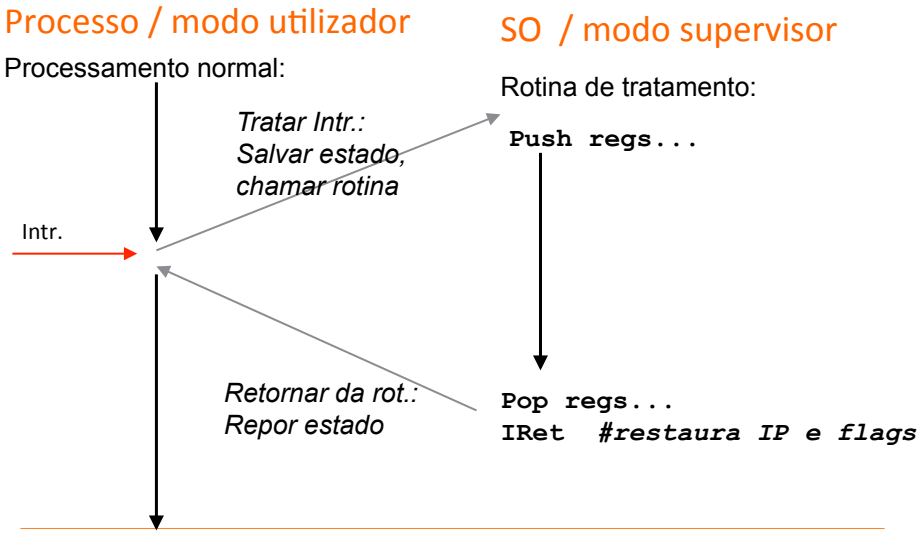
Arquitetura de Computadores

MiEI – 2015/16

DI-FCT/UNL

Aula 15

Tratamento de uma interrupção corresponde a uma troca de contexto



Salvaguarda/restauro do estado da computação

- O *hardware* só salva automaticamente o IP (ou 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
- Exemplo Intel:

```
Interrupt_handler:
            pusha     # empilha todos os regs. Gerais

            ...       # tratamento da interrupção

            popa      # restaura todos os registos
            iret       # interrupt return
```


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 do "interrupt return"
- Mas as interrupções devem estar desligadas o menor tempo possível –- podem-se perder dados que cheguem ou tornar a sua saída mais lenta.
- A rotina de tratamento deve fazer STI (IF ← 1) assim que seja seguro.

Interrupções de múltiplos controladores

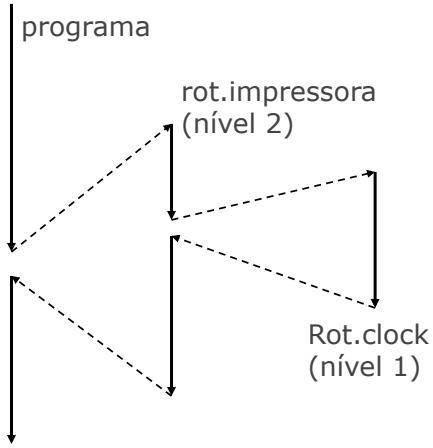
- As necessidades dos controladores diferem:
  - Tempo máximo ao fim do qual têm de receber atenção
  - Duração do processamento da interrupção
- O processamento de uma interrupção de um periférico "lento" pode levar mais tempo do que o tempo máximo entre interrupções de um periférico "rápido"
- Para satisfazer estas diferenças são atribuídas prioridades diferentes aos controladores:
  - Interrupções mais prioritárias interrompem as rotinas de tratamento das menos prioritárias

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
- A cada periférico é atribuída um nível (*N*):
  - por exemplo: *maior N* → prioridade mínima; 0 → prioridade máxima
- 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 (clock) prioridade/nível 1
  - Periférico lento (impressora) prioridade/nível 2
  - Se estamos a executar a rotina de tratamento da impressora e chega uma interrupção do clock: vai-se executar a do clock.
  - Quando esta acaba voltamos à rotina da impressora; quando esta acaba volta-se ao programa que estava a correr

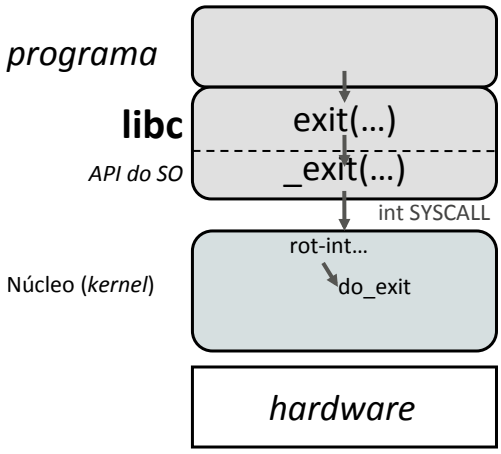


Tipos de interrupções/excepções

- **Externas ao CPU**: ligadas ao *hardware:*
  - Dispositivos de entrada/saída (I/O)
  - Temporizador (clock)
  - Falha no *hardware*
- **Internas ao CPU**: ligadas ao *software* (devido à instrução executada)
  - **Interrupção por software:** existe uma instrução máquina que provoca deliberadamente uma interrupção: **INT**
    - Pode ser usada, p.e., para fazer chamadas ao SO
  - **Excepções** (exemplos: divisão por 0, *overflow*, acesso a memória não autorizada, execução de instrução privilegiada em modo utilizador, ...)
  - Não mascaráveis pela IF

Chamada ao sistema `_exit`

Libc: `void exit(int status);`



Código possível

```
_exit:
   ...
   mov $EXIT, %eax
   mov 8(%ebp), %ebx
   int $SYSCALL
   ...
```

No caso do linux/x86:
EXIT=1
SYSCALL=0x80

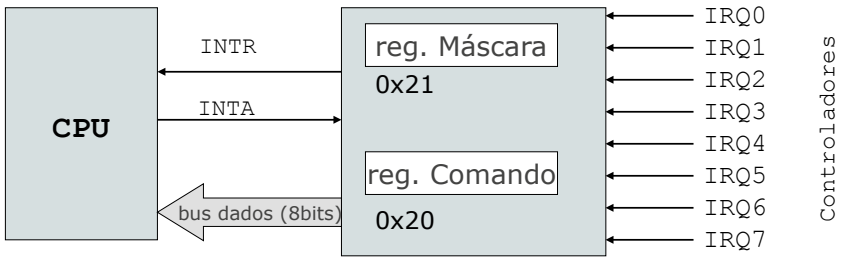
Recapitulando

- O uso de interrupções permite sobrepor (*overlap*) a computação e a realização de entradas/saídas pelos periféricos
- Permitem adaptar a velocidade relativa de CPUs e periféricos, tirando melhor partido do CPU e dos periféricos
- Permite estar atento e tratar várias situações em simultâneo, sem ter de as verificar e sem correr o risco de as não detetar
- Um computador com interrupções permite ter melhor desempenho do que um que não tem

Exemplo – Intel nos PC

- Suporta até 255 tipos de interrupções vetorizadas
  - o vetor de endereços está na memória
- Tem uma linha de interrupções gerais externas
  - a flag IF é alterada pelas instruções STI e CLI
- Usa um controlador de interrupções programável -- PIC (8259A ou equivalente)
  - Identifica o tipo de cada interrupção, depois usado como índice no vetor de interrupções
  - Define as prioridades entre interrupções
- Instrução de interrupção por software: INT nº
- O retorno da rotina de tratamento é feito pela instrução IRET

Controlador 8259A (PIC)



- Registo de máscara: cada bit *i* a 1 indica que a linha IRQ*i* está mascarada (é ignorada)
- Cada IRQ tem associado no PIC um número de identificação
- Registo de comando: permite programar a forma de atendimento, os identificadores, etc.
  - comando: *End of Interrupt* (EOI) informa o PIC que a rotina de atendimento acabou (código 0x20)

Máscara de IRQs no PIC

- Registo de máscara, indica IRQ ignorados pelo PIC
  - 8 bits → se 1, esse nível (IRQ) está desligado; se 0, esse nível está ligado
  - Exemplos:
    - Desligar o nível IRQ3

```
cli
in 0x21, %al
or 0b00001000, %al
out %al, 0x21
sti
```

    - Ligar o nível IRQ4

```
cli
in 0x21, %al
and 0b11101111, %al
out %al, 0x21
sti
```


Prioridades no PIC

- Cada IRQ é um nível de prioridade de interrupções:
  - IRQ 0 mais prioritário; IRQ 7 menos prioritário
- Quando uma interrupção é aceite pelo CPU todas as IRQ igual ou de maior nível ficam pendentes no PIC
- Cada rotina de tratamento deve terminar com
  - o comando EOI: `out 0x20, 0x20`
  - Para indicar que todos os IRQ podem voltar a ser atendidos
  - Exemplo:

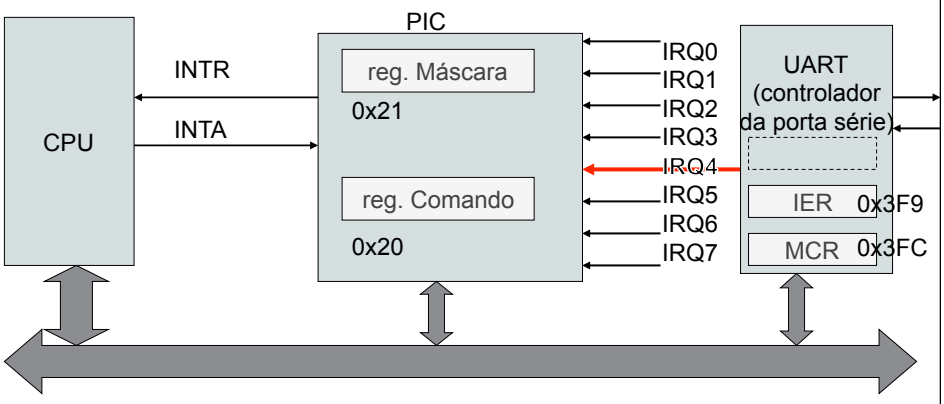
```
RotAtendimento:
                sti
                  ... # tratamento int
                out 0x20, 0x20
                iret
```

Sequência de eventos

- Periférico envia interrupção
- 8259A aceita (ou não) a interrupção
- 8259A determina a prioridade
  - Se estiver em curso um tratamento de prioridade igual ou superior este fica pendente
- 8259A alerta o CPU (ativa o pino INTR)
- CPU faz "acknowledge" (INTA)
- 8259A coloca o identificador correspondente no bus de dados
- CPU salta para a rotina de tratamento de interrupções indicada por: `vetorIntr[identificador]`

Interrupções na Porta Série 1

- COM1:
  - tem atribuído o IRQ4; identificação 12



- Revisões