Arquitetura de Computadores

Aula T24 – 02 Junho de 2023

Dispositivos de entrada/saída:

- Interação entre o CPU e dispositivo de entrada e saída: interrupções

Bibliografia:
OSTEP Cap. 36, secções 36.1 a 36.3

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

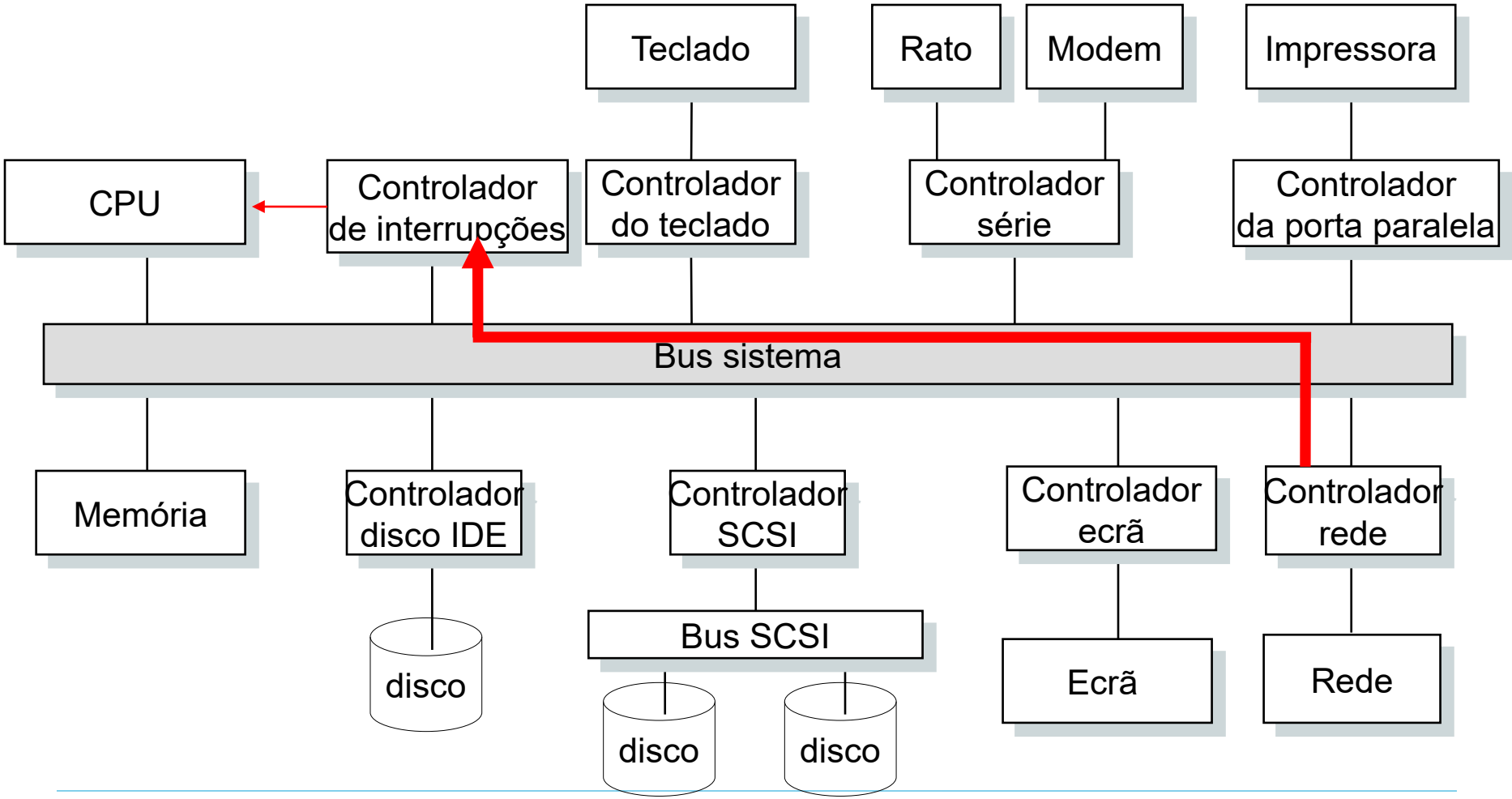
Inconvenientes do uso do “polling”

- 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 atua autonomamente depois do CPU iniciar a operação de transferência; quando esta termina o controlador envia uma interrupção ao CPU.

Interrupções assinalam conclusão da operação ou erro



Interrupções externas (assíncronas)

- Causada por eventos externos ao CPU
- Exemplos:
  - Interrupções provenientes de periféricos
    - Carregar em ctl-c at no teclado
    - Chegada de um pacote (packet) via rede
    - Chegada de um bloco (setor) do disco
    - Fim de uma transferência
  - “Soft reset”
    - Combinação ctl-alt-delete no PC

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 obtem 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

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)

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

Ciclo de execução de instruções

```
while (1){
        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)
        }
        Fetch : obter a instrução na posição de memória
                 apontada pelo PC; actualizar PC
        Execute : executar a instrução
}
```

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

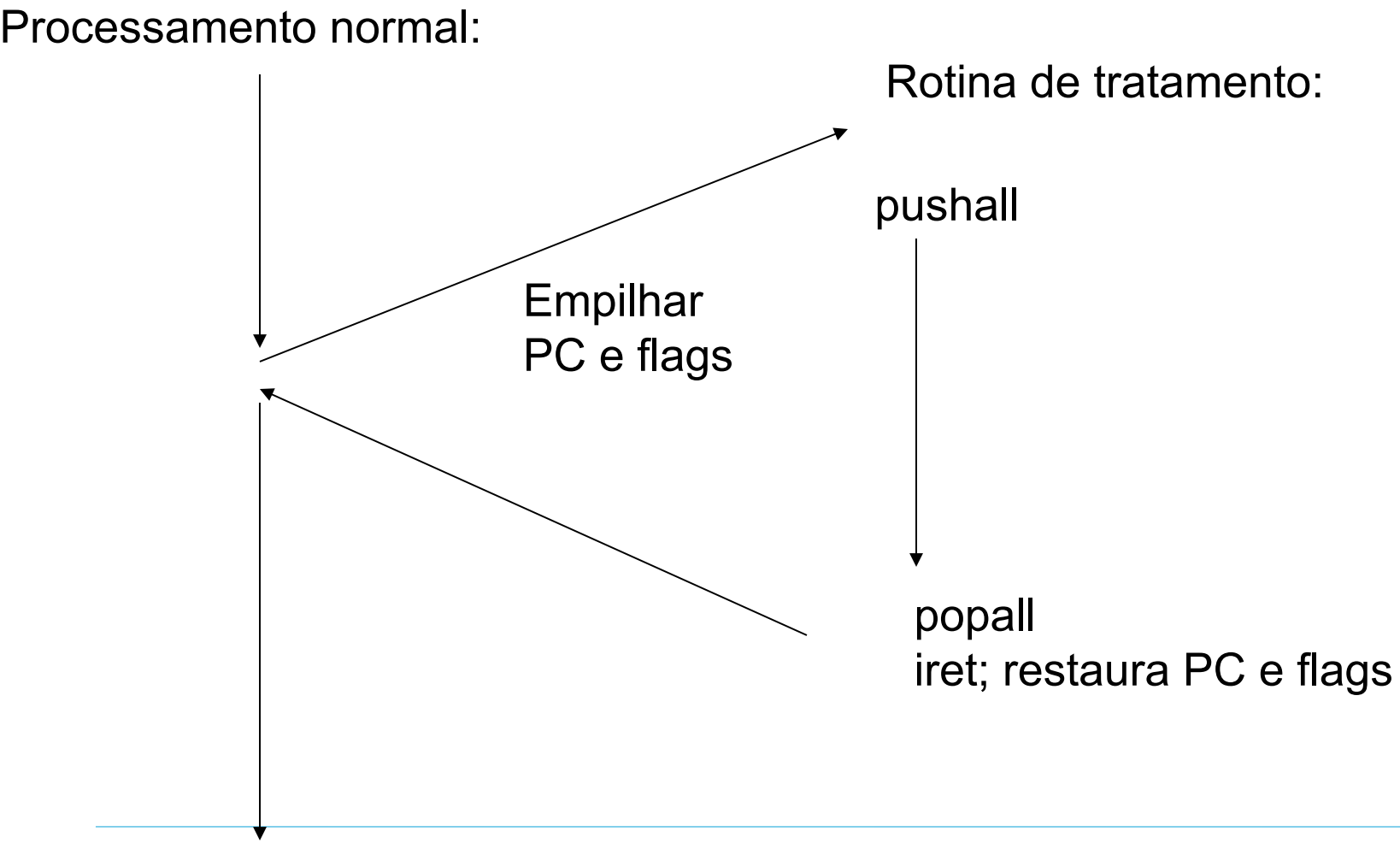
- Há uma instrução máquina que se encarrega disso:
  - Return_from_interrupt
  - Se foi empilhado primeiro o PC e depois as flags
  - A instrução faz
    - Pop flags
    - Pop PC
  - A execução continua no ponto onde se encontrava quando ocorreu a interrupção

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
```

Tratamento de uma interrupção corresponde a uma troca de contexto

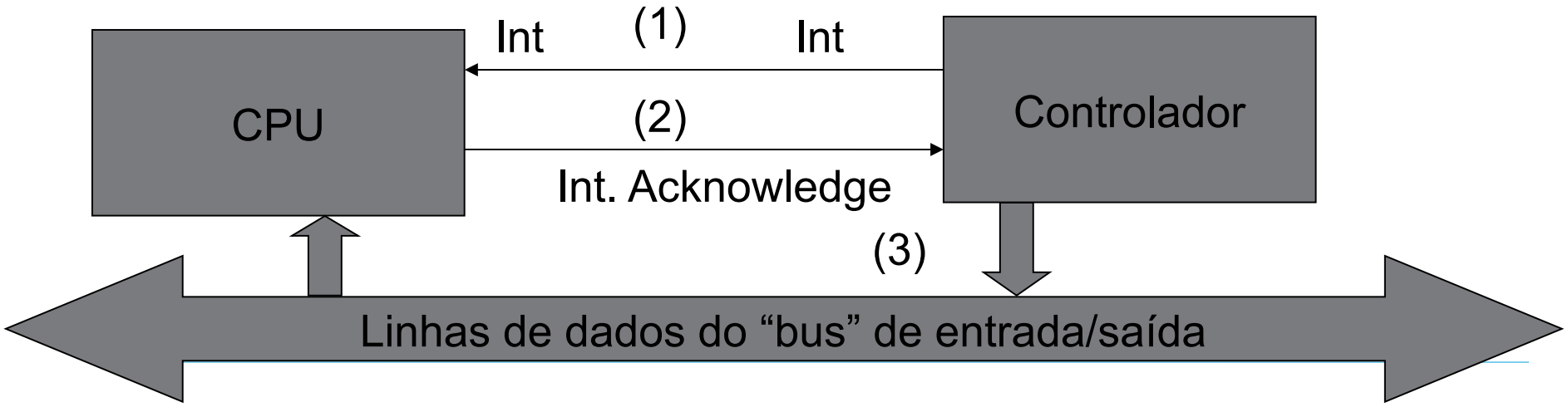


Como identificar o controlador que fez a interrupção?

- 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

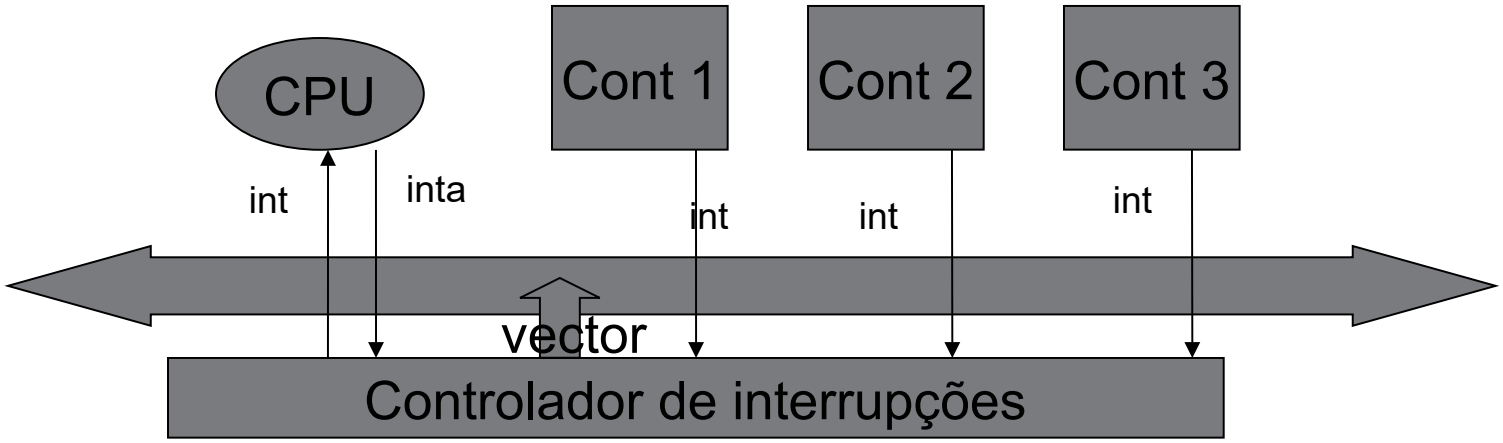
Identificação por hardware (1)

- 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 no "bus" de dados; o CPU usa esse valor para identificar a rotina de tratamento a chamar

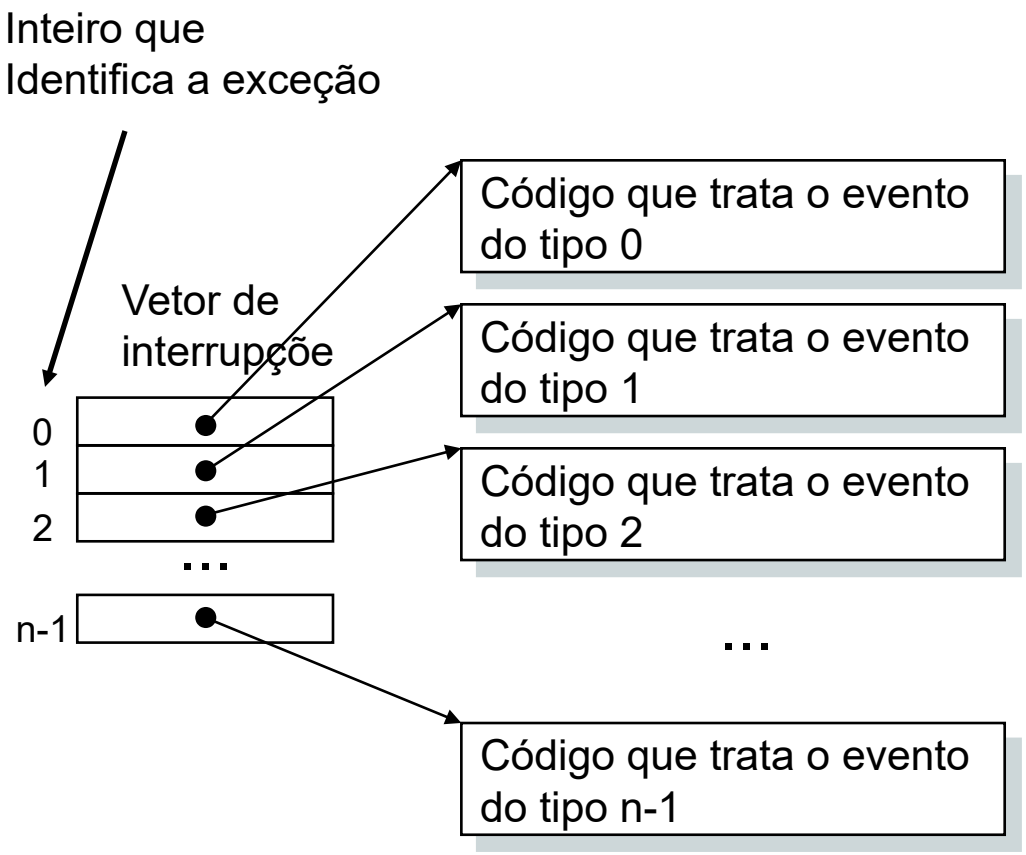


Identificação por hardware (2)

- Através de um controlador de interrupções
- Pode atribuir prioridades



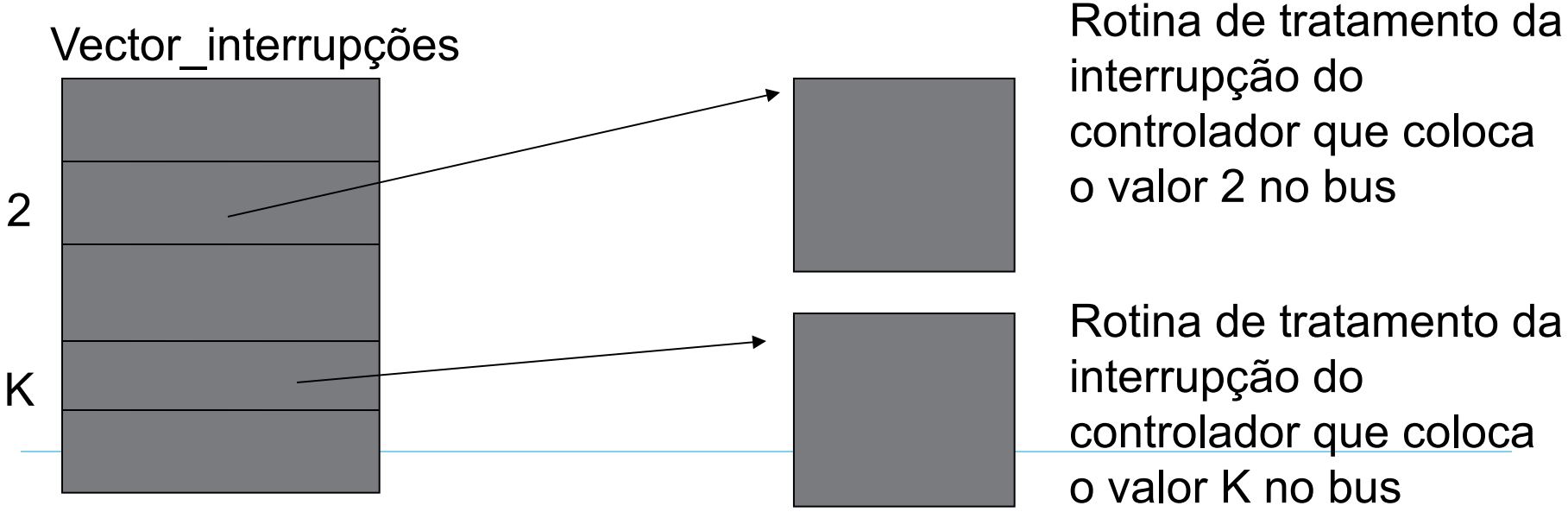
Vetor de interrupções



Cada controlador de periférico terá associado um tipo distinto

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

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 de instruções (actualização)

```
while (1){
        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)
                }
        }
        Fetch : obter a instrução na posição de memória apontada
                 pelo PC; actualizar PC
        Execute : executar a instrução
}
```

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.

Interrupções de múltiplos controladores

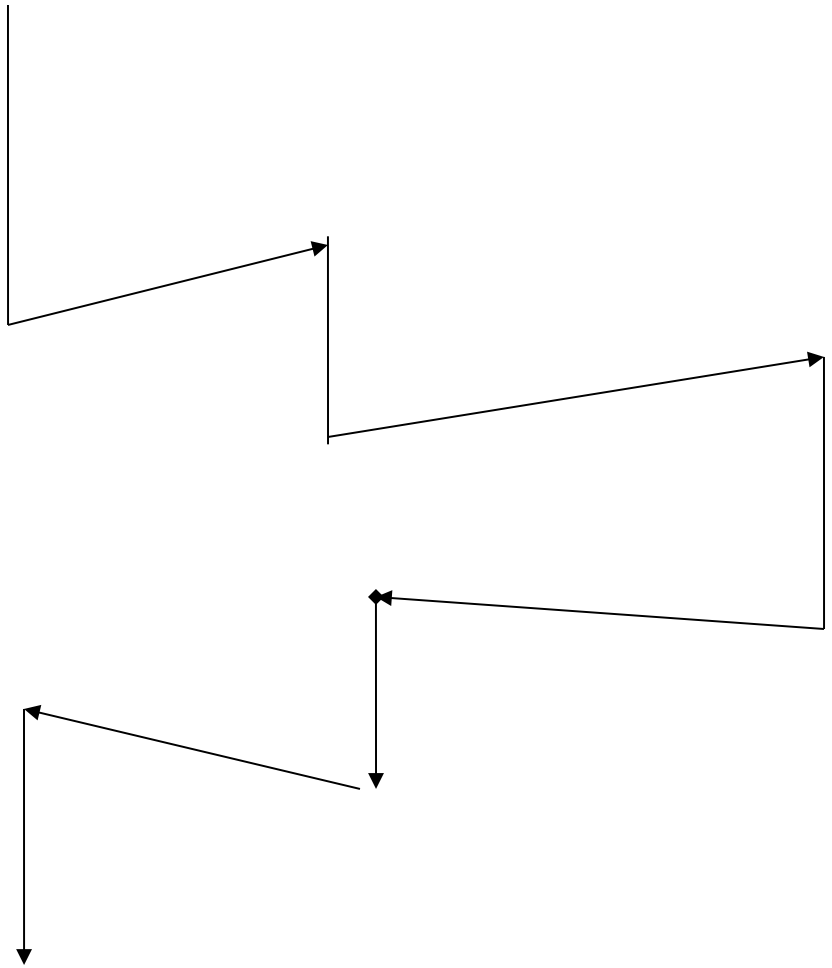
- As necessidades dos controladores diferem:
  - Tempo máximo ao fim do qual têm de receber atenção
  - Duração do processamento da interrupção
- O processamento de uma interrupção de um periférico lento pode levar mais tempo do que o tempo máximo de espera de um periférico rápido
- Para satisfazer estas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

Prioridade das interrupções (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)



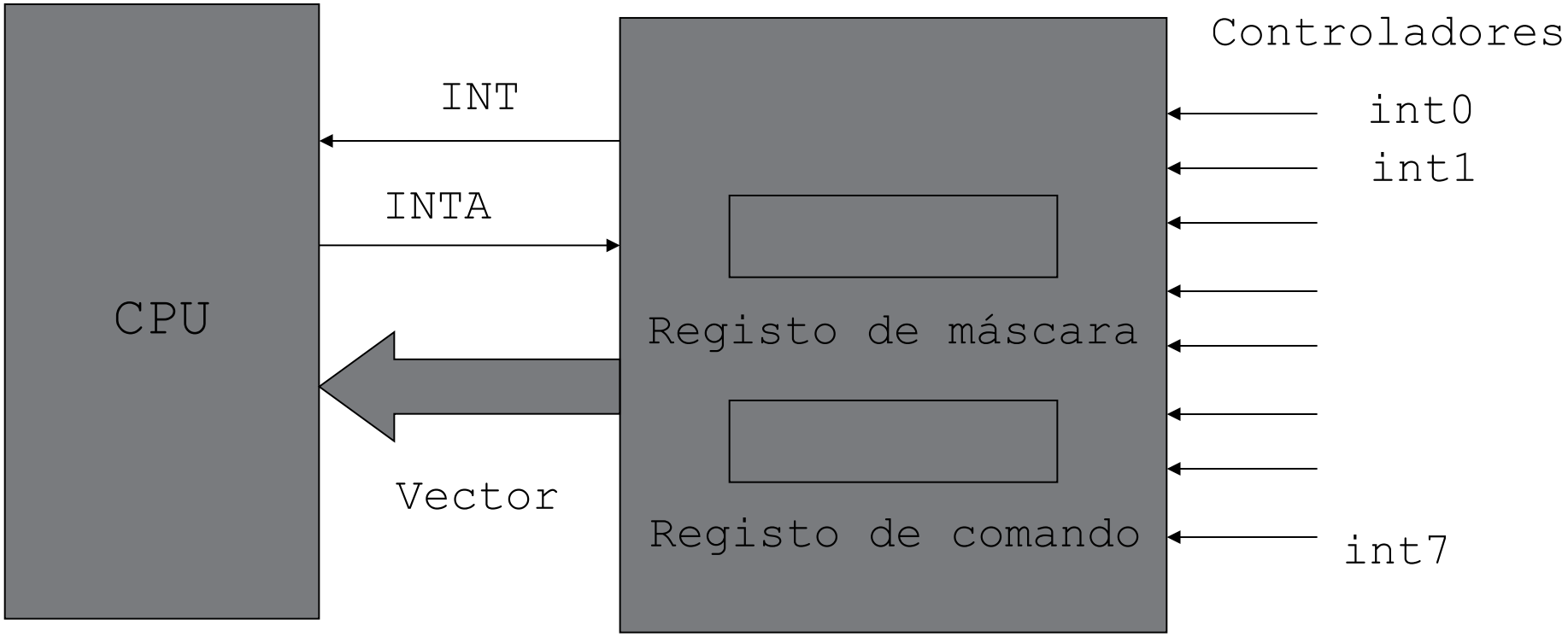
Exemplo – Bus PC

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

Sequência de eventos

- 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

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

Interrupções na UART

- COM1: - IRQ4; índice 12 do vector