Entradas/saídas

Sumário:

Programação de controladores de entradas/saídas usando interrupções.

Programação do controlador série de um PC usando interrupções.

Bibliografia:

S. Dandamudi, *Fundamentals of Computer Organization and Design*, Springer 2003, Secções 20.1, 20.2, 20.3 e 20.6

Inconvenientes do uso do “polling”

- Os controladores que só suportam espera activa 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 efectuar computações enquanto espera que a transferência de dados acabe
- O controlador actua 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
- O controlador de E/S transfere dados de/ para o periférico enquanto o CPU faz outra coisa
- O controlador de E/S interrompe o CPU
- CPU finaliza a operação e pode lançar a transferência seguinte

Ponto de vista do CPU

- Emite comando de leitura
- Faz outra coisa
- Em cada ciclo de instrução verificar se há interrupção
- Se há interrupção
  - Salvar estado da computação em curso(registos)
  - Processar interrupção
    - Ir buscar o dado ao controlador e armazená-lo na memória

Problemas a resolver

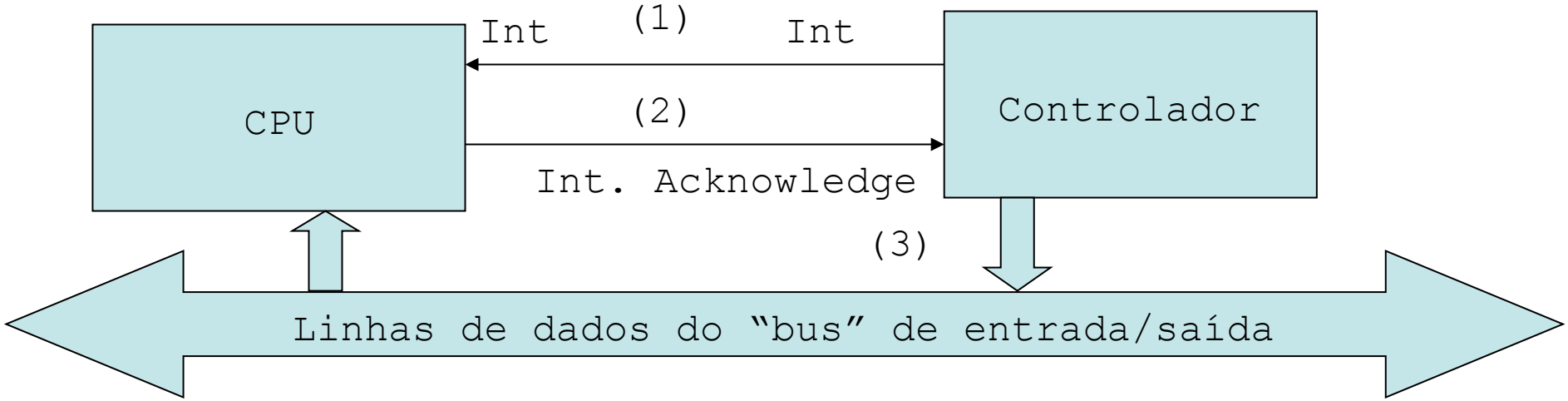
- Como identificar o controlador que efectuou a interrupção?
- Como lidar com múltiplas interrupções?
  - i.e. uma rotina de tratamento de interrupções ser por sua vez interrompida

Como identificar o controlador que fez a interrupção? (1)

- Uma linha diferente para cada controladpr
  - 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

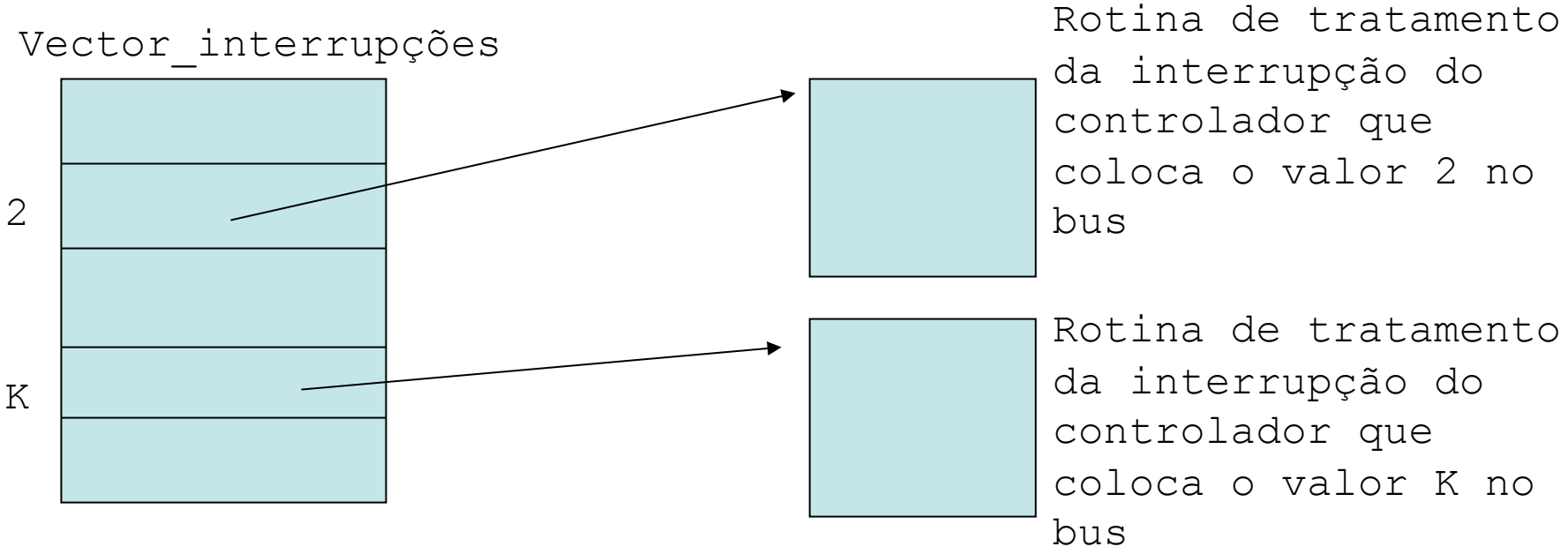
Como identificar o controlador que fez a interrupção? (2)

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



Vector de interrupções

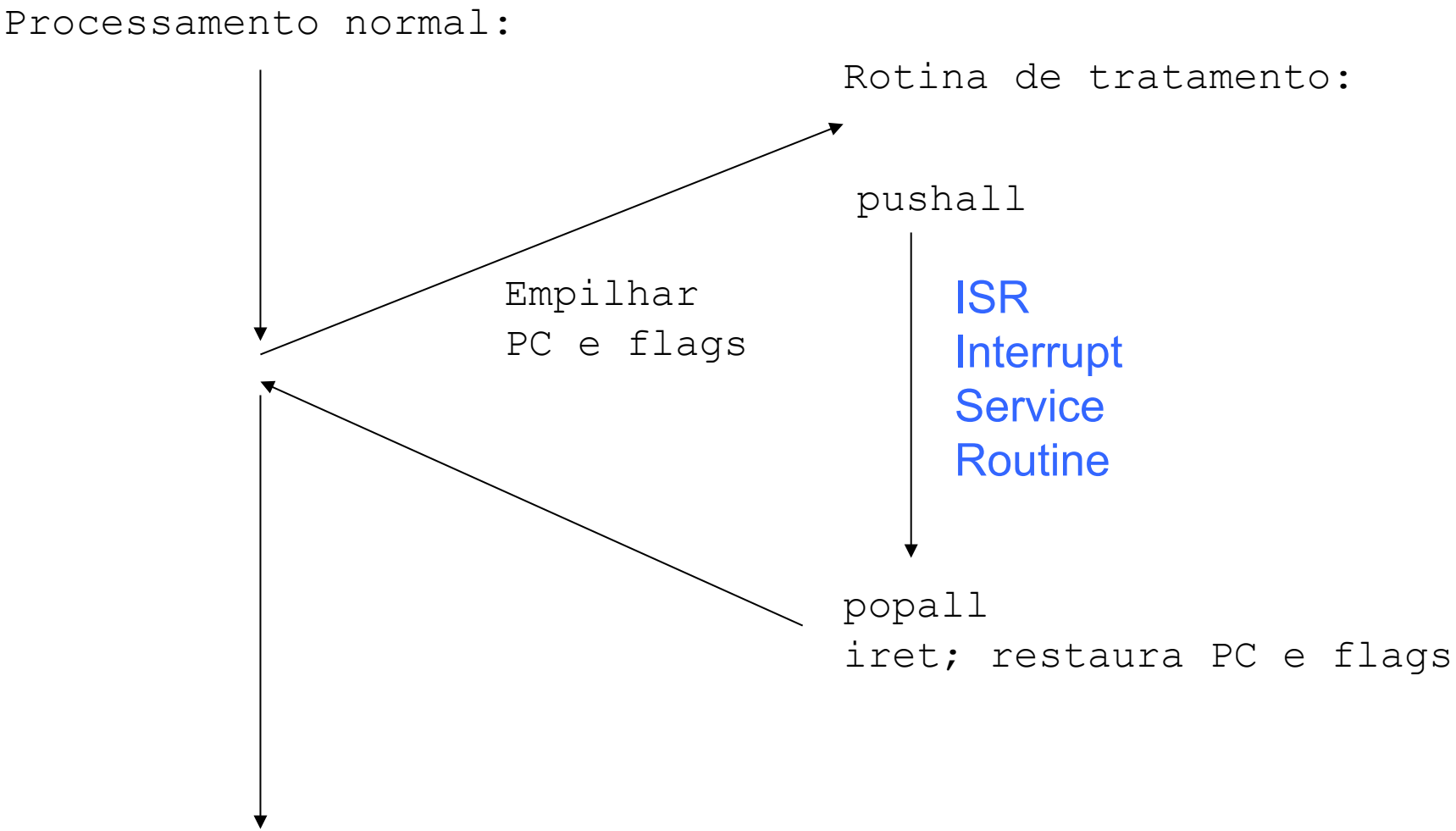
- Após obtido o código V associado à interrupção
- O vector de interrupções contém N entradas; cada entrada é o endereço de uma rotina de tratamento; PC ← vector_interrupções[V]



Ciclo de execução de instruções

```
while (1){
    Fetch : obter a instrução na posição de memória
    apontada pelo PC; actualizar PC
    Execute : executar a instrução
    if( interrupção pendente){
        salvar o PC e as “flags” (usualmente no “stack”)
        identificar a origem da interrupção
        carregar o PC com o endereço da rotina que
            faz o tratamento da situação (interrupt
            handler)
    }
}
```

Tratamento de uma interrupção corresponde a uma troca de contexto



Estado da computação

- *PSW (**P**rocessor **S**tatus **W**ord)*
  - *Program counter* ou *Instruction Pointer*
  - Registo de *Flags*
    - Posicionadas pelas operações aritméticas e lógicas (Zero, Parity, Sign, Overflow, …)
  - Modo de funcionamento do CPU
    - Utilizador/Supervisor
      - Só no modo supervisor é que é possível executar instruções privilegiadas
    - Receptivo ou não a interrupções
      - Interrrupt Flag (ver mais à frente)
- Registos do processador
  - Pentium: eax, ebx, ecx, ….

Salvaguarda/restauro do estado da computação

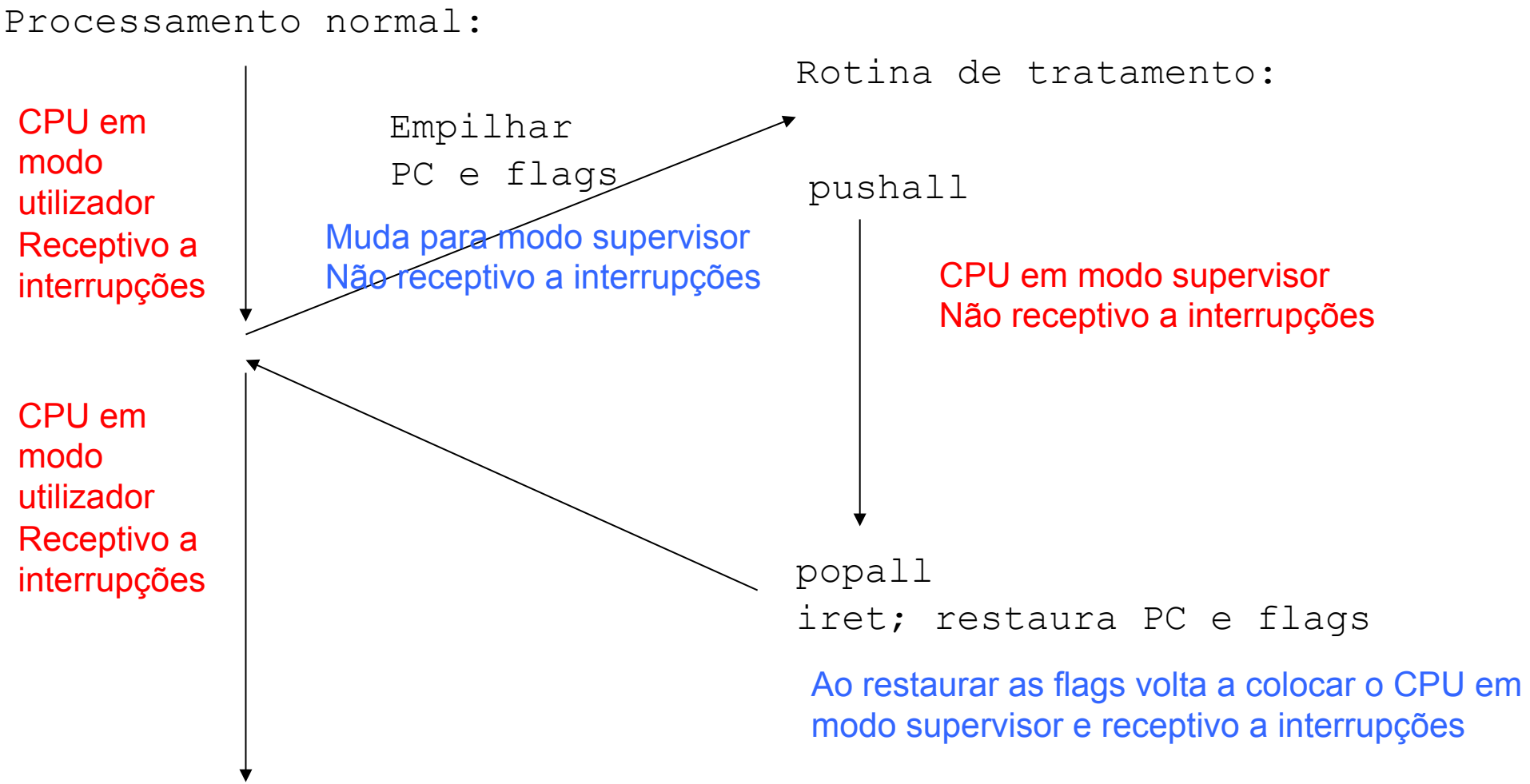
- O hardware só salva automaticamente o PC e as flags
- É a rotina de tratamento de interrupções que tem a obrigação de salvar os registos do CPU que "estraga".
- Alguns CPUs têm instruções máquina para empilhar / desempilhar todos os registos

```
Interrupt_handler:
      pushall ; empilha todos os regs.
      ...     ; tratamento da interrupção
      popall  ; restaura todos os registos
      iret        ; return_from_interrupt
```

Como é que se retorna da rotina de tratamento da interrupção?

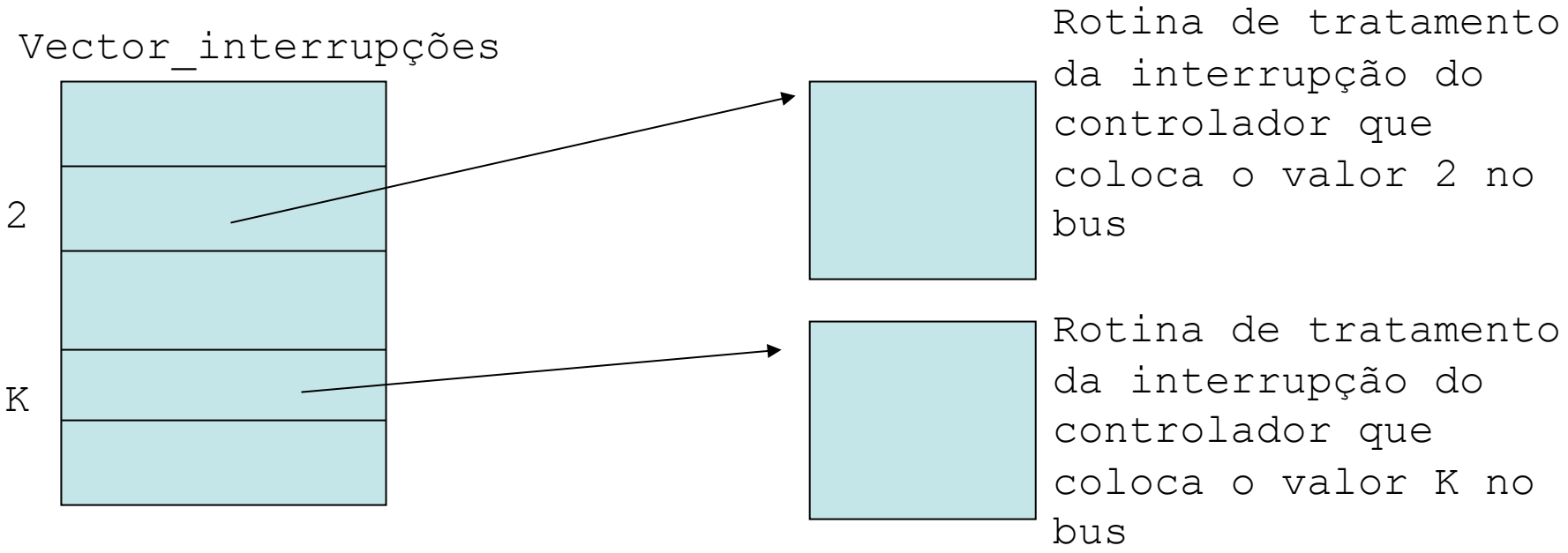
- Há uma instrução máquina que se encarrega disso:
  - Return_from_interrupt
  - Se foi empilhado primeiro o PC e depois as flags
  - A instrução faz
    - Pop flags
    - Pop PC
  - A execução continua no ponto onde se encontrava quando ocorreu a interrupção

Quando se entra na ISR o CPU muda para modo supervisor



Vector de interrupções

- Após obtido o código V associado à interrupção
- O vector de interrupções contém N entradas; cada entrada é o endereço de uma rotina de tratamento; PC ← vector_interrupções[V]



Inicialização do vector de interrupções

- O software de sistema inicializa o vector 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

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 de instruções (actualização)

```
while (1){
    Fetch : obter a instrução na posição de memória apontada pelo
    PC; actualizar 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 Pentium é STI (Set Interrupt Flag)
  - Disable Interrupts(IF ← 0): no ciclo de fetch/execute não é testado se há interrupções pendentes; No Pentium 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 fôr feito, só será colocada a 1 quando houver o “interrupt return”
- 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 fôr seguro.

Tipos de interrupções

- Interrupções hardware (ou externas)
  - Linha exterior ao CPU activada por um controlador de E/S
- Interrupções internas
  - Resultam da execução da instrução máquina anterior
    - **Excepções:** divisão por 0, overflow, violação de memória, execução de instrução privilegiada em modo utilizador, ...
    - **Interrupção por software:** uma instrução máquina que provoca deliberadamente uma interrupção – usada para fazer chamadas ao sistema

Interrupções de múltiplos controladores

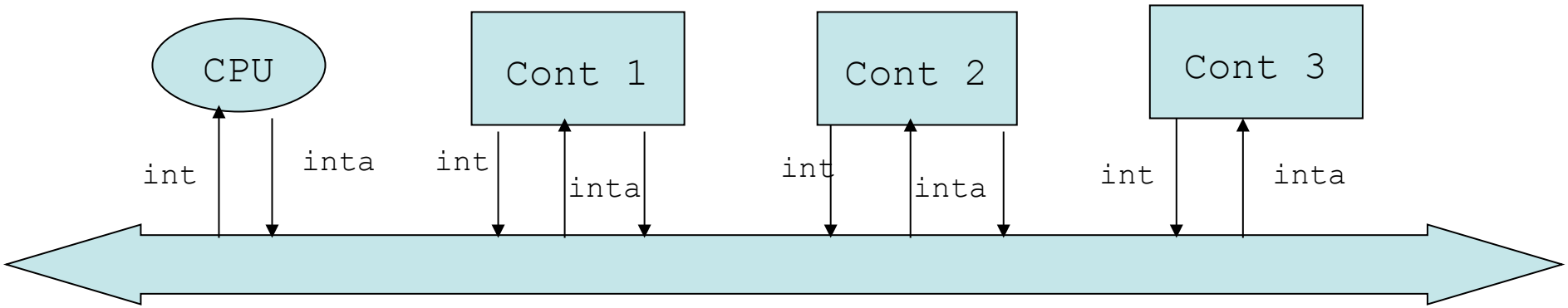
- As necessidades dos controladores diferem:
  - Tempo máximo ao fim do qual têm de receber atenção
  - Duração do processamento da interrupção
- O processamento de uma interrupção de um periférico lento pode levar mais tempo do que o tempo máximo de espera de um periférico rápido
- Para satisfazer estas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

Como identificar o controlador que fez a interrupção

- Se há mais de um controlador, é preciso garantir que só um deles coloca um valor no “bus” de dados
  - “Daisy Chain”
    - Interrupt Acknowledge enviado pelo CPU através de uma cadeia que passa por todos os controladores (int_ack_in e int_ack_out)
    - Periféricos mais rápidos mais “perto” do CPU
    - Pouco flexível
  - Arbitragem em paralelo
    - O árbitro do “bus” decide qual dos módulos tem acesso ao “bus”
    - É o árbrito (ou controlador de interrupções) que envia o código da interrupção para o “bus”
    - IBM PC/ Bus PCI

Como identificar o controlador que fez a interrupção (2)

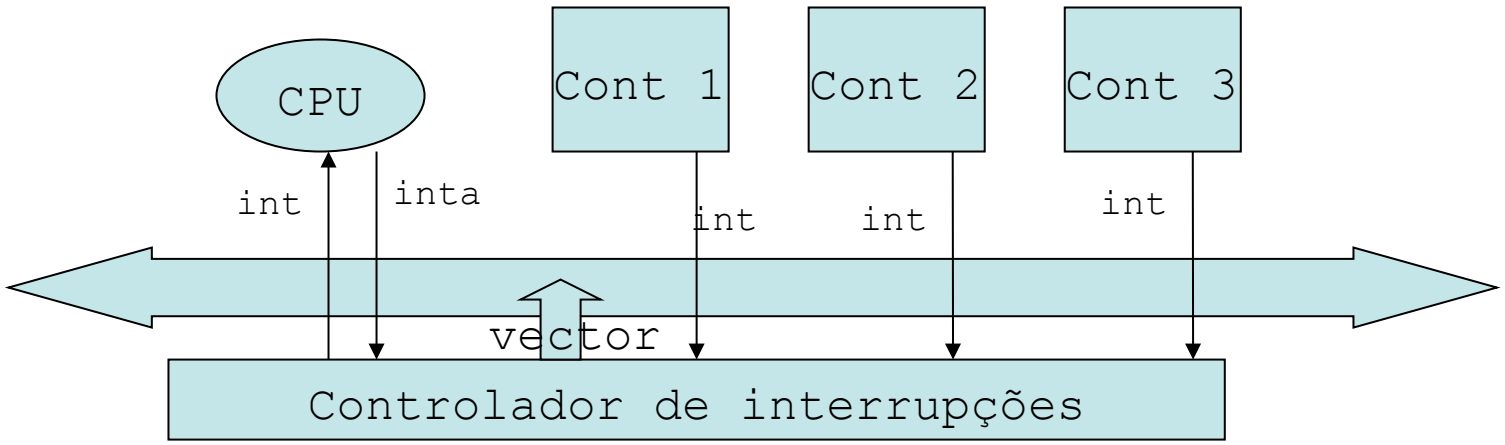
– “Daisy Chain”



O controlador i só deixa passar o sinal INTA para o controlador i+1 se não tiver uma interrupção pendente

Como identificar o controlador que fez a interrupção (3)

– Arbitragem em paralelo; através de um controlador de interrupções

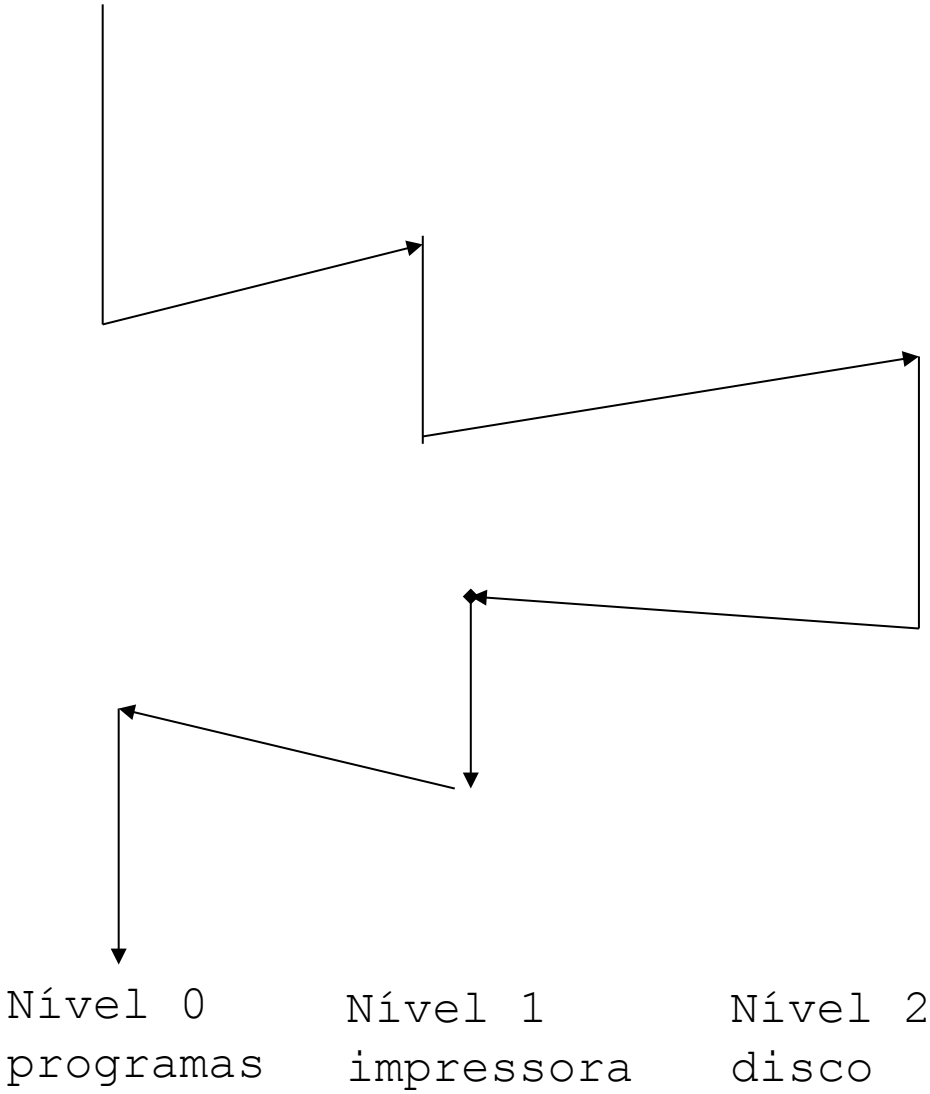


Prioridade das interrupções (1)

- 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 (2)

- 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 vector de interrupções

Índice no vector de interrupções

- Atribuição de níveis a controladores
  - **Manual**: sistemas embebidos; outros casos em que o número e tipo de periféricos fixo
  - **Automática**: no “boot” o hardware/software de sistema verifica que periféricos existem e atribui a cada um uma prioridade e um índice no vector (exemplo bus PCI)
  - **Dinâmica:** o bus USB permite inserir/remover periféricos com o sistema “no ar”: nesse caso há uma interrupção gerada pelo controlador e é instalado um novo “interrupt handler” para o novo periférico.

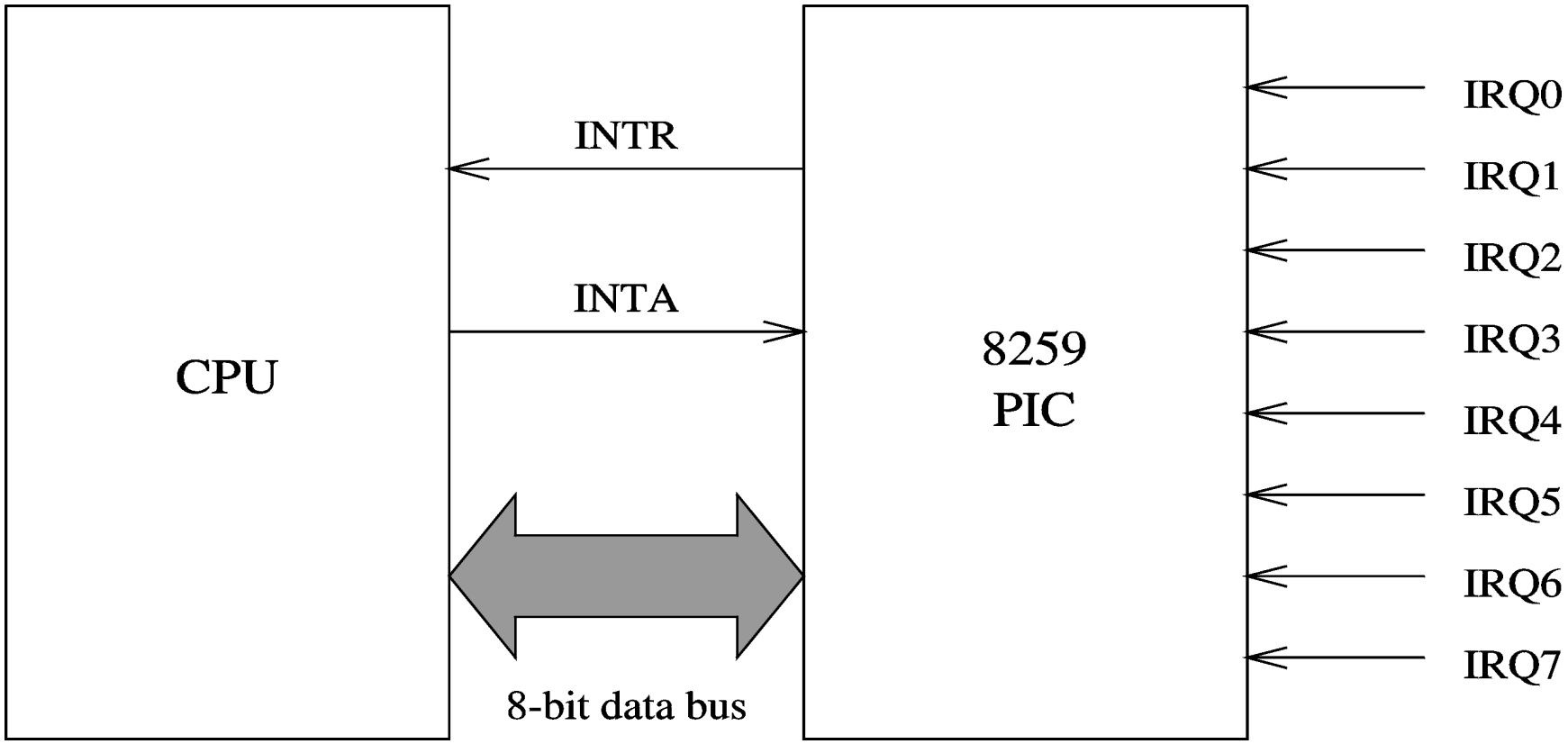
Interrupções múltiplas

- Cada linha de interrupção tem uma prioridade
- Uma linha de alta prioridade chega ao CPU mesmo que outra linha de interrupções (menos prioritária) esteja activa

Exemplo – 80x86

- 80x86 tem uma linha de interrupções
- Os sistemas baseados no 80x86 usam o controlador de interrupções 8259A
- O 8259A tem 8 linhas de interrupção

8259 PIC


CPU
INTR
INTA
8-bit data bus
8259
PIC
IRQ0
IRQ1
IRQ2
IRQ3
IRQ4
IRQ5
IRQ6
IRQ7

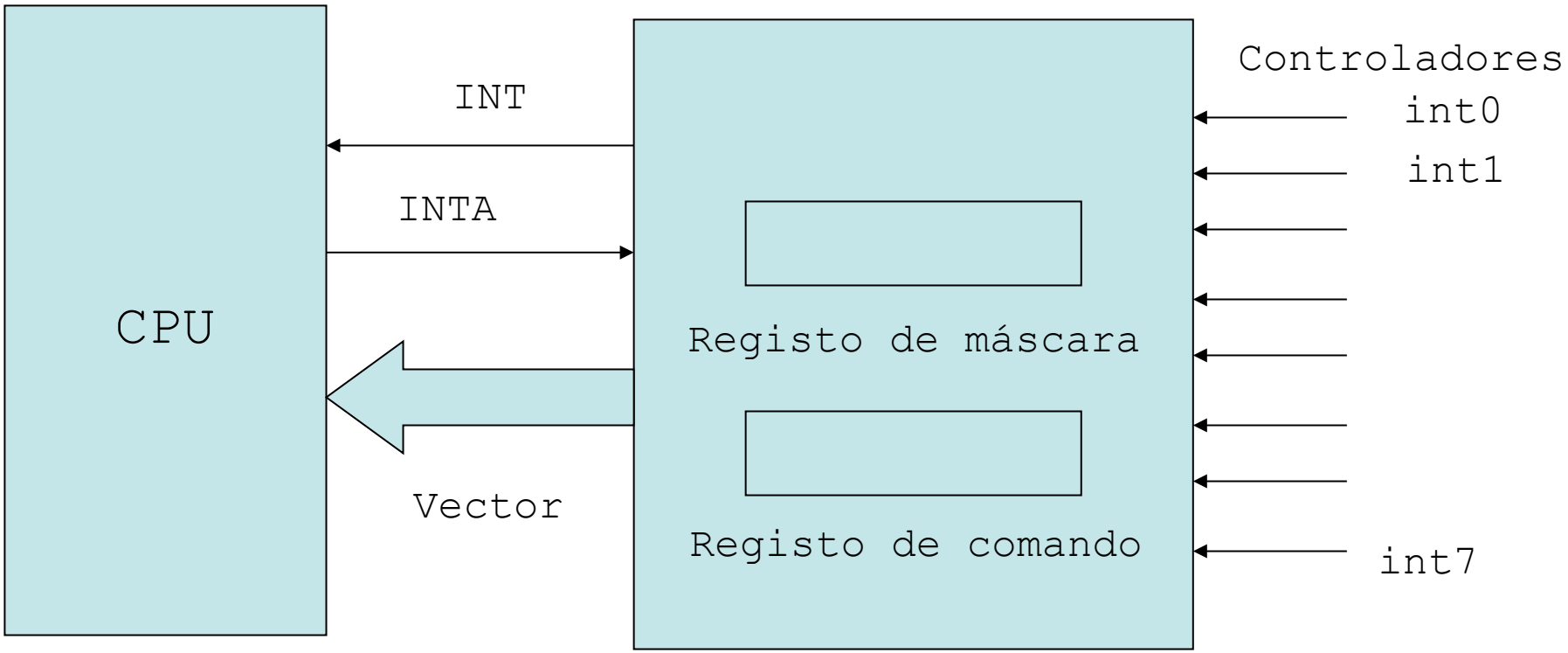
Sequência de eventos

- Periférico envia interrupção
- 8259A aceita a interrupção
- 8259A determina a prioridade
- 8259A alerta 80x86 (activa o pino INTR)
- CPU faz “acknowledge”
- 8259A coloca o vector correspondente no bus de dados
- CPU salta para a rotina de tratamento de interrupções

8259 Programmable Interrupt Controller (PIC)

- 8259 suporta até 8 controladores de dispositivos
  - Os controladores enviam interrupções que são recebidos nas entradas IRQ0 a IRQ7
- 8259 pode ser programado para atribuir prioridades de várias fomas.
  - Nos PCs é usado um esquema de prioridade fixa
    - IRQ0 tem a prioridade mais elevada prioridade e o IRQ7 a mais baixa
- 8259 tem dois registos para programação
  - Interrupt Command Register (ICR)
    - Usado para dar comandos ao 8259
  - Interrupt Mask Register (IMR)

Controlador 8259A (PIC)



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

8259 PIC

- Interrupt Mask Register (IMR) é um registo de 8 bits
  - Usado para activar ou desactivar individualmente as linhas de entrada de interrupção de IRQ0 a IRQ7
    - Bit 0 está associado ao IRQ0, bit 1 ao IRQ1, . . .
    - Um bit a 0 activa a interrupção correspondente (1 desactiva)
- CPU só reconhece interrupções hardware (externas) quando a IF está a 1
- Endereços:
  - ICR: 20H
  - IMR:21H

Registos do PIC

- Registo de comando (0x20):
  - Comando EOI: 0x20
  - out 0x20, 0x20
- Registo de máscara
  - 8 bits;1- nível desligado, 0 – nível ligado
  - Desligar o nível 3

```
cli
in al, 0x21
or al, 0000 1000B
out 0x21,al
sti
```

  - Ligar o nível 2

```
cli
in al, 0x21
and al, 1111 1011B
out 0x21,al
sti
```

8259 PIC

Exemplo: Desactivar todas as interrupções do 8259 , salvo o “system timer”

```
mov     AL,0FEH
out     21H,AL
```

- O 8259 precisa de saber quando é que a rotina de tratamento de interrupção (ISR) termina para que possa deixar passar outras interrupções pendentes
  - O comando End-of-interrupt (EOI) dá essa informação ao 8259; escreve-se 20H no ICR

```
mov     AL,20H
out     20H,AL
```

  - Este pedaço de código deve ser usada antes do **`iret`**

Sistema de interrupções do bus ISA

- O bus ISA utiliza 2 8259As em conjunto
- Ligação via interrupção 2
- Suporta 15 linhas
  - 16 linhas menos 1 para a interligação
- Incorporated nos “chip sets” para compatibilidade

Níveis de interrupção e índice no vector de interrupções

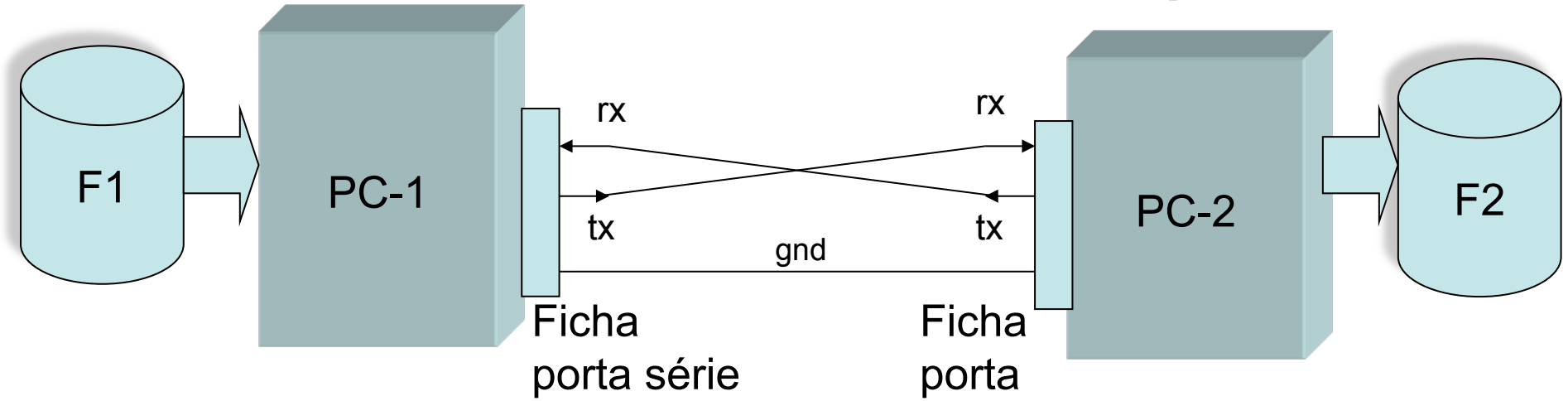
- Uso das entradas IRQ0 a IRQ7 na arquitectura dos PCs

No. do IRQ	Índice no vector	Dispositivo
0	08H	Relógio
1	09H	Teclado
2	0AH	reservado (2º 8259)
3	0BH	Porta série (COM1)
4	0CH	Porta série (COM2)
5	0DH	Disco rígido
6	0EH	Floppy Drive
7	0FH	Porta paralela (LPT1)

Exemplo do uso da porta série com interrupções

- Transferência de ficheiros entre dois computadores usando a porta série
- O uso de interrupções tem vantagens
  - Melhor utilização da ligação
  - Menor possibilidade de perder dados na recepção

Troca de ficheiros usando a porta série




```
#define EOF 0xFF
FILE *f = fopen("F1","r");
do{
    c =fgetc(f);
    send_serial(c);
}while(!feof(f));
send_serial(EOF);
fclose(f);
```

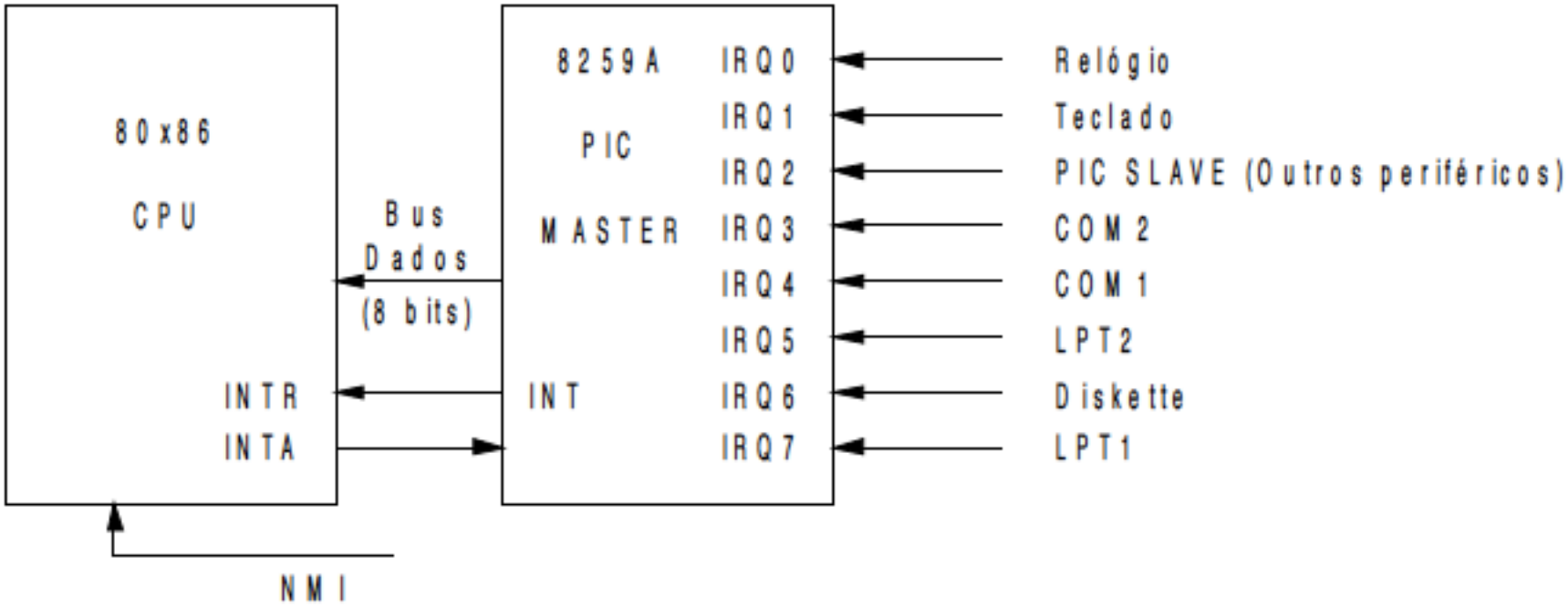
```
#define EOF 0xFF
FILE *f = fopen("F2","r");
do{
    c = receive_serial();
    fputc(c, f);
}while( c != EOF );
fclose(f);
```

Espera activa Vs Interrupções

- As chamadas *fgetc* e *fputc* podem provocar chamadas ao sistema e bloqueio do processo
  - Podem ser perdidos bytes na recepção e a transmissão decorre mais devagar do que seria possível
  - Solução: usar recepção e transmissão com interrupções

UART e interrupções

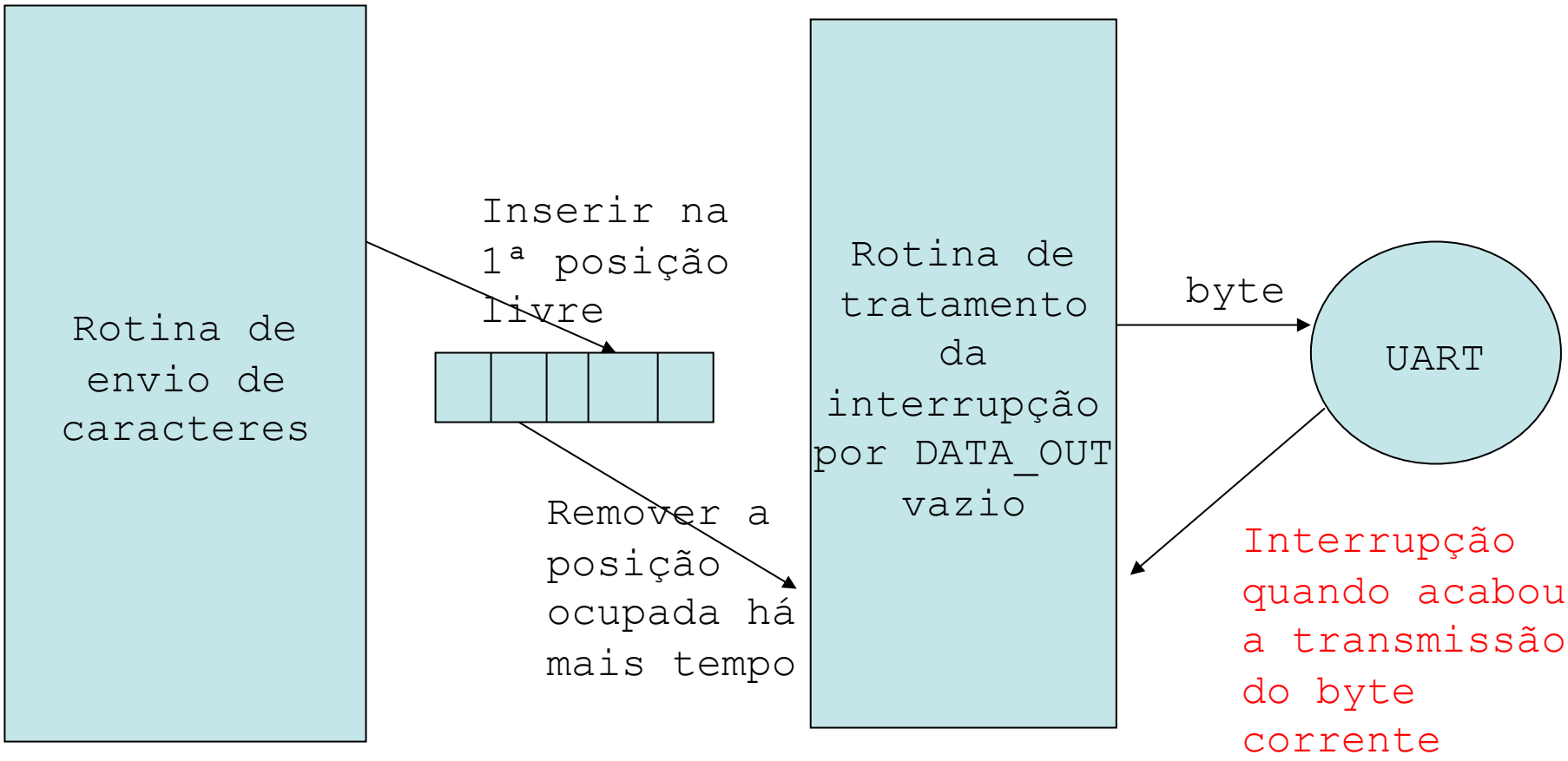
- Porta COM1: pode fazer interrupções



Interrupções na UART

- COM1: - IRQ4; índice 12 do vector de interrupções
- Para a UART gerar interrupções é preciso:
  - Colocar a 1 os bits 3 e 1 do registo MCR (0x3FC)
  - No registo IER (Interrupt Enable Register) – 0x3F9:
    - Colocar a 1 o bit 0 – chegada de um carácter
    - Colocar a 1 o bit 1- “buffer” de transmissão vazio
  - Ver detalhes no guia do trabalho prático desta semana

Transmissão com interrupções



Gestão do “buffer” circular

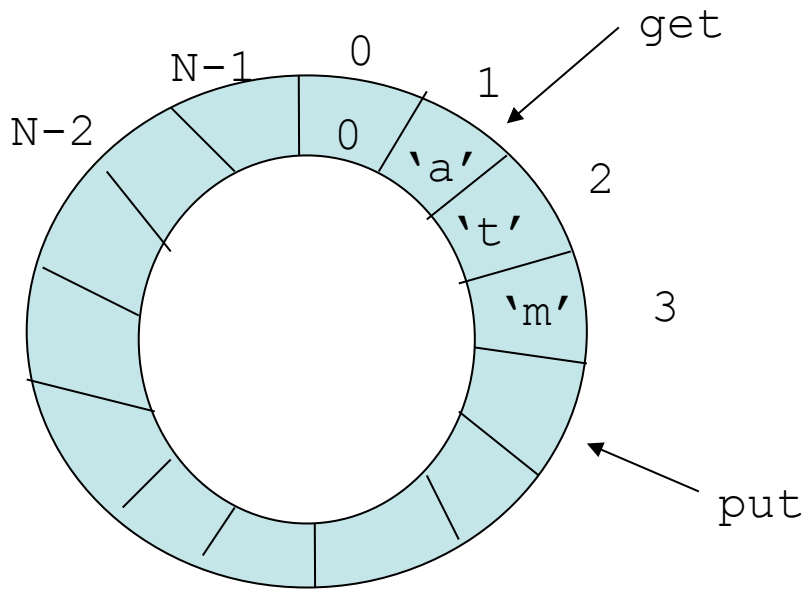
```
#define BUFSIZ 128

unsigned char buf[BUFSIZ];

int put;  // 1ª casa livre

int get;  // casa ocupada há mais
          // tempo

int nc;   // nº de bytes no buffer
```




```
unsigned char bufGet(void){
/* assume que buf não está
vazio */
  unsigned char x = buf[get];
  get = (get + 1) % BUFSIZ;
  nc --;
  return x;
}
int bufEmpty(){
  return (nc == 0);
}
```

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

Transmissão com interrupções

```
Enviar_serie(ch){
    if bufFull()
                esperar;
    if bufEmpty(){
        bufPut(ch)
        ligar interrupções TXEMPTY
    } else
        bufPut(ch)
}
```

```
Rotina de tratamento de
interrupções TX_EMPTY{
    ch = bufGet();
    outportb( DATA_OUT, ch );
    if bufEmpty()
        desligar as
        interrupções TXEMPTY

    outportb( 8259_COMMAND,
EOI)
}
```

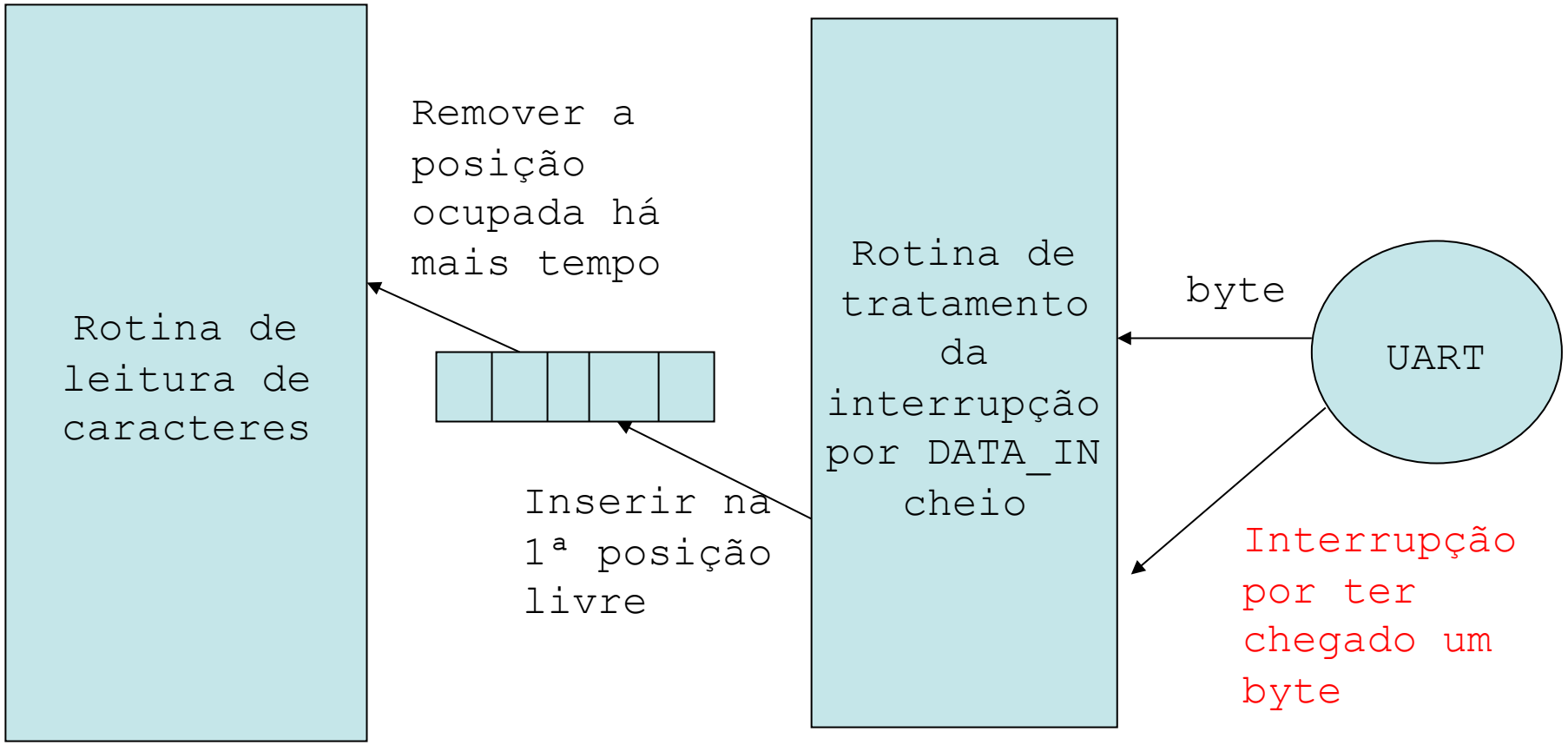
Notas importantes:

-a rotina **enviar_série** espera em ciclo que haja espaço no “buffer”

-A rotina de interrupções desliga as interrupções da UART se o “buffer” fica vazio

-A rotina enviar_série verifica se o “buffer” está vazio; se tal acontecer deposita o carácter no “buffer” e liga as interrupções na UART

Recepção com interrupções



Recepção com interrupções

```
unsigned char Receber_serie(){
    while (bufEmpty());
                /* esperar */
    disable();
    ch = bufGet();
    enable();
    return ch;
}
```

```
Rotina de tratamento de
interrupções RX_FULL{
    ch = inportb( DATA_IN );
    if bufFull()
        # erro, não há espaço no
        buffer mas não se pode
        esperar
    else bufPut(ch);

    outportb( 8259_COMMAND, EOI)
}
```

Notas importantes:

-a rotina **receber_série** espera em ciclo (com as interrupções ligadas) que haja um elemento no “buffer”

-A rotina de interrupções pode encontrar o “ buffer” cheio; se tal acontece não pode ficar em espera activa (porquê ?)

-bufGet() e bufFull() actualizam nc e pode ocorrer o problema apontado a seguir. Por isso, a presença do *disable()* – CLI e *enable()* - STI

Actualização do nº de bytes no “buffer” (1)

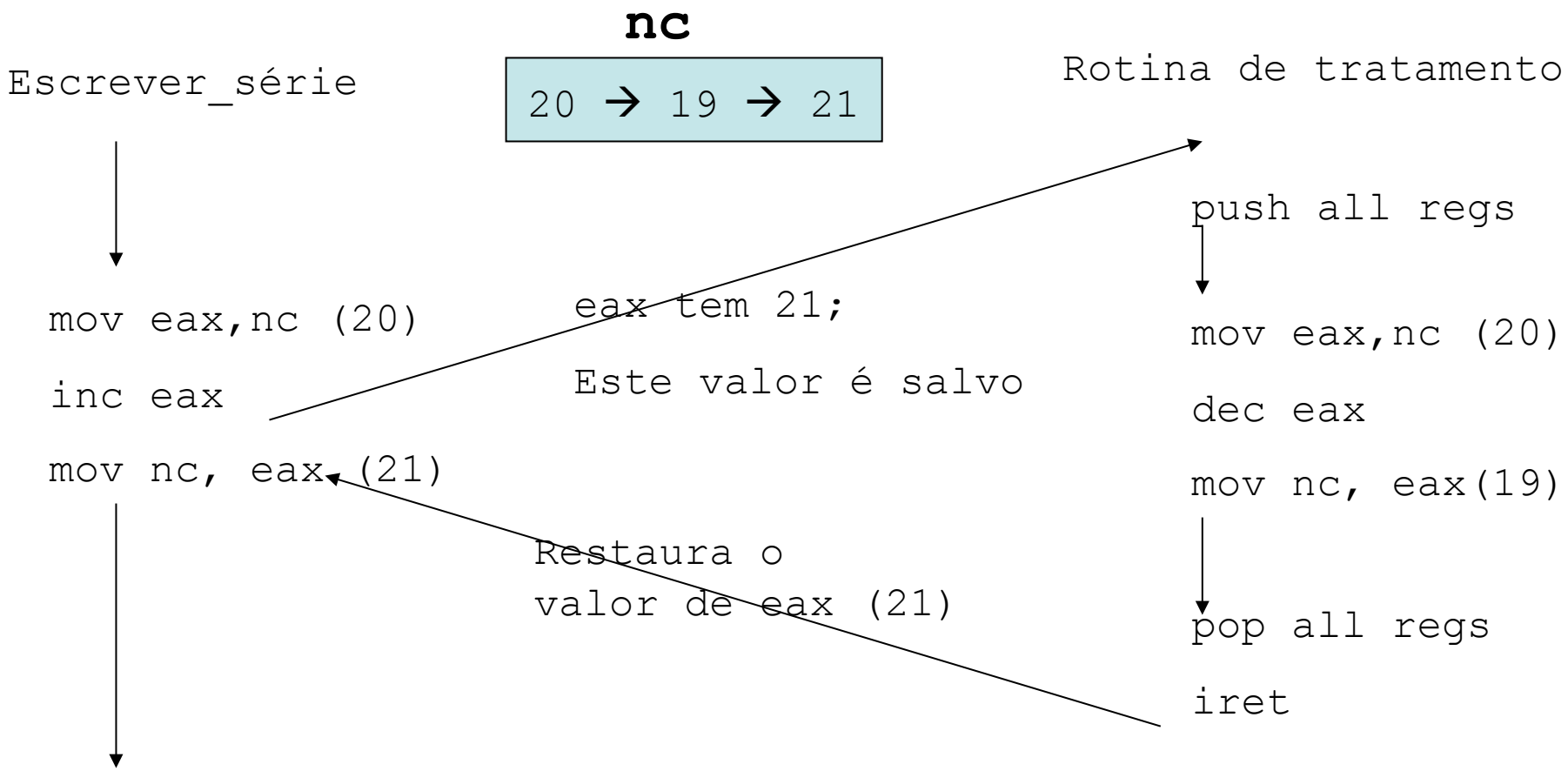
- A rotina **escrever_série** incrementa o número de bytes no buffer
  - *nc* ++
  - O compilador traduz para

    mov eax, [nc]
    inc eax
    mov [nc], eax
- A **rotina de atendimento de interrupções de transmissão** decrementa o número de bytes no buffer
  - *nc --*
  - O compilador traduz para

    mov eax, [nc]
    dec eax
    mov [nc], eax

Actualização do nº de bytes no “buffer” (2)

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



Actualização do nº de bytes no “buffer” (3)

- 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 acção de actualização da variável foi interrompida a meio; antes de voltar à actualização foi chamada uma rotina que actualiza a mesma variável
- A variável nc é partilhada pela rotina enviar_série e pela rotina de tratamento de interrupções. A sua actualização não pode ser interrompida – constitui o que se chama uma secção crítica
- Para resolver isto é preciso alterar a rotina escrever_série

```
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 actualizado correctamente

Vantagens das interrupções

- O uso de interrupções permite sobrepor (overlap) a computação e a realização de entradas/saídas
- Permitem adaptar a velocidade relativa de CPUs e periféricos
- Um computador que usa interrupções permite programar mais facilmente as operações de entrada/saída e tem mais desempenho do que um que não tem