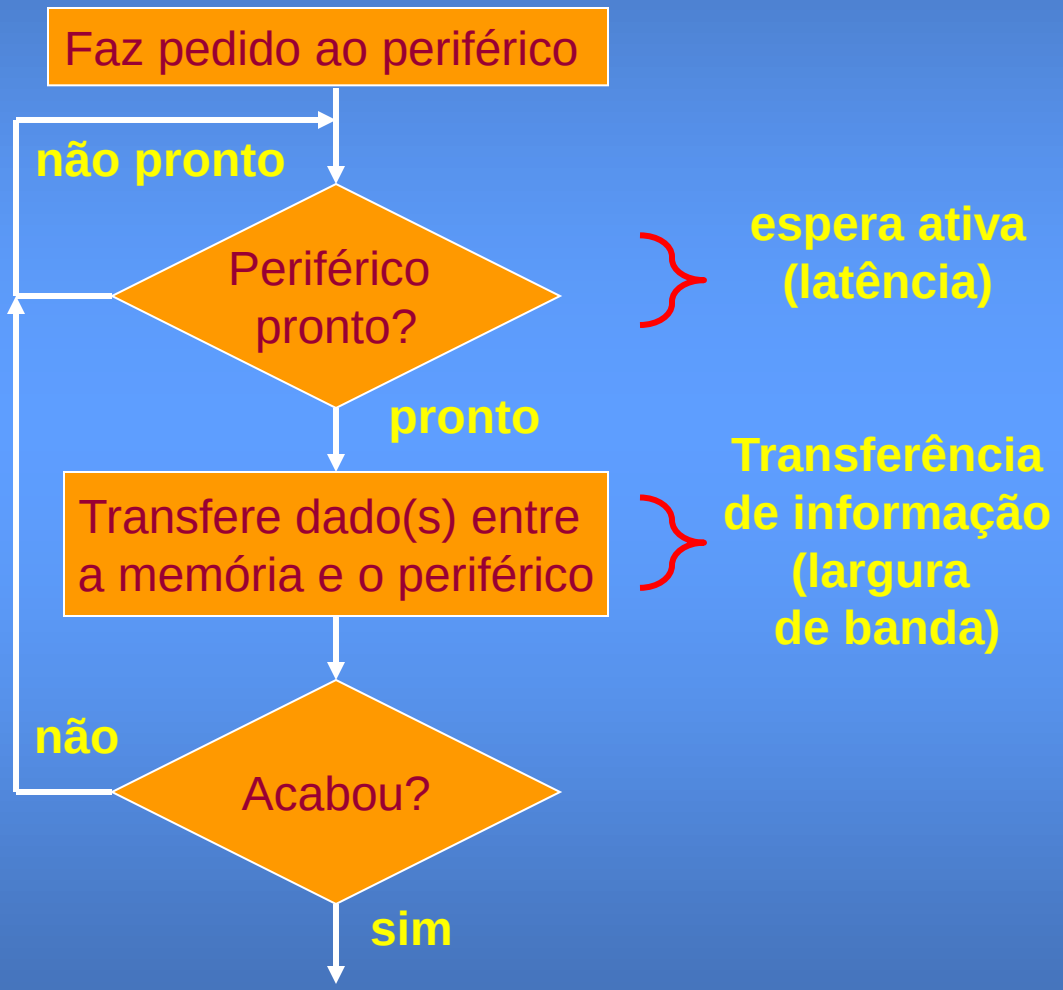


# Modos de entradas/saídas

- Modos de transferência de informação entre o processador/memória e os periféricos:
  - Sob controlo do programa (*polling*)
  - Por interrupção
  - Por acesso direto à memória (DMA – Direct Memory Access)
  - Com co-processador de entradas/saídas
- Num extremo (*polling*), o processador trata de tudo. No outro, o processador limita-se a programar o co-processador.
- Dado que as entradas/saídas são lentas, a ideia é reduzir o tempo que o processador gasta à espera dos periféricos (libertando-o para outras tarefas).

# Polling

- O programa controla tudo.
- O processador faz espera ativa contínua (senão pode perder dados) sobre periféricos lentos
- A transferência é feita por software.



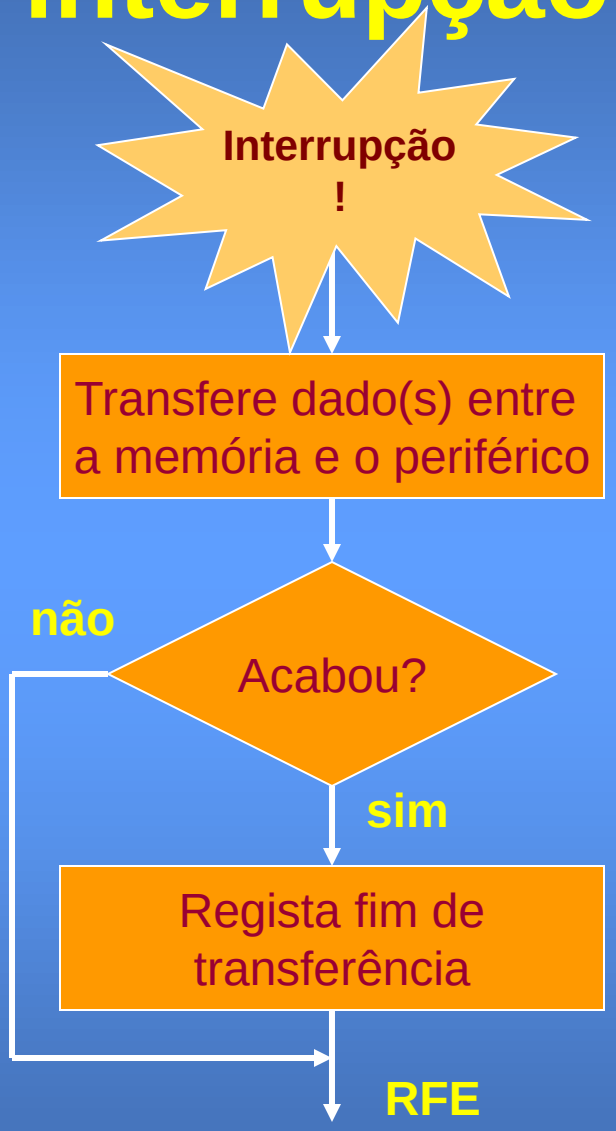
# Transferência por interrupção

Faz pedido ao periférico

vai fazer outras coisas

. . .

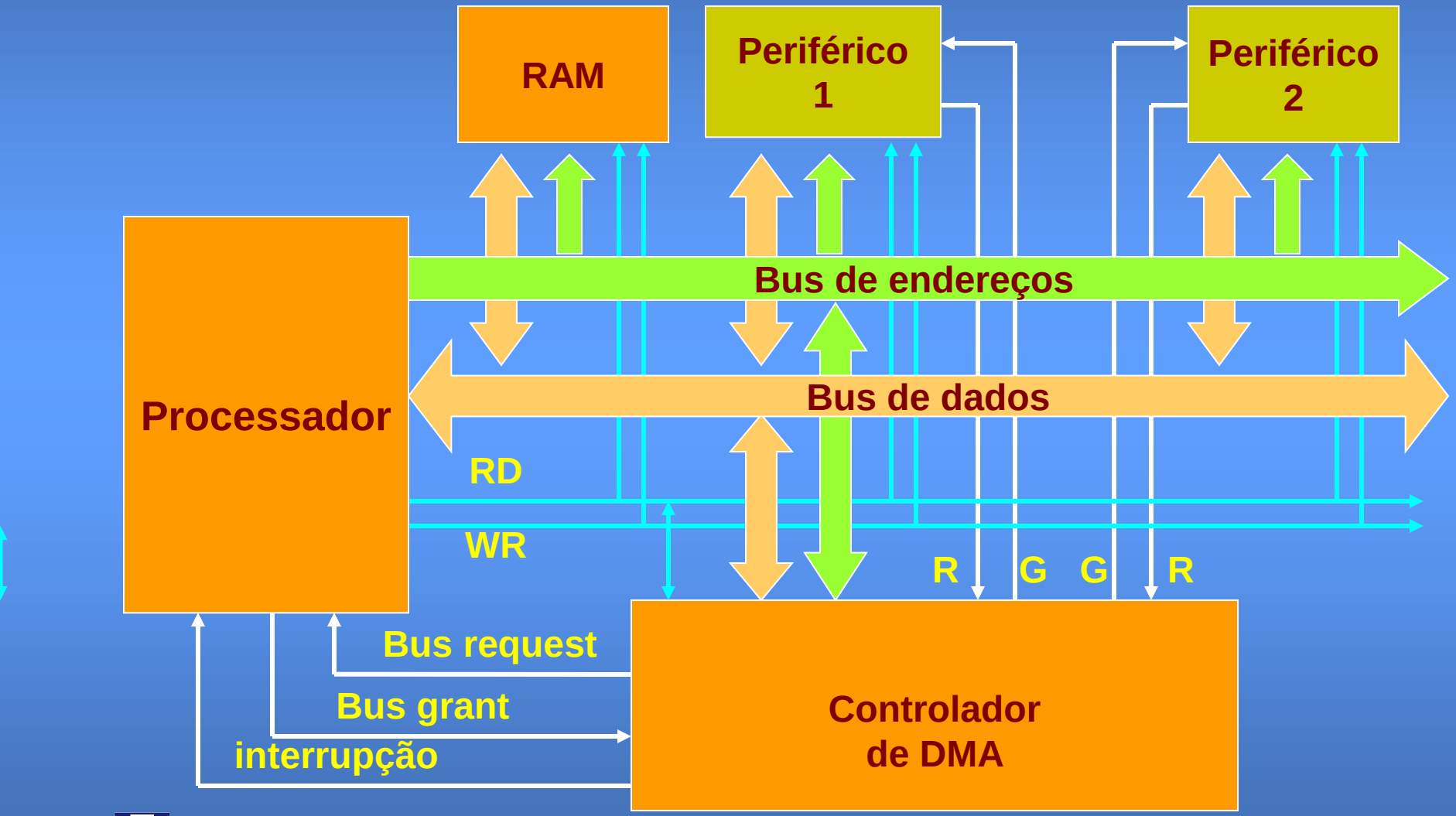
- A espera já não é ativa.
- O processador só é "incomodado" quando há coisas para fazer.
- Cada periférico tem o seu *device driver.*



# DMA (Direct Memory Access)

- A transferência de informação entre o processador/memória e os periféricos é feita em hardware por um controlador especializado.
- O processador só tem de programar o controlador de DMA, escrevendo em portos próprios do controlador (que em si também é um periférico):
  - Endereço de origem
  - Endereço de destino
  - Número de palavras a transferir
  - Qual o modo de DMA
- Durante a transferência, os endereços de origem e destino são incrementados automaticamente.

# Controlador de DMA



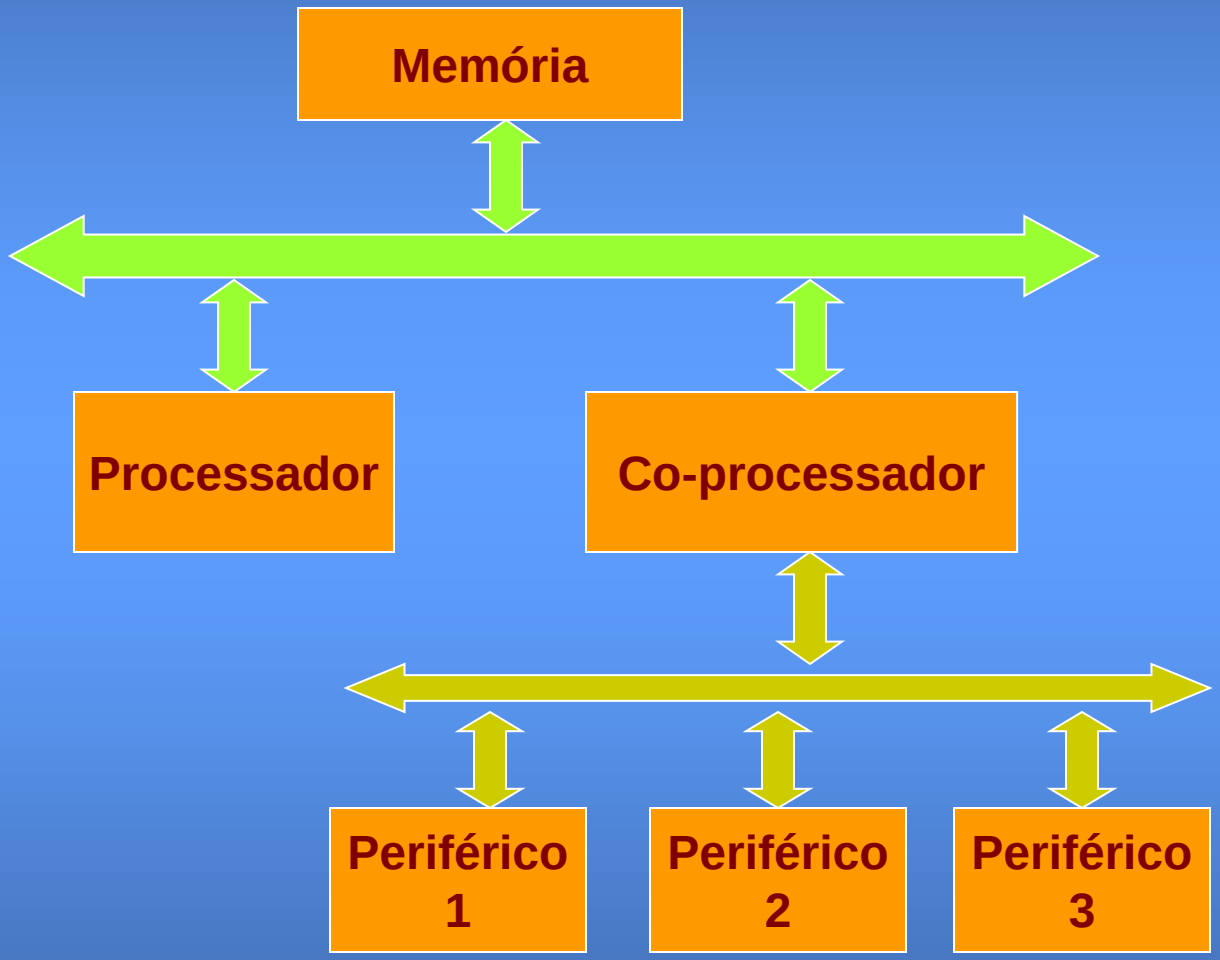
# Tipos e modos de DMA

- Tipos de DMA:
  - Simultâneo (dado é lido da fonte e escrito ao mesmo tempo no destino através do bus dados)
  - Sequencial (dado é lido primeiro para um registo interno do controlador e escrito a seguir → permite transferências memória-memória)
- Modos de DMA:
  - Palavra, ou *cycle stealing* (o controlador liberta o bus após cada palavra transferida)
  - Rajada, ou *burst* (controlador toma conta do bus até não haver mais palavras disponíveis imediatamente)
  - Bloco (controlador toma conta do bus até todos os dados terem sido transferidos, mesmo que tenha de estar à espera de dados → só para periféricos muito rápidos)

# Co-processador de entradas/saídas

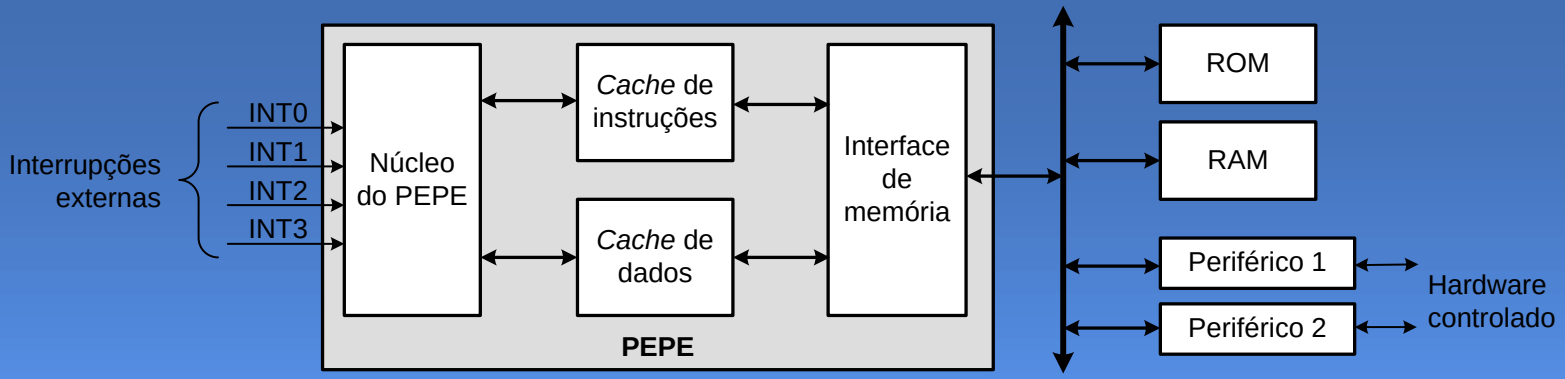
- É o modo mais flexível e poderoso de fazer entradas/saídas
- O co-processador corresponde a um controlador de DMA que pode executar um programa (em vez de apenas modos fixos), fazendo *fetch* por si próprio.
- Tem um conjunto de instruções limitado, especializado em entradas/saídas.
- O co-processador partilha a memória do processador (e compete com ele no acesso ao bus e à memória. As rotinas de entradas/saídas têm de ser feitas com cuidado).
- A comunicação processador/co-processador faz-se tipicamente por variáveis partilhadas.

# Co-processador de E/S (cont.)

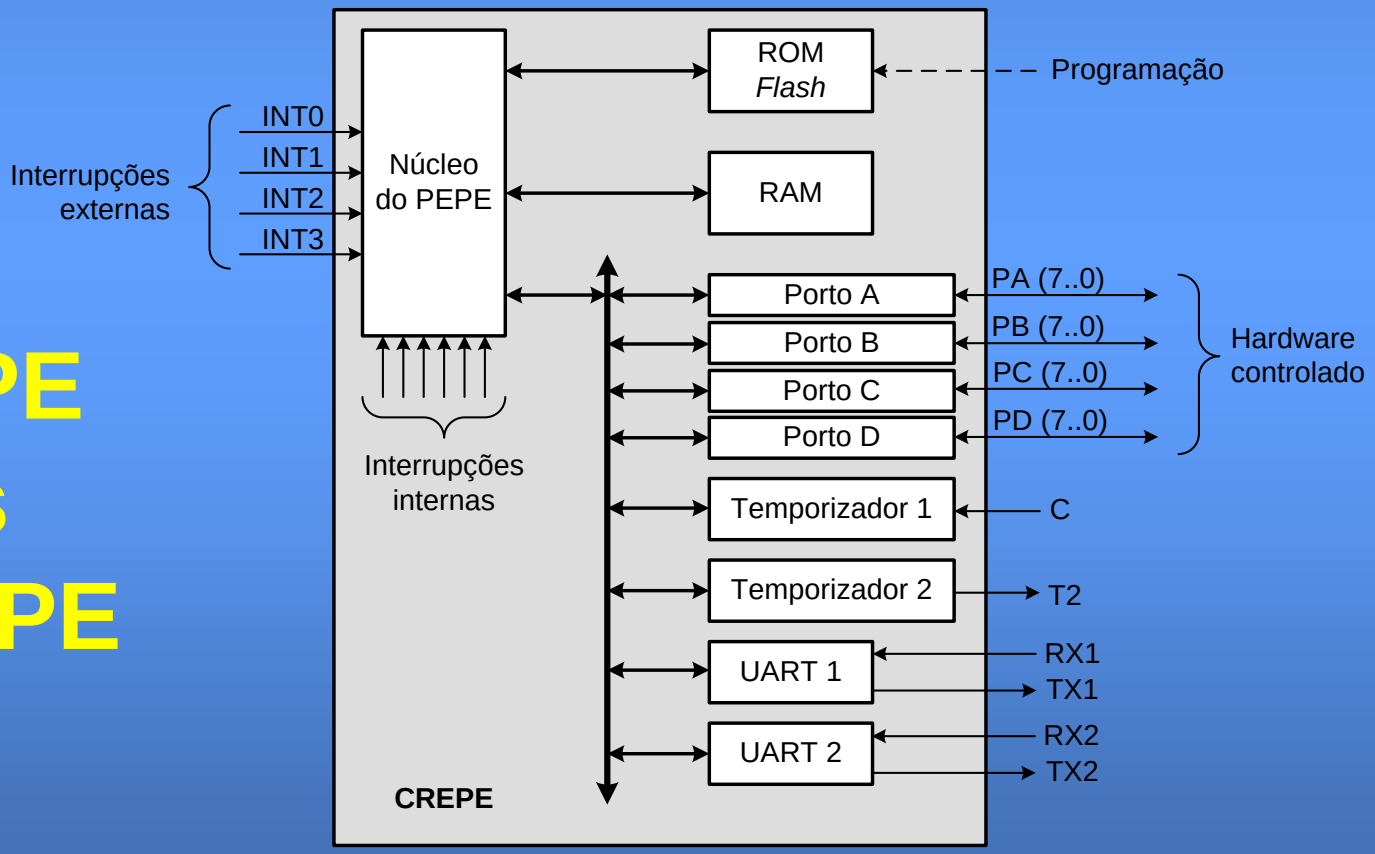


# Lidar com  vários periféricos

- Um computador tem normalmente vários periféricos e pode misturar os vários modos de transferência de dados.
- Deve-se ter em atenção:
  - A transferência sob controlo do programa (*polling*) deve ser reservada para periféricos lentos, sem temporizações críticas e com protocolos que possam ser interrompidos;
  - A transferência por interrupções é mais eficiente, mas pesada para transferência de grandes quantidades de informação (a transferência em si é feita por software);
  - A transferência por DMA (ou com co-processador) é a mais eficiente, mas o processador pode não conseguir atender interrupções durante uma transferência.



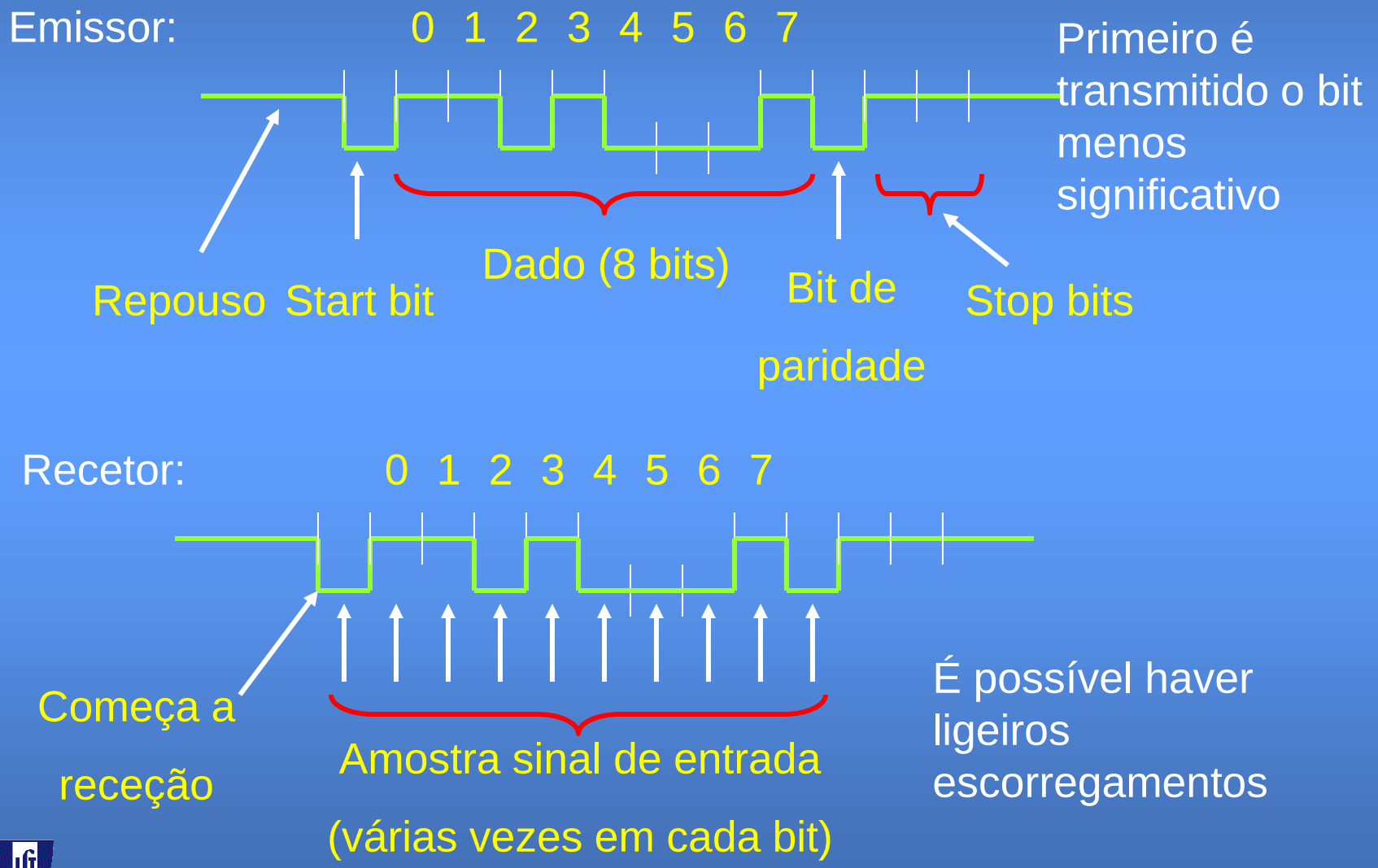

# PEPE vs CREPE



# Barramentos série assíncronos

- A comunicação é orientada ao byte, serializado
- Barramento está normalmente em repouso (1).
- Quando o emissor decide transmitir:
  - coloca a linha a 0 durante um bit (*start bit*)
  - envia os 8 bits do byte em sequência
  - envia um bit de paridade (para deteção de erros)
  - envia de 1 a 2 *stop bits* a 1 (para sincronização)
- A cadência de transmissão dos bits (*baud-rate*) tem de ser aproximadamente a mesma em todos os dispositivos no barramento (mas não tem de ser exatamente igual)
- O assincronismo deriva do tempo arbitrário entre bytes. Usa-se em aplicações de baixo ritmo de transmissão (sistemas de controlo, por exemplo)

# Comunicação série assíncrona



# Comunicação série assíncrona (cont.)

- Há diversas *baud-rates* normalizadas:
  - 110 bit/s
  - 75 bit/s e seus múltiplos: 150, 300, 600, 1200, 2400, 4800, 9600, 19200, 38400
  - 14400 bit/s e seus múltiplos: 28800, 33200, 57600
- Outros parâmetros:
  - Paridade: par, ímpar ou nenhuma
  - Stop bits: 1, 1.5 ou 2
- Existem já chips que implementam este protocolo:
  - UART (*Universal Asynchronous Receiver and Transmitter*)
  - USART (suporta também o protocolo síncrono)

# Desempenho de processadores

- A melhor forma de medir o desempenho de um processador (relativamente a outros) é medir o tempo de execução de um programa.
- Equação básica do desempenho:

$$T = \frac{N * D}{F}$$

T – tempo de duração do programa
N – número de instruções no programa
D – Duração média (em ciclos de relógio) de cada instrução
F – Frequência do relógio (ciclos/segundo)

# Os limites do desempenho

- N, D e F não são independentes:
  - Para reduzir N, cada instrução tem de fazer mais, o que pode aumentar D e reduzir F;
  - Para reduzir D, as instruções têm de ser mais simples, o que obriga a ter mais instruções para fazer o mesmo;
  - Para aumentar F (sem melhorar a tecnologia), só com uma arquitetura mais simples, o que obriga a aumentar N.
- Um processador de F = 2 GHz pode ser mais rápido do que outro de F = 2.5 GHz, se tiver um menor valor de D ou de N.
- Os processadores têm evoluído por:
  - melhor tecnologia (F mais elevado);
  - melhor arquitetura (menor valor de D);
  - melhores compiladores (menor valor de N).

$$T = \frac{N * D}{F}$$

# Avaliação do desempenho

- Problema típico: comparar o desempenho de dois ou mais computadores.
- Comparar os fatores individuais não faz sentido (porque são dependentes uns dos outros).
- Métrica simples: MIPS (Mega Instructions Per Second). Ou seja, o fator F/D não chega. O valor de N pode ser diferente.
- Fabricantes divulgam normalmente o valor máximo do MIPS e não médio (porque depende do peso relativo da ocorrência das várias instruções)

- Mas… um computador não é apenas o processador!

$$T = \frac{N * D}{F}$$

# O computador como um todo: *benchmarks*

- Um *Ferrari* numa auto-estrada urbana à hora de ponta não consegue andar mais depressa do que o mais pequeno utilitário.
- Ao comparar os dois carros, não interessa apenas medir a rotação máxima ou a potência do motor. Tem de se analisar o resultado global da sua utilização.
- Em computadores: em vez de MIPS, usam-se *benchmarks*, que são programas que exercitam os vários aspetos de um computador (processador, memória e periféricos).
- Valor do *benchmark*: número de vezes/segundo que o *benchmark* executa.

# As limitações dos periféricos

- Taxas de transferência típicas:
  - Teclado (depende do operador...) – 10 bytes/seg
  - LAN, 100 Mbits/seg – 12.5 Mbytes/seg
  - Disco – 40 Mbytes/seg
  - Bus de dados a 200 MHz (64 bits) – 1600 Mbytes/seg
  - Registos internos a 2 GHz (64 bits) – 16000 Mbytes/seg
- Um processador com o dobro do relógio não corre necessariamente programas em metade do tempo!

  Tempo total = tempo execução em memória + tempo periféricos
- Se o tempo gasto à espera dos periféricos for de 50%, duplicar a velocidade do processador apenas reduz o tempo total em 25%

# A lei de Amdahl

- Assumindo que se melhora um fator que afeta apenas parte do tempo de execução:

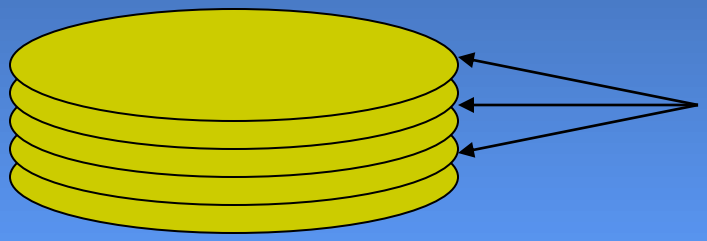
$$Tempo\_melhorado = \frac{Tempo\_parte\_afectada}{n^o\_vezes\_mais\_rápido} + Tempo\_parte\_não\_afectada$$

- Mesmo que se melhore um dos fatores (velocidade do processador, p. ex.), os restantes podem limitar severamente a melhoria global.
- Deve-se procurar otimizar os fatores usados mais frequentemente, isto é, com mais peso no programa.

# Medidas de desempenho do I/O

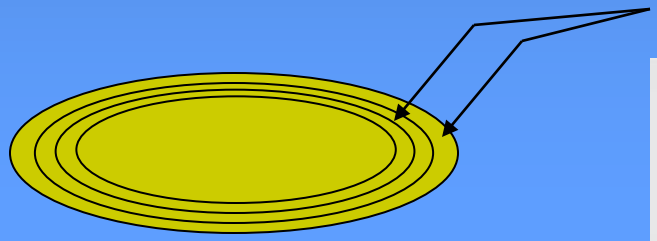
- Há 2 grandezas fundamentais:
  - Latência (milisseg). Tempo até se iniciar a transferência (relacionado com o tempo de procura de informação, tempo de inicialização do canal de transferência, etc).
  - Largura de banda (Mbytes/seg). Máxima quantidade de informação transferida por unidade de tempo.
- Cada acesso a um periférico inclui um período de latência e outro de transferência (à velocidade máxima ou perto)
- A velocidade de transferência efetiva (média) depende do peso relativo da latência.

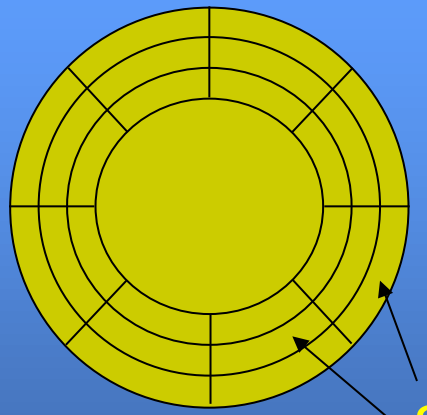
# Discos duros



Discos (2 faces cada)

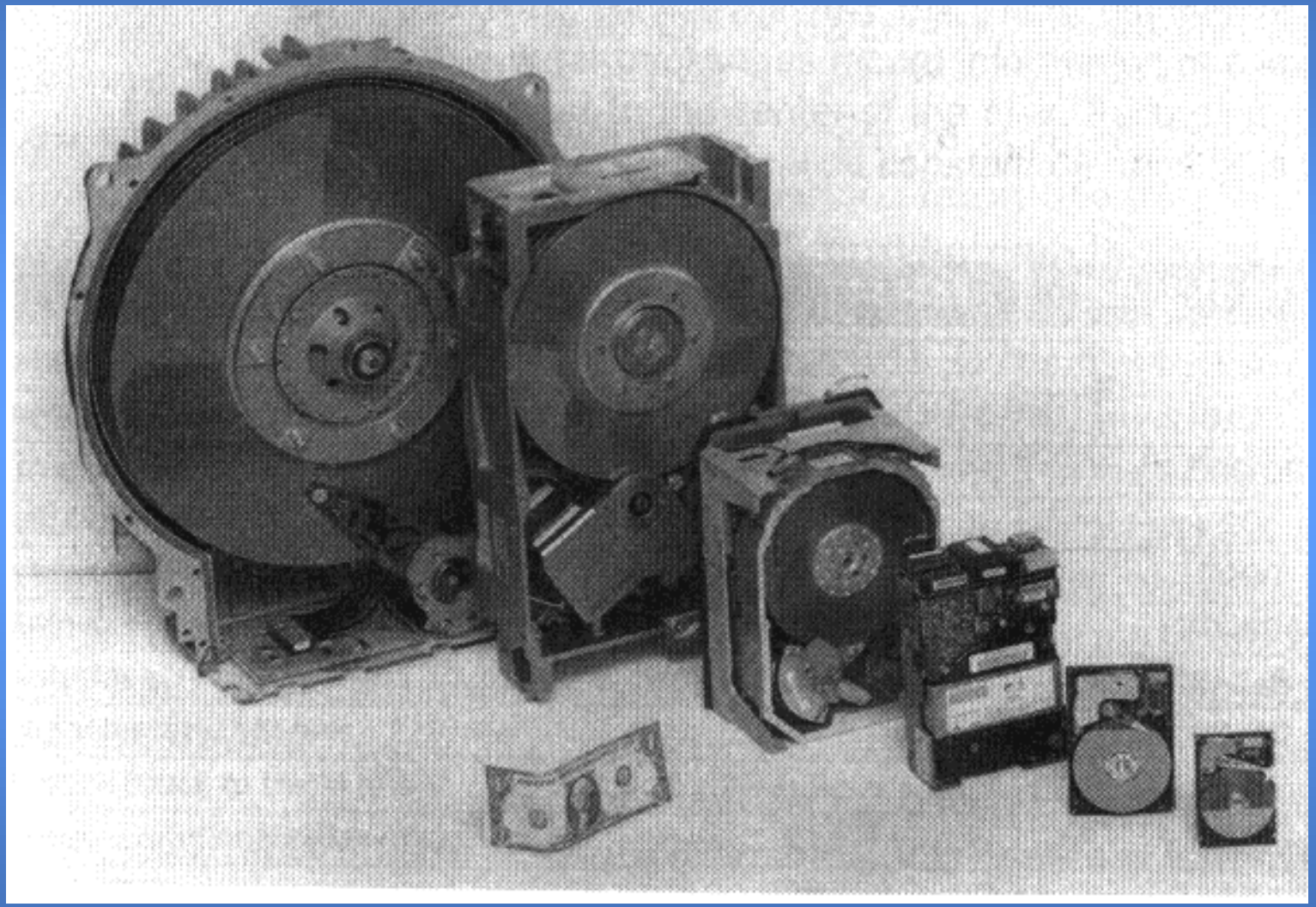
Pistas concêntricas





Sectores

# Evolução dos discos duros



# Acesso a um disco duro

- Supondo:
  - Um tempo médio de posicionamento do braço de 9 ms
  - Velocidade de rotação de 5400 rpm (rotações/minuto)
  - 10 MB/s de velocidade de transferência
- Qual o tempo médio para ler um setor de 1 KB?
  - Latência de rotação em média – 5.5 ms (1/2 volta a 90 rotações/seg)
  - Latência total (posicionamento + rotação) – 14.5 ms (9 + 5.5)
  - Tempo de transferência – 0.1 ms (1 KB/10 MB)
  - Tempo total de leitura – 14.6 ms (14.5 + 0.1)
  - Isto são 14 600 000 ciclos de um processador de 1GHz!
  - Peso da latência (dominante!) – 99.3% (14.5 / 14.6)
- Em média, conseguem-se ler 68 sectores/seg (1/14.6 ms) em acesso aleatório. A taxa é muito mais elevada se os setores forem lidos de seguida (latência só no primeiro setor).
- Se o disco fosse de 7200 rpm, a latência de rotação → 4 ms

# Comunicação via rede

- Fatores fundamentais na comunicação:
  - Tempo de acesso à informação (disco, por exemplo)
  - Tempo de processamento local (normalmente desprezável, mas pode ser importante se os dados tiverem muito processamento – compressão, por exemplo)
  - Tempo de comunicação (tal como o acesso aos discos, inclui latência e tempo de transmissão)
- O tempo total de comunicação é o somatório destes tempos parciais.
- Normalmente, o fator limitativo é o disco, mas uma rede lenta pode estrangular a comunicação.
- Exemplo típico: servidor acedido por rede.

# Exemplo: servidor de WWW

- Supondo um servidor de WWW com:
  - Disco com tempo de procura de pista (*seek-time*) de 8 ms, 10.000 rpm, e 10 Mbytes/seg de taxa de transferência
  - Páginas HTML com o tamanho médio de 8 KB e que cabem totalmente dentro de um setor (para simplificar).
  - Barramento de periféricos de 133 Mbytes/seg
  - Rede: clientes locais a 100 Mbits/seg e por internet a 10 Mbits/seg

a) Quanto tempo é necessário, em média, para ler uma página HTML do disco desde a altura em que se começa a procurá-la no disco?

- O tempo médio de achar o início da página: = 11 ms
  - 8 ms (seek time) para achar a pista no disco
  - 3 ms (tempo de meia volta, em média).
- O tempo médio de leitura de uma página HTML é de 8 Kbytes/10 Mbytes/seg ≅ 0.8 ms.
- Resposta final = 11 + 0.8 = 11.8 ms

# Exemplo: servidor de WWW

- Supondo um servidor de WWW com:
  - Disco com tempo de procura de pista (*seek-time*) de 8 ms, 10.000 rpm, e 10 Mbytes/seg de taxa de transferência
  - Páginas HTML com o tamanho médio de 8 KB e que cabem totalmente dentro de um setor (para simplificar).
  - Barramento de periféricos de 133 Mbytes/seg
  - Rede: clientes locais a 100 Mbits/seg e por internet a 10 Mbits/seg

b) Quantas páginas HTML por segundo (em média) podem ser transferidas do servidor para os clientes na rede local?

- Rede local (≅ 12.5 Mbytes/segundo) e o barramento (133 Mbytes/segundo) são menos limitativos do que a taxa de transferência do disco (10 Mbytes/seg).
- Velocidade máxima que o disco permite: uma página por cada 11.8 ms (em média), ou 1/11.8 ms ≅ 85 páginas/segundo.

# Exemplo: servidor de WWW

- Supondo um servidor de WWW com:
  - Disco com tempo de procura de pista (*seek-time*) de 8 ms, 10.000 rpm, e 10 Mbytes/seg de taxa de transferência
  - Páginas HTML com o tamanho médio de 8 KB e que cabem totalmente dentro de um setor (para simplificar).
  - Barramento de periféricos de 133 Mbytes/seg
  - Rede: clientes locais a 100 Mbits/seg e por internet a 10 Mbits/seg

c) Quantas páginas HTML por segundo (em média) podem ser transferidas do servidor para os clientes remotos (internet)?

- Internet: Apenas 1.25 Mbytes/segundo, menor que a velocidade de transferência do disco). A disponibilização de uma página no cliente demora:
  - 11 ms para achar a página no disco
  - 6.4 ms (tempo médio de leitura de uma página HTML é de 8 Kbytes/1.25 Mbytes/seg)

Logo, podem ser transferidos 1/17.4 ms ≅ 57 páginas/segundo

# Exemplo: servidor de WWW

- Supondo um servidor de WWW com:
  - Disco com tempo de procura de pista (*seek-time*) de 8 ms, 10.000 rpm, e 10 Mbytes/seg de taxa de transferência
  - Páginas HTML com o tamanho médio de 8 KB e que cabem totalmente dentro de um setor (para simplificar).
  - Barramento de periféricos de 133 Mbytes/seg
  - Rede: clientes locais a 100 Mbits/seg e por internet (mais lenta) a 1 Mbits/seg

c) Quantas páginas HTML por segundo (em média) podem ser transferidas do servidor para os clientes remotos (internet)?

- Internet: Apenas 0.125 Mbytes/segundo, menor que a velocidade de transferência do disco). A disponibilização de uma página no cliente demora:
  - 11 ms para achar a página no disco
  - 64 ms (tempo médio de leitura de uma página HTML é de 8 Kbytes/0.125 Mbytes/seg)

Logo, podem ser transferidos 1/75 ms ≅ 13 páginas/segundo

# Conclusões

- Os periféricos são componentes fundamentais dos computadores, para uso interno e para comunicação com os utilizadores e outros computadores.
- Os periféricos são normalmente mais lentos do que a memória, pois lidam com sinais externos e/ou dispositivos eletromecânicos.
- Por este motivo, os computadores têm barramentos hierárquicos.
- Existem várias formas de um computador lidar com os periféricos (*polling*, interrupções, DMA, co-processador de entradas saídas, etc). A mais adequada depende da aplicação.

# Conclusões (cont.)

- O desempenho dos computadores depende de vários fatores que não são independentes (tecnologia, arquitetura e compiladores).
- O que interessa otimizar é o tempo de execução e não um dado fator (para comparação, usam-se *benchmarks*).
- Mesmo que se melhore um dos fatores sem afetar os restantes, o tempo de execução pode não melhorar tanto como esse fator (lei de Amdahl).
- Os estrangulamentos acabam por ser dominantes (“lei do elo mais fraco”...)

# Exercícios

1. Suponha que estabeleceu uma ligação entre dois computadores usando uma linha série com um protocolo assíncrono, usando um bit de paridade, dois stop bits e um ritmo de transmissão de 19200 baud.
   a) Quanto tempo demora, no mínimo, a transmitir 10 Kbytes?
   b) Supondo que o ritmo de transmissão foi o máximo possível, qual a percentagem dos bits de dados no total de bits transmitidos?
   c) Imagine que, devido ao processamento local da informação, o emissor não consegue transmitir os dados ao ritmo máximo possível. Supondo que não altera nada no recetor, como é que o recetor lida com esta situação?
   d) Suponha agora que o problema está no recetor, isto é, que tem de processar os bytes recebidos e que não consegue recebê-los e processá-los ao ritmo máximo possível. Indique possíveis soluções.

# Exercícios

2. Suponha que um computador com 2 GHz de frequência de relógio corre um programa que executa 1000 instruções, das quais cerca de 20% demora 1 ciclo de relógio a executar, 30% demoram 2 ciclos e 50% acedem à memória, gastando 2 ciclos de relógio mais o tempo de acesso à memória (4 nanossegundos).
   a) Estime qual o tempo total de execução do programa.
   b) Imagine que se arranjou uma memória duas vezes mais rápida, com tempo de acesso de 2 nanossegundos. Estime qual a melhoria no tempo de execução do programa.
   c) Suponha agora que 10% dos acessos à memória são na realidade acessos a um controlador de um disco (periférico mapeado em memória), cujo *seek-time* é de 8 ms e cuja velocidade de rotação é de 7200 rpm. Cada acesso destes faz o processador esperar até estar concluído. Volte a estimar o tempo de execução do programa.

# Exercícios

3. Suponha que no caso do servidor de WWW referido nestes slides é possível ter vários discos, cada um com uma cópia de todas as páginas HTML.
   a) Explique porque é que tal esquema permite melhorar o número de páginas/segundo enviadas para os clientes.
   b) Supondo que só há clientes ligados ao servidor em rede local, estime (em média) (i) qual o número máximo de discos que o servidor consegue explorar e (ii) quantas páginas/segundo o servidor pode enviar nesse caso.
   c) Idem, mas considerando agora o caso de haver apenas clientes remotos, ligados ao servidor por internet.

# Exercícios

4. Pretende-se que um processador leia de um disco um ficheiro com 140 setores de 1 Kbyte cada. O acesso a um setor obriga a um posicionamento do braço (*seek-time* médio de 8 ms), estando o disco a rodar a 7200 rpm. O controlador do disco lê então o setor para um *buffer* interno (de tamanho igual ao do setor), podendo o processador aceder aos bytes desse setor por vulgares acessos à memória (a esse buffer).
   a) O processador faz a transferência dos dados do buffer para a memória principal por software, demorando 50 nanossegundos por cada byte, e que o processador espera que um setor esteja disponível usando o método de *polling*. Estime o tempo total gasto pelo processador com a leitura do ficheiro para memória.
   b) Suponha agora que o controlador notifica o processador que um setor está disponível por meio de uma interrupção, e que enquanto está à espera está a correr outros programas. Estime (i) o tempo total que a leitura demora e (ii) o tempo total gasto pelo processador com essa leitura.
   c) Imagine que para melhorar o tempo da leitura, a transferência de cada setor para a memória passou a ser feita por DMA em modo de rajada, que consegue que cada byte seja transferido em 4 nanossegundos. Responda às questões da alínea anterior mas relativas a este caso.