

A photograph of a server room with rows of black server racks. A semi-transparent blue rectangle is overlaid on the right side of the image, containing white text. The text includes a main title, chapter information, and section information.

Periféricos e interrupções

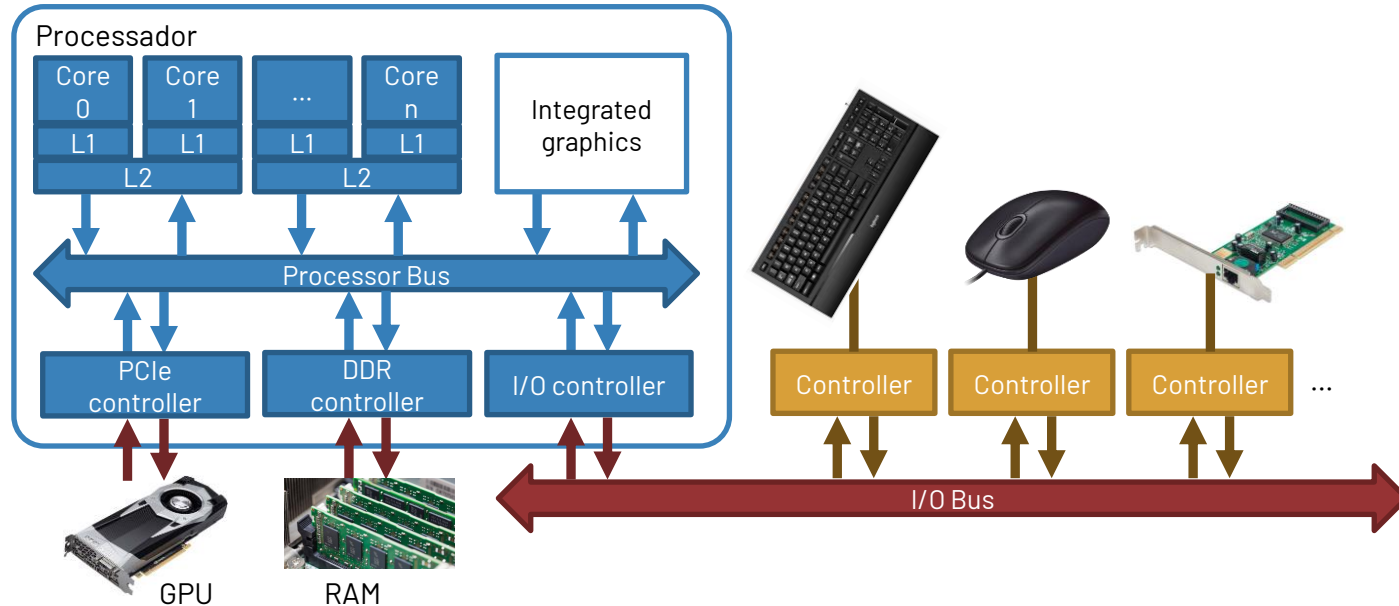
Cap. 2: “Instructions: Language of the Computer”
Sec. 4.9: “Exceptions”, pp 315-317



Periféricos e interrupções

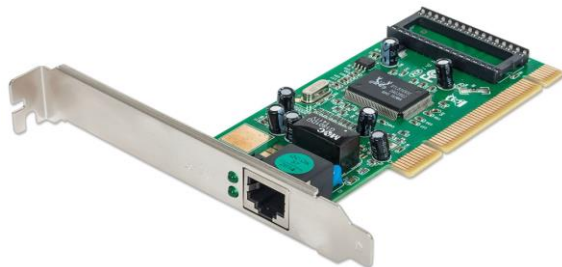
Visão geral

- O processador não é uma entidade isolada.



- Cada periférico pode ser simplesmente visto como um conjunto de registos/memória, que permitem controlar o periférico
 - Diferentes periféricos têm diferentes números de registos
 - A largura dos registos (número de bits) depende do periférico

Exemplo:



R0	
R1	
R2	
R3	
R4	
R5	
R6	
R7	

Registos do periférico

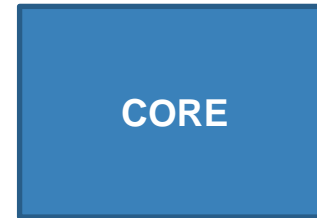
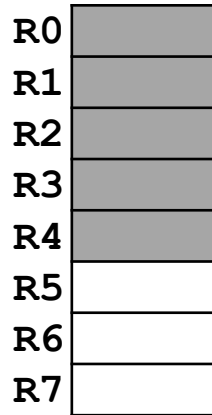
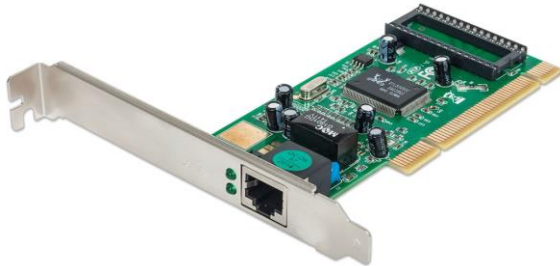
(não confundir com os registos do processador)

Visão geral de um periférico

Configuração do periférico

- Num primeiro estágio, o processador configura o modo de funcionamento do periférico, escrevendo em registos específicos

Exemplo:



Registos do periférico

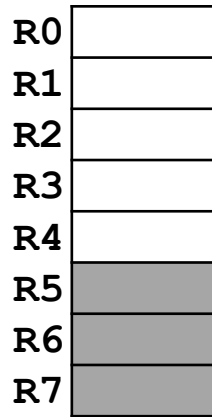
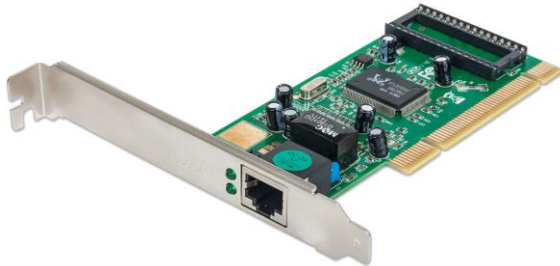
(não confundir com os registos do processador)

Visão geral de um periférico

Leitura e escrita de dados no periférico

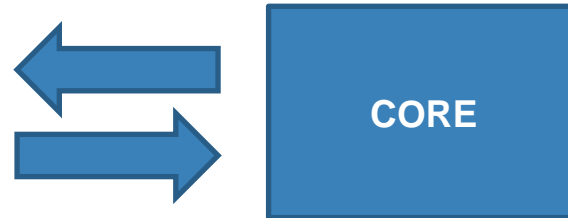
- Num segundo estágio, o processador lê e/ou escreve dados do periférico
 - Em qualquer altura, o core pode alterar a configuração do periférico, mudando a forma de operação

Exemplo:



Registos do periférico

(não confundir com os registos do processador)

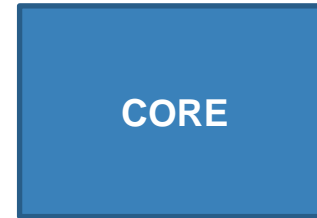
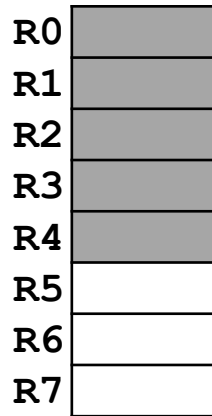
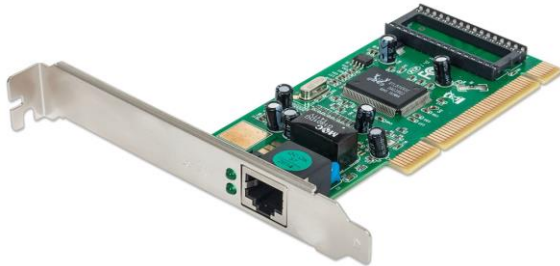


Visão geral de um periférico

Leitura e escrita de dados no periférico

- Em qualquer altura, o core pode alterar a configuração do periférico, mudando a forma de operação

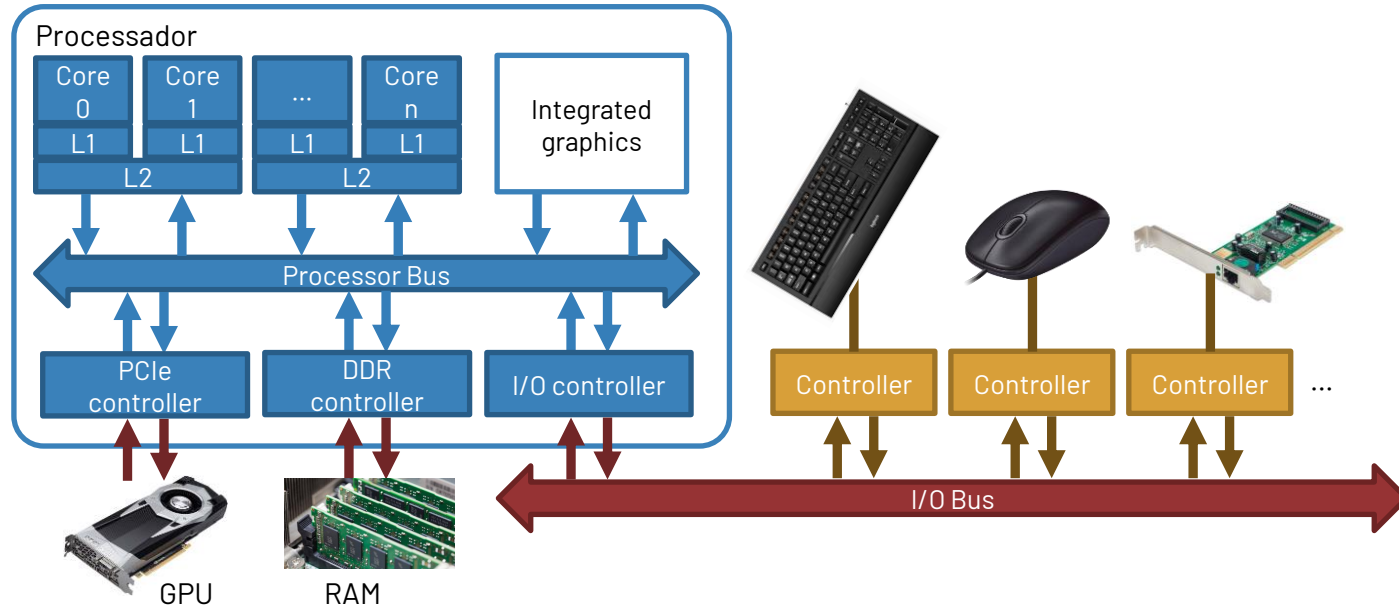
Exemplo:



Registos do periférico

(não confundir com os registos do processador)

- Como comunicar com os periféricos?





Comunicação com periféricos

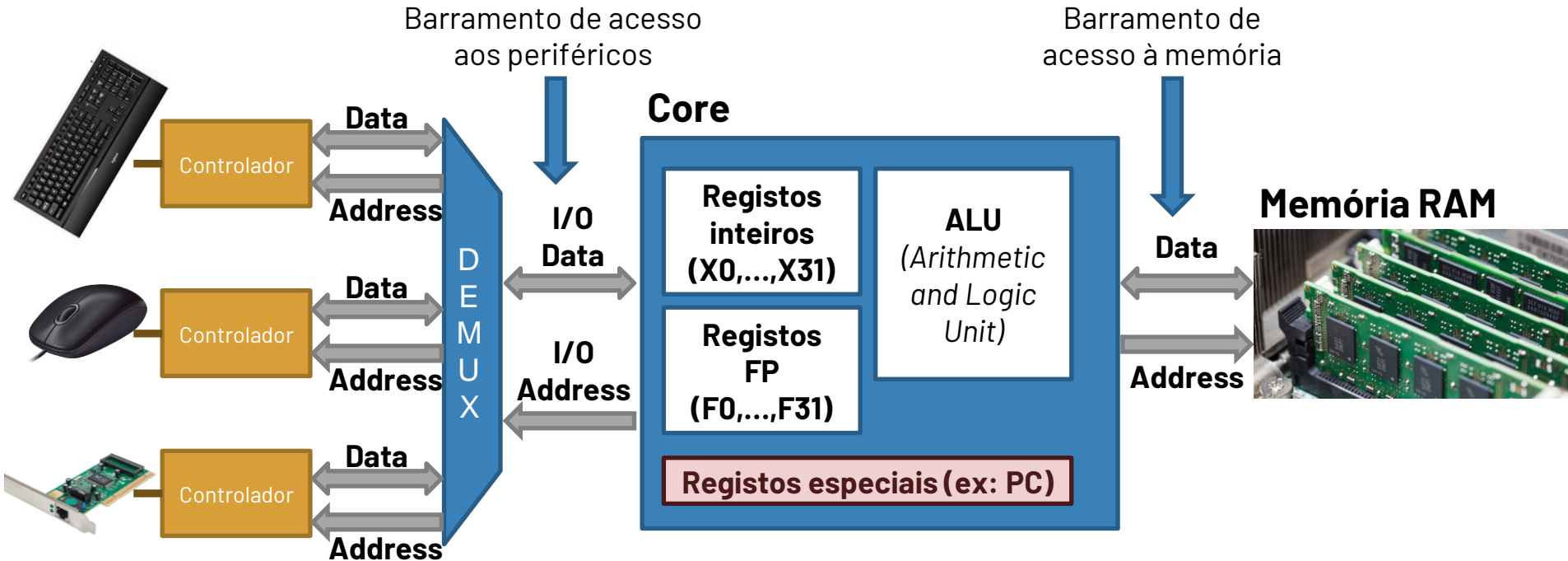
Solução #1

Port-Mapped I/O

Barramentos independentes de acesso à memória e aos periféricos

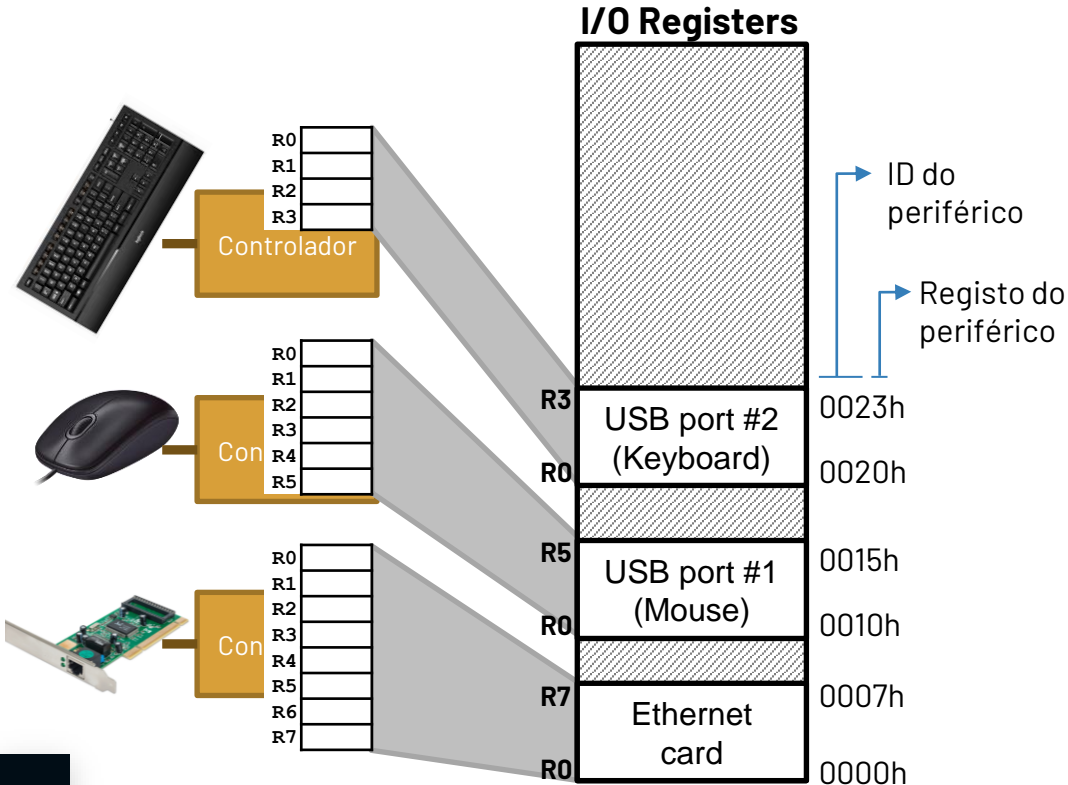
Comunicação com periféricos: port-mapped I/O

Arquitetura da solução



Comunicação com periféricos: port-mapped I/O

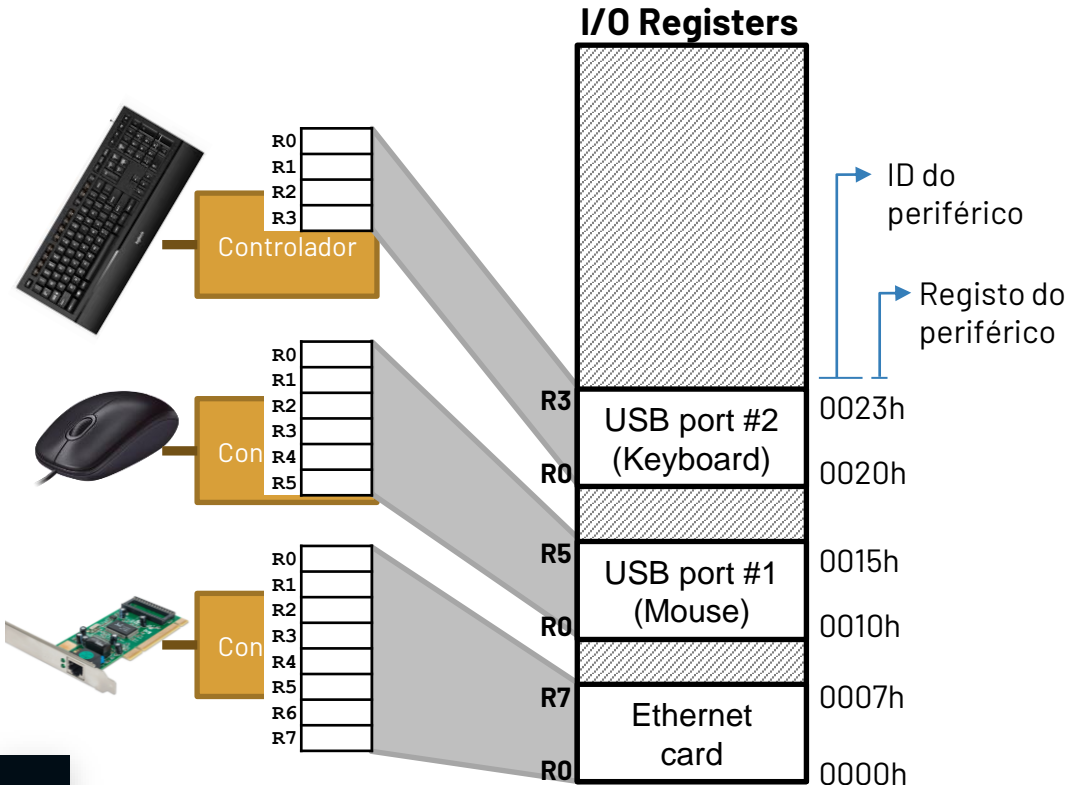
Mapeamento dos registos dos periféricos



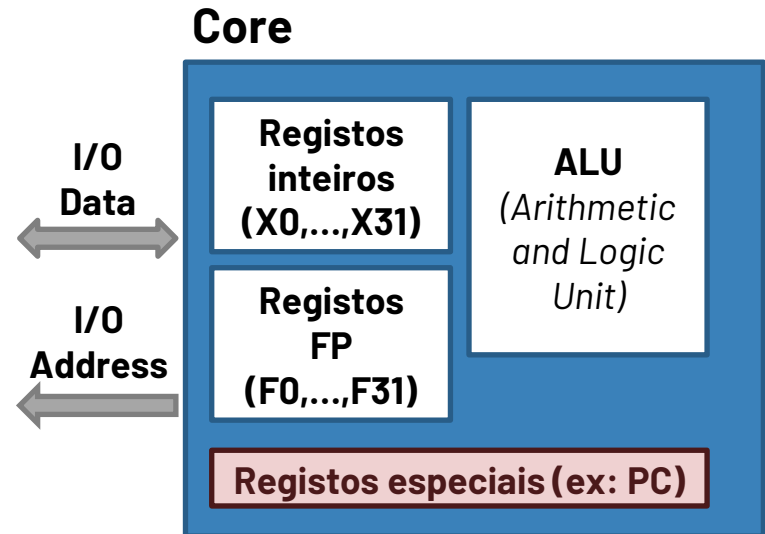
- Os registos dos periféricos são mapeados numa gama de endereços específica
- Neste exemplo, os 4 bits menos significativos do endereço permitem indicar o registo do periférico - os restantes bits indicam o periférico
- Diferentes implementações usam diferentes mapeamentos

Comunicação com periféricos: port-mapped I/O

Acesso aos registos dos periféricos



- A leitura/escrita no periférico é realizada indicando o endereço do registo pretendido



Comunicação com periféricos

Alterações ao ISA (*Instruction Set Architecture*)

- Nesta solução existem barramentos independentes:

(1) Acesso a dados na memória

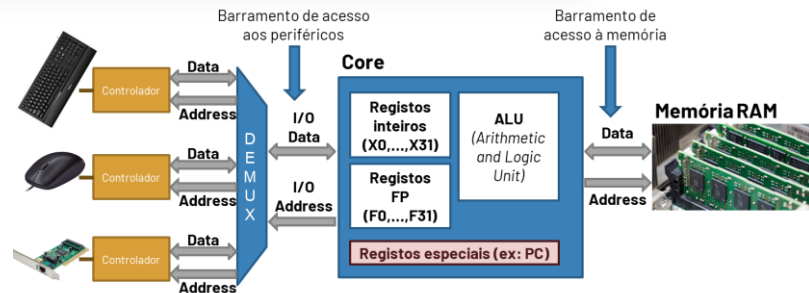
Realizado através de instruções convencionais de leitura e escrita na memória, ex: `lb, lh, lw, ...` ou `sb, sh, sw, ...`

(2) Acesso (leitura/escrita) de dados nos periféricos

Realizado através de instruções específicas de acesso aos periféricos, ex:

`iolw xn, xpto` (Leitura do registo de I/O com endereço `xpto` e escrita no registo de uso geral `Xn`)

`iosw xn, xpto` (Escrita do valor no registo de uso geral `Xn` no registo de I/O com endereço `xpto`)



Comunicação com periféricos

Alterações ao ISA (*Instruction Set Architecture*)

- Nesta solução existem barramentos independentes:

(1) Acesso a dados na memória

Realizado através de instruções convencionais de leitura e escrita na memória, ex: `lb, lh, lw, ...` ou `sb, sh, sw, ...`

(2) Acesso (leitura/escrita) de dados nos periféricos

Realizado através de instruções específicas de acesso aos periféricos, ex:

`iolw` `xn, xpto`

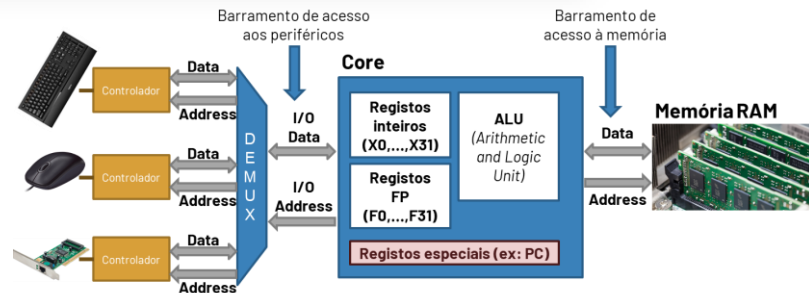
No ISA da Intel a mnemónica desta instrução é `IN`

registro de uso geral Xn)

`iosw` `xn, xpto`

No ISA da Intel a mnemónica desta instrução é `OUT`

I/O com endereço xpto)





Comunicação com periféricos

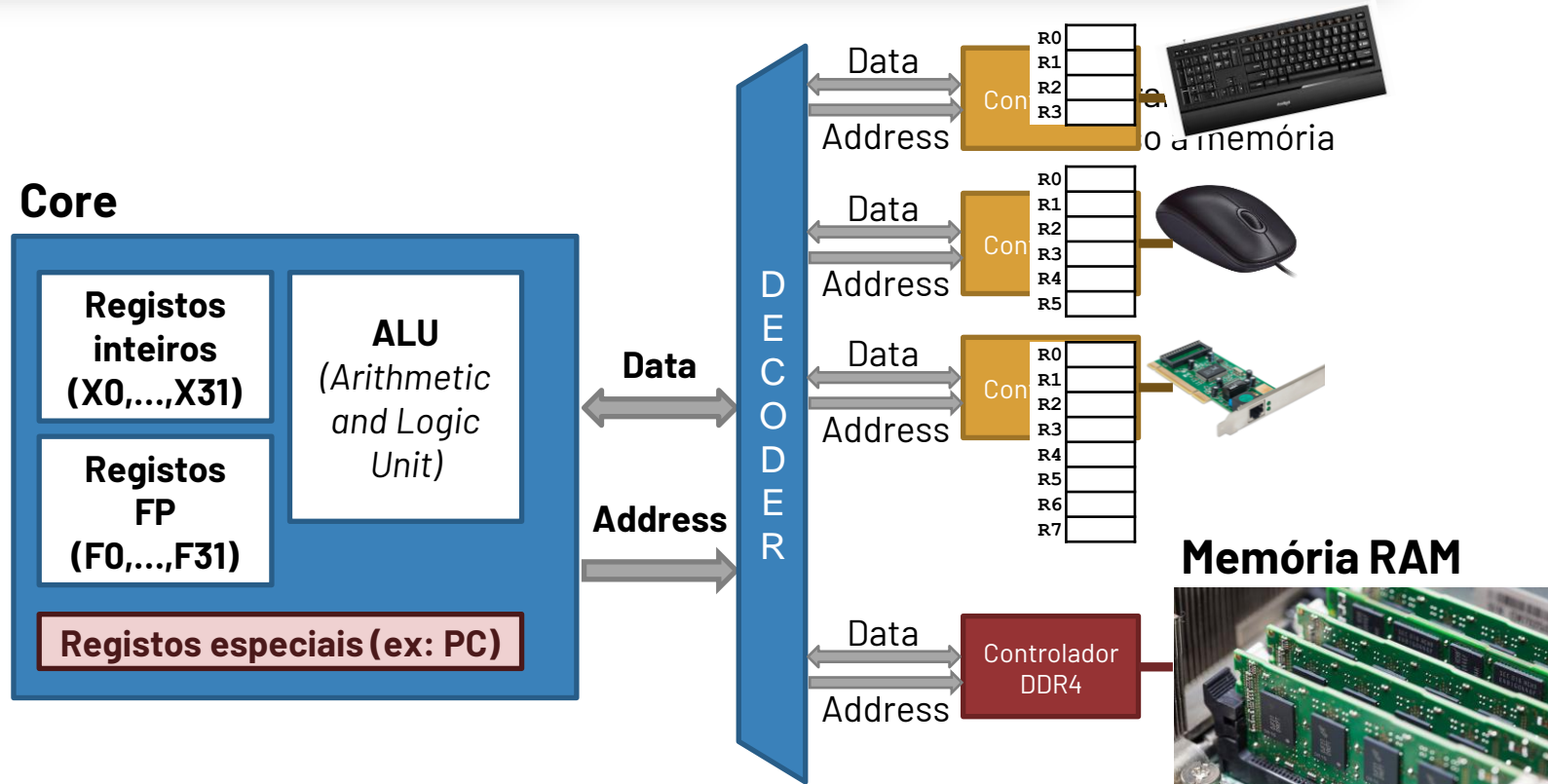
Solução #2:

Memory-Mapped I/O

Barramento único para acesso à memória e aos periféricos

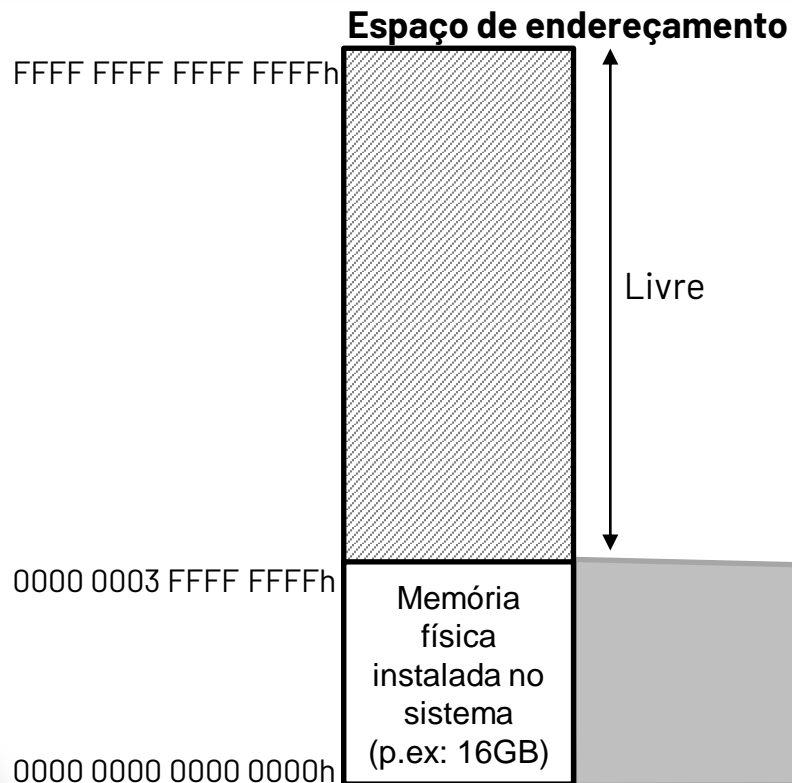
Comunicação com periféricos: memory-mapped I/O

Arquitetura da solução



Comunicação com periféricos: memory-mapped I/O

Mapeamento dos registos no espaço de endereçamento



Raramente o espaço de endereçamento é totalmente usado!!!

Podemos utilizar parte do espaço de endereçamento para mapear os registos dos periféricos

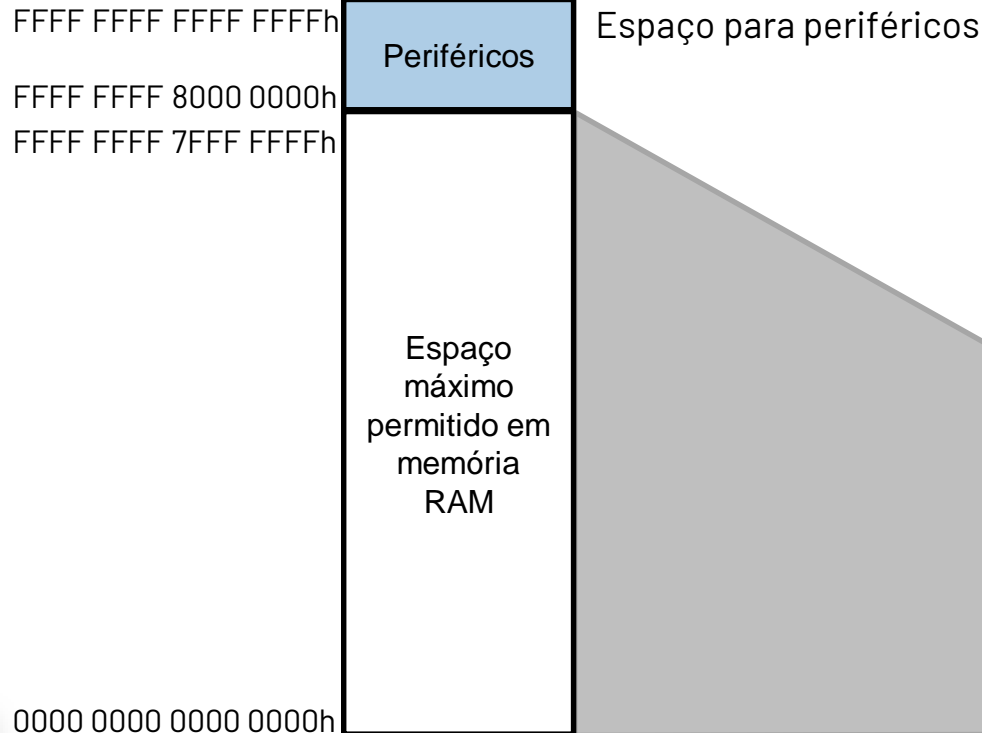
Memória RAM



Comunicação com periféricos: memory-mapped I/O

Mapeamento dos registos no espaço de endereçamento

Divisão do Espaço de endereçamento



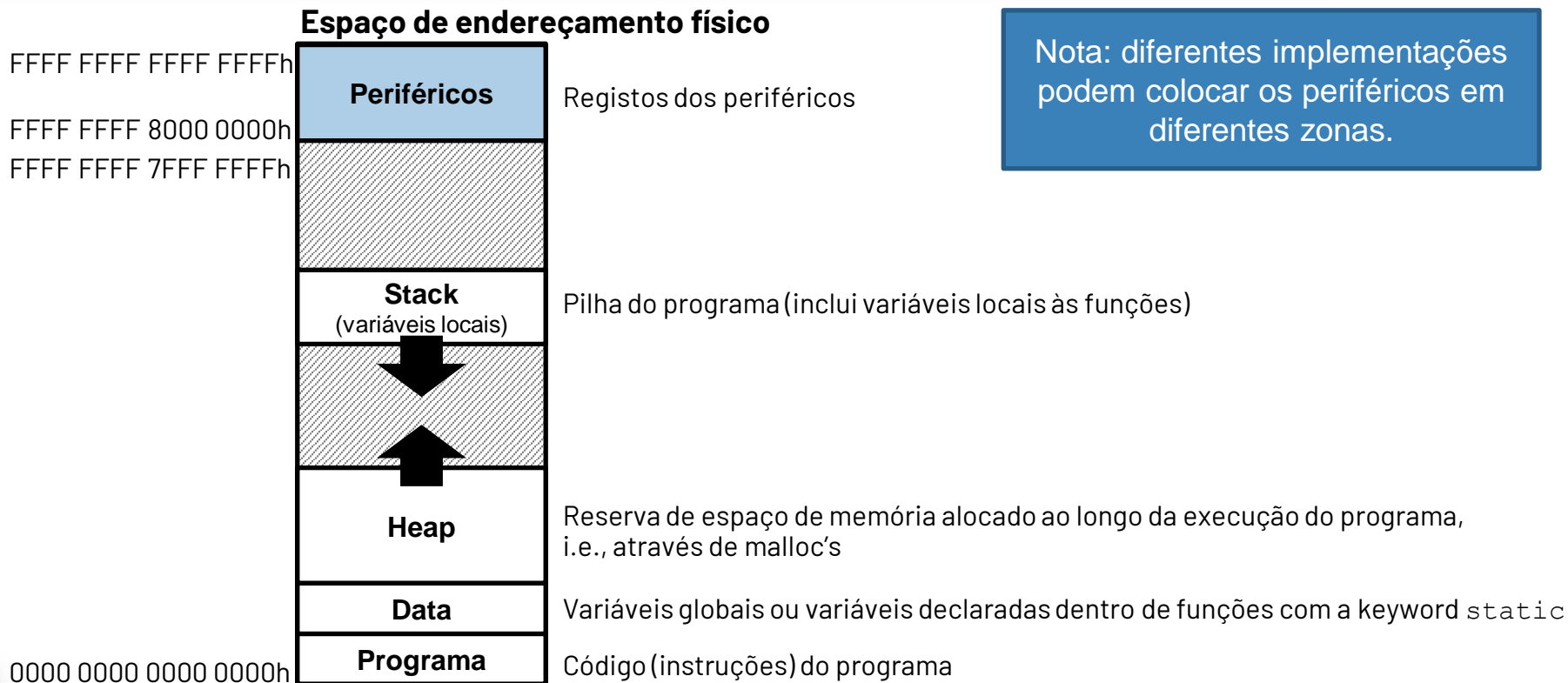
Implementações diferentes podem mapear os periféricos em zonas diferentes do espaço de endereçamento

Memória RAM



Comunicação com periféricos: memory-mapped I/O

Mapeamento dos registos no espaço de endereçamento



Nota: diferentes implementações podem colocar os periféricos em diferentes zonas.

Comunicação com periféricos: memory-mapped I/O

Leitura e escrita nos registos dos periféricos

- Como nesta solução existe um único barramento que permite aceder a dados e a periféricos, podemos usar as instruções convencionais para aceder aos periféricos:

Acesso a dados na memória e leitura e escrita nos registos dos periféricos

Realizado através de instruções convencionais de leitura e escrita na memória, ex:
lb, lh, lw, ... ou sb, sh, sw, ...

O ARMv8 e o RISC-V usam *memory-mapped I/O*, embora o mapeamento dos periféricos não seja exatamente o descrito nos slides anteriores.



Interrupções e exceções

Visão geral

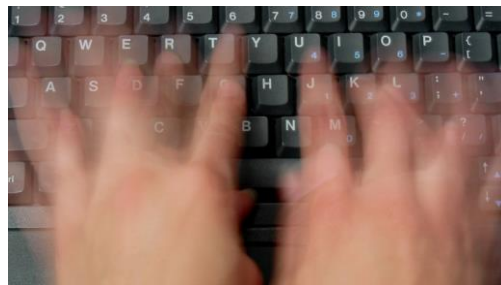
Definição de interrupção e exceção

- Considere, a título de exemplo, um dos periféricos de entrada mais comuns: o **teclado**
- Um utilizador experiente pressiona, em média, cerca de 10 teclas por segundo.



Definição de interrupção e exceção

- Considere, a título de exemplo, um dos periféricos de entrada mais comuns: o **teclado**
- Um utilizador experiente pressiona, em média, cerca de 10 teclas por segundo.



- Ou seja: o periférico (teclado) tem uma nova tecla pronta a ser lida pela processador a cada $0,1 = 10^{-1}$ segundos.
- Assumindo que o processador executa a uma frequência de 2 GHz, podemos estimar que cada instrução demora cerca de $1/(2 \times 10^9) = 0,5 \times 10^{-9} = 0,5 \text{ ns}$
- Assim sendo, o processador terá de **esperar** (sem fazer mais nada útil!) cerca de $10^{-1}/(0,5 \times 10^{-9}) = 2 \times 10^8 = 200$ milhões de ciclos de relógio, entre cada tecla pressionada pelo utilizador.

Entretanto... não há mais nada para fazer???

Definição de interrupção e exceção

- Assuma que, para curar uma infecção, uma pessoa tem de tomar um antibiótico de 8 em 8 horas: às 12h (almoço), às 20h (jantar) e às 4h (a meio da noite).
- Como fazer para garantir a toma das 4h (a meio da noite)?



Definição de interrupção e exceção

- Assuma que, para curar uma infecção, uma pessoa tem de tomar um antibiótico de 8 em 8 horas: às 12h (almoço), às 20h (jantar) e às 4h (a meio da noite).
- Como fazer para garantir a toma das 4h (a meio da noite)?
- Podemos ficar acordados, com atenção ao relógio, à espera da hora da toma...



Definição de interrupção e exceção

- Assuma que, para curar uma infeção, uma pessoa tem de tomar um antibiótico de 8 em 8 horas: às 12h (almoço), às 20h (jantar) e às 4h (a meio da noite).
- Como fazer para garantir a toma das 4h (a meio da noite)?
- Uma alternativa melhor será optarmos por fazer outra coisa mais útil (ex: dormir!) e pedir a alguém (ex: despertador) que nos avise que está na hora da toma



... ou seja...

... que nos **interrompa** a atividade em curso (dormir) para fazermos **outra** atividade **mais urgente**

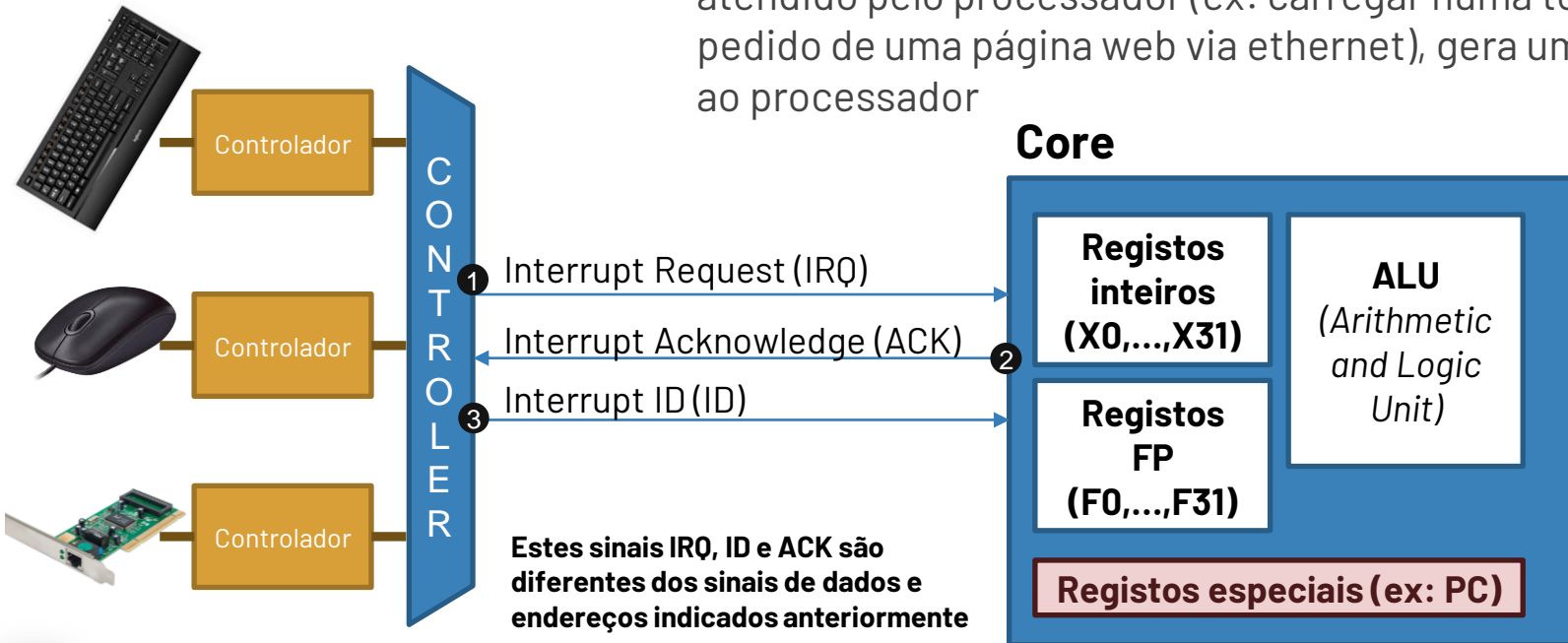
 **INTERRUPÇÃO**

- Embora a definição não seja consensual, geralmente denomina-se por:
 - **Interrupção**
Evento gerado por uma entidade externa ao processador (ex: um periférico), que requer um processamento especial, e muitas vezes urgente:
 - O utilizador carregou no teclado ou mexeu o rato
 - O utilizador inseriu uma pen USB
 - O servidor recebeu um pedido via ethernet (ex: pedido de uma página web)
 - **Exceção**
Evento gerado em consequência da execução de uma instrução
 - Divisão por zero
 - Execução de uma instrução ilegal (ex: a instrução não existe ou não é permitida neste contexto)
 - Chamada a uma rotina de SO

Interrupções

Arquitetura do sistema

- Quando um periférico gera um evento que tem de ser atendido pelo processador (ex: carregar numa tecla, ou um pedido de uma página web via ethernet), gera um pedido ao processador



Tratamento de interrupções e exceções

Funcionamento: visão geral

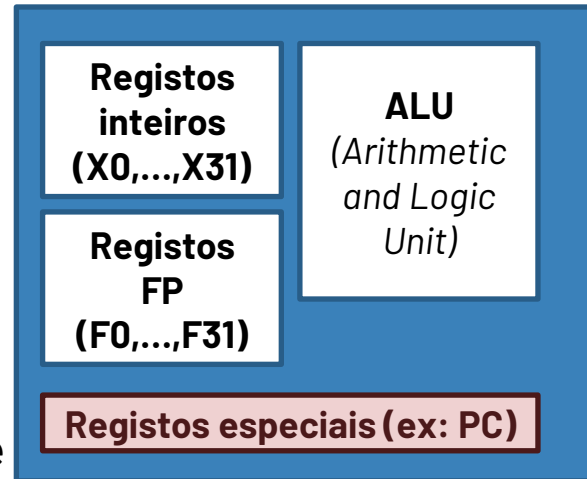
```
la    x5, V0
la    x6, V1
la    x7, Res
PC → lw    x10, 0(x5)
lw    x11, 0(x6)
jal   x1, avg
sw    x10, 0(x7)
...

avg:  add    x10, x10, x11
      srai   x10, x10, 1
      jalr   x0, x1, 0
```

- O processador interrompe a normal execução do código para tratar a interrupção/exceção
 1. Termina a execução da instrução atual (`lw x10, 0(x5)`)
 2. Chama a sub-rotina que atende a interrupção/exceção
 3. Retorna ao ponto inicial, executando a instrução seguinte (`lw x11, 0(x6)`)

Na prática, o tratamento da interrupção/exceção pode ser visto como uma chamada a uma rotina específica para tratamento de interrupções.

Contudo, neste caso não existe uma chamada explícita no código à rotina de tratamento de interrupções/exceções, nem é conhecido o ponto do programa onde a rotina é chamada.



Core

PC →

```
la    x5, V0
la    x6, V1
la    x7, Res
lw    x10, 0(x5)
lw    x11, 0(x6)
jal   x1, avg
sw    x10, 0(x7)
...

avg:  add    x10, x10, x11
      srai   x10, x10, 1
      jalr   x0, x1, 0
```

- Em alguns casos, a ocorrência de uma exceção impede a normal execução da instrução
- Nesses casos a instrução que está a executar não é terminada, mas suspensa. Assim, o retorno da rotina de tratamento da exceção não é a instrução seguinte, mas a própria instrução.
 - Uma melhor definição seria:

Após o tratamento da rotina de interrupção/exceção, o core retorna à ultima instrução não terminada

- Em alguns casos a ocorrência de uma exceção pode levar à terminação do programa. Exemplos:
 - **"Illegal exception"** – tentativa de execução de uma instrução não existente, não suportada, ou não permitida
 - **"Segmentation fault"** – tentativa de acesso a uma zona de dados que ainda não foi alocado ao processo

Tratamento de interrupções e exceções

Funcionamento: visão geral

PC →

```
la    x5, V0
la    x6, V1
la    x7, Res
lw    x10, 0(x5)
lw    x11, 0(x6)
jal   x1, avg
sw    x10, 0(x7)
...
```

```
avg:  add    x10, x10, x11
      srai   x10, x10, 1
      jalr   x0, x1, 0
```

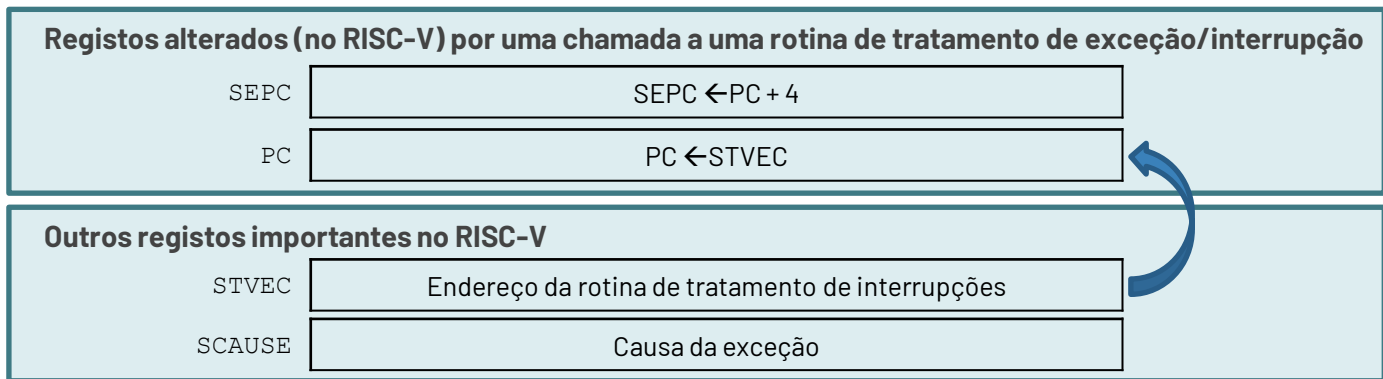
- A execução da rotina que trata a interrupção/exceção deve ser transparente e (em geral) não interferir com a execução do código principal. Assim, esta não deve:
 1. Alterar qualquer registo do processador
 2. Alterar a stack (não recebe ou devolve qualquer valor)

Nota: em alguns casos esta regra pode não ser cumprida. Por exemplo: a execução de uma instrução ilegal gera uma exceção que termina a execução do programa

Tratamento de interrupções e exceções

Chamada à rotina de tratamento de interrupções/exceções

- Assim, a chamada à rotina que atende a interrupção deve:
 1. **Guardar o endereço de retorno**
No caso do RISC-V, o endereço de retorno é guardado no *Supervisor Exception Program Counter (SEPC)*
 2. **Colocar o PC a apontar para a primeira instrução da rotina de tratamento de interrupções**
No caso do RISC-V, o endereço da rotina de tratamento de interrupções é dado pelo registo *Supervisor Trap Vector Base Address Register (STVEC)*



O registo SCAUSE guarda a causa da interrupção/excepção (e.g., FP divide by zero, exceção gerada pelo utilizador com as instruções ECALL/EBREAK, interrupção externa causada por teclado, rato, USB, etc.)

Tratamento de interrupções e exceções

Estrutura da rotina de tratamento de interrupções/exceções

- A rotina de tratamento de interrupções deve:
 1. Salvar na pilha o valor de *TODOS* os registos modificados pela rotina de tratamento de interrupções
 - A convenção do compilador indicada no *RISC-V Reference Guide* não se aplica nas rotinas de tratamento de interrupções/exceções
 2. Verificar a causa da interrupção/exceção, consultado o registo SCAUSE
 3. Realizar o processamento relativo ao processamento da rotina de interrupções (geralmente corresponde um *case switch* para cada uma das causas possíveis)
 2. Repor o contexto
 3. Retornar ao programa original (a saída é efetuada através de uma instrução própria de sret)

```
sret: PC ← SEPC
```

Tratamento de interrupções e exceções

Estrutura da rotina de tratamento de interrupções/exceções

- Os registos especiais usados no tratamento de interrupções não são acessíveis de forma normal.
- Estão mapeados numa tabela de registos de Control and Status Registers (CSRs) – ver *RISC-V Reference Guide*

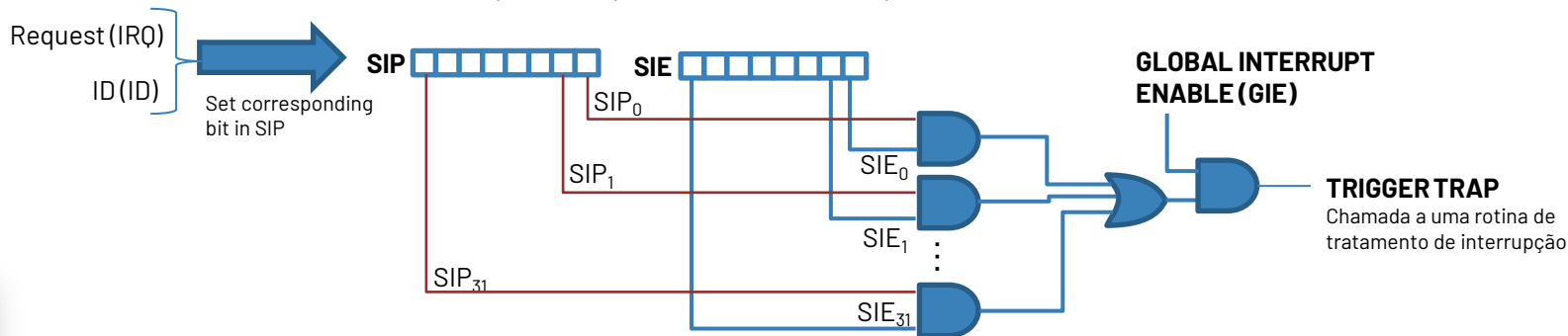
Exception Handling Registers:

Register	Description	CSR ID
sepc	Supervisor Exception PC	0x141
scause	Supervisor Exception Cause	0x142
stvec	Supervisor Trap Vector Base Register	0x105

Tratamento de interrupções e exceções

Tabela de vetores de interrupção/exceção

- Existem ainda outros registos auxiliares
 - **sie** – supervisor interrupt-enable register
Permite ativar ou desativar todas as interrupções/exceções (global enable) ou apenas ativar/desativar as interrupções/exceções de um periférico específico
Em micro-controladores existe 1 bit para cada periférico (ou classe de periféricos) e 1 bit para o global enable.
No RISC-V é um pouco mais complexo, devido ao suporte para Sistemas Operativos (SOs).
 - **sip** – supervisor interrupt-pending register
Permite saber se existe uma interrupção ou exceção por atender, mesmo que o atendimento tenha sido desligado (colocando o global enable a zero, ou o enable específico de um periférico a zero).
Geralmente existe 1 bit para cada periférico (ou classe de periféricos)



Tratamento de interrupções e exceções

Tabela de vetores de interrupção/exceção

- Existem ainda outros registos auxiliares

- `sie` – supervisor interrupt-enable register

Permite ativar ou desativar todas as interrupções/exceções (global enable) ou apenas ativar/desativar as interrupções/exceções de um periférico específico

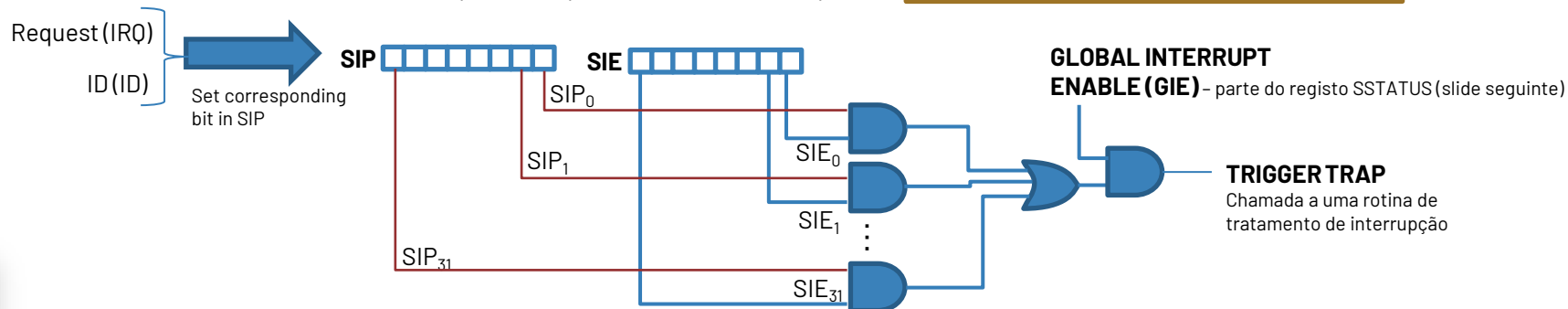
Em micro-controladores existe 1 bit para cada periférico (ou classe de periférico)
No RISC-V é um pouco mais complexo, devido ao suporte para Syscall

- `sip` – supervisor interrupt-pending register

Permite saber se existe uma interrupção ou exceção por atender, global enable a zero, ou o enable específico de um periférico a zero

Geralmente existe 1 bit para cada periférico (ou classe de periférico)

Geralmente o sistema define uma prioridade para cada interrupção na eventualidade de haver duas (ou mais) interrupções ativas simultaneamente



Tratamento de interrupções e exceções

Tabela de vetores de interrupção/exceção

- Existem ainda outros registos auxiliares
 - `sstatus` – supervisor status register

Contém vários bits com a informação do estado do processador, em particular:


- GIE (Global Interrupt Enable) – permite mascarar (valor lógico zero) todas as interrupções. Colocar a um para ativar as interrupções
- SPIE (Global Prior Interrupt Enable) – contém o valor lógico do bit GIE antes de atendermos a interrupção. Por omissão o valor de GIE é colocado a zero à entrada da rotina de tratamento de interrupções. À saída da rotina (instrução `sret`), o valor de GIE é colocado com o valor de GPIE.

Tratamento de interrupções e exceções

Tabela de vetores de interrupção/exceção

- Existem ainda outros registos auxiliares
 - `sie` - supervisor interrupt-enable register
 - `sip` - supervisor interrupt-pending register
 - `sstatus` - supervisor status register

A Causa indica o bit respetivo



	31	...	5	4	3	2	1	0
scause	<number>							
stvec	<address>							
sie	INT31	...	INT5	INT4	INT3	INT2	INT1	INT0
sip	INT31	...	INT5	INT4	INT3	INT2	INT1	INT0
sstatus	-	...	SPIE	-	-	-	GIE	-

states the number of the first pending interruption
states the address of the Interrupt Service Routine
0 - Interruptions masked; 1 - Interruptions enabled
0 - No pending interruption; 1 - Pending interruption

GIE - Global Interrupt Enable. Set to zero to disable ALL interruptions/exceptions

SIPIE - Status of GIE prior to entering the interrupt service routine. Set to zero when entering the interrupt service routine.

Cada bit corresponde a uma CAUSA diferente

Tratamento de interrupções e exceções

Estrutura da rotina de tratamento de interrupções/exceções

- Para aceder aos CSRs é preciso usar instruções especiais:

<code>csrrc</code>	<code>xd, csrid, xa</code>	<code>; xd ← CSR[csrid], CSR[csrid] ← CSR[csrid] & ~xa</code>
<code>csrrci</code>	<code>xd, csrid, imm2</code>	<code>; xd ← CSR[csrid], CSR[csrid] ← CSR[csrid] & ~imm2</code>
<code>csrrs</code>	<code>xd, csrid, xa</code>	<code>; xd ← CSR[csrid], CSR[csrid] ← CSR[csrid] xa</code>
<code>csrrsi</code>	<code>xd, csrid, imm2</code>	<code>; xd ← CSR[csrid], CSR[csrid] ← CSR[csrid] imm2</code>
<code>csrrw</code>	<code>xd, csrid, xa</code>	<code>; xd ← CSR[csrid], CSR[csrid] ← xa</code>
<code>csrrwi</code>	<code>xd, csrid, imm2</code>	<code>; xd ← CSR[csrid], CSR[csrid] ← imm2</code>

Tratamento de interrupções e exceções

Estrutura da rotina de tratamento de interrupções/exceções

- Para aceder aos CSRs é preciso usar instruções especiais:

csrrc	<code>xd,csrid,xa</code>	$; xd \leftarrow CSR[csrid], CSR[csrid] \leftarrow CSR[csrid] \& \sim xa$
csrrci	<code>xd,csrid,imm2</code>	$; xd \leftarrow CSR[csrid], CSR[csrid] \leftarrow CSR[csrid] \& \sim imm2$

```
csrrci    x10,id,0b0010
```

Duas operações:

1) $x10 \leftarrow CSR[id]$

2) $CSR[id] \leftarrow CSR[id] \& \sim (00..0\underline{1}0)$

$CSR[id] \leftarrow CSR[id] \& 11..1\underline{0}1$

Na operação *bitwise and*: 1 é um elemento neutro, 0 faz clear/reset.

Assim, vamos fazer **clear** dos bits indicados a 1 na palavra original

Tratamento de interrupções e exceções

Estrutura da rotina de tratamento de interrupções/exceções

- Para aceder aos CSRs é preciso usar instruções especiais:

csrrs	<code>xd,csrid,xa</code>	$; \text{xd} \leftarrow \text{CSR}[\text{csrid}], \text{CSR}[\text{csrid}] \leftarrow \text{CSR}[\text{csrid}] \mid \text{xa}$
csrrsi	<code>xd,csrid,imm2</code>	$; \text{xd} \leftarrow \text{CSR}[\text{csrid}], \text{CSR}[\text{csrid}] \leftarrow \text{CSR}[\text{csrid}] \mid \text{imm2}$

```
csrrsi    x10,id,0b0010
```

Duas operações:

1) $\text{x10} \leftarrow \text{CSR}[\text{id}]$

2) $\text{CSR}[\text{id}] \leftarrow \text{CSR}[\text{id}] \mid 00\dots0\underline{1}0$

Na operação *bitwise or*: 0 é um elemento neutro, 1 faz set.

Assim, vamos fazer **set** dos bits indicados a 1

Exemplo

Tratamento de interrupções

Considere que pretende implementar um velocímetro para bicicleta baseado num processador RISC-V, ao qual é acoplado um periférico (SCAUSE=5) que gera uma interrupção sempre que o aro da roda interrompe o sensor ótico.



A. Escreva o código da rotina de tratamento de interrupções que incrementa a variável `N_VOLTAS` sempre que ocorre uma interrupção

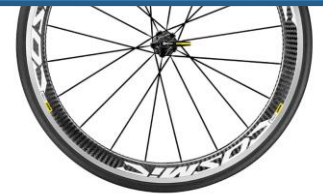
Exemplo

Tratamento de interrupções

```
N_VOLTAS:      .word    0

# rotina de tratamento de interrupções
INT:           addi     sp,sp,-8          # salvaguarda do contexto
               sw       x10,4(sp)
               sw       x11,0(sp)
               # verificar qual a interrupção
               csrrw     x10,scause,x0
               addi      x11,x0,5
               bne       x10,x11,other_int
               # se for a interrupção 5...
               la        x10,N_VOLTAS    # incrementa
               lw        x11,0(x10)       # o número
               addi      x11,x11,1        # de voltas
               sw        x11,0(x10)
               # repõe o contexto e sai
               lw        x11,0(sp)
               lw        x10,4(sp)
               addi      sp,sp,8
               sret                          # saída da interrupção
```

Verifica se é a interrupção 5. Se não for, segue para other_int



other_int: ...

Exemplo

Tratamento de
interrupções

Considere que pretende implementar um velocímetro para bicicleta baseado num processador RISC-V, ao qual é acoplado um periférico (SCAUSE=5) que gera uma interrupção sempre que o aro da roda interrompe o sensor ótico.

B. Escreva o código que configura a rotina de tratamento de interrupções



Exemplo

Tratamento de interrupções

```
MAIN:      li      sp,0x80000000
           # define a rotina que irá atender as interrupções
           la      x10,INT
           csrrw   x0,stvec,x10

           # ativa as interrupções
           li      x10,0b0100000
           csrrs   x0,sie,x10
           csrrsi  x0,sstatus,0b10
           ...
loop:      ...
           j       loop

INT:      ...
```

Assumindo que o bit x do registo CSR[sie] guarda o local enable, como queremos ativar a interrupção 5, ativamos o bit 5

O bit 1 guarda o GIE

Exemplo

Vamos assumir que o programa só realiza a contagem do número de ciclos.

Sempre o número de voltas é um múltiplo de 10, salta para `atualiza_display`.

```
MAIN:      li      sp,0x80000000
           # define a rotina que irá atender as interrupções
           la      x10,INT
           csrrw   x0,stvec,x10

           # ativa as interrupções
           li      x10,0b0100000
           csrrw   x0,sie,x10
           csrrsi  x0,sstatus,0b10

           la      x10,N_VOLTAS
           li      x12,10
loop:      lw      x11,0(x10)
           rem     x11,x11,x12
           bne     x11,x0,loop

atualiza_display:
           (...)
           j       loop

INT:      (...)
```

Execução do código

X10	
X11	
X12	

X2/SP	
PC	
SEPC	
STVEC	
SSTATUS	
SIE	

```
MAIN: 00h: li      sp,0x80000000
        # define a rotina que irá atender as interrupções
```

```
08h: la      x10,INT
```

```
10h: csrrw   x0,stvec,x10
```

```
        # ativa as interrupções
```

```
14h: li      x10,0b0100000
```

```
1Ch: csrrw   x0,sie,x10
```

```
20h: csrrsi  x0,sstatus,0b10
```

```
24h: la      x10,N_VOLTAS
```

```
2Ch: li      x12,10
```

```
loop: 34h: lw      x11,0(x10)
```

```
38h: rem     x11,x11,x12
```

```
3Ch: bne     x11,x0,loop
```

```
actualiza_display:
```

```
(...)
```

```
??h: j       loop
```

```
INT:      (...)
```

Pseudo-instrução:

Necessita de 2 instruções nativas

Pseudo-instrução:

Como o imediato é pequeno bastava 1 instrução nativa, mas vamos assumir 2

Execução do código

X10	
X11	
X12	

X2/SP	0x8000 0000
PC	0x0000 0008
SEPC	
STVEC	
SSTATUS	
SIE	

Instrução executada

MA

PC

00h: li sp,0x80000000

define a rotina que irá atender as interrupções

08h: la x10,INT

10h: csrrw x0,stvec,x10

ativa as interrupções

14h: li x10,0b0100000

1Ch: csrrw x0,sie,x10

20h: csrrsi x0,sstatus,0b10

24h: la x10,N_VOLTAS

2Ch: li x12,10

loop: 34h: lw x11,0(x10)

38h: rem x11,x11,x12

3Ch: bne x11,x0,loop

actualiza_display:

(...)

??h: j loop

INT: (...)

Exec
código

Recorde:

INT é o endereço da
primeira instrução
(ponteiro para a função)

X10	INT
X11	
X12	

X2/SP	0x8000 0000
PC	0x0000 0010
SEPC	
STVEC	
SSTATUS	
SIE	

```
MAIN: 00h: li      sp,0x80000000
        # define a rotina que irá atender as interrupções
        Instrução executada → 08h: la      x10,INT
        PC → 10h: csrrw  x0,stvec,x10

        # ativa as interrupções
        14h: li      x10,0b0100000
        1Ch: csrrw  x0,sie,x10
        20h: csrrsi x0,sstatus,0b10

        24h: la      x10,N_VOLTAS
        2Ch: li      x12,10
loop:  34h: lw      x11,0(x10)
        38h: rem    x11,x11,x12
        3Ch: bne    x11,x0,loop

actualiza_display:
        (...)

        ??h: j      loop

INT:    (...)
```

Execução do código

X10	INT
X11	
X12	

X2/SP	0x8000 0000
PC	0x0000 0014
SEPC	
STVEC	INT
SSTATUS	
SIE	

```

MAIN: 00h: li      sp,0x80000000
        # define a rotina que irá atender as interrupções

Instrução executada → 08h: la      x10,INT
                       10h: csrrw   x0,stvec,x10

        # ativa as interrupções

PC → 14h: li      x10,0b0100000
      1Ch: csrrw   x0,sie,x10
      20h: csrrsi  x0,sstatus,0b10

      24h: la      x10,N_VOLTAS
      2Ch: li      x12,10

loop: 34h: lw      x11,0(x10)
      38h: rem     x11,x11,x12
      3Ch: bne     x11,x0,loop

actualiza_display:
        (...)

      ??h: j       loop

INT:    (...)

```

Execução do código

X10	0x0000 0020
X11	
X12	

X2/SP	0x8000 0000
PC	0x0000 001C
SEPC	
STVEC	INT
SSTATUS	
SIE	

```

MAIN: 00h: li      sp,0x80000000
        # define a rotina que irá atender as interrupções
008h: la      x10,INT
010h: csrrw   x0,stvec,x10

        # ativa as interrupções
14h: li      x10,0b0100000
1Ch: csrrw   x0,sie,x10
20h: csrrsi  x0,sstatus,0b10

24h: la      x10,N_VOLTAS
2Ch: li      x12,10
loop: 34h: lw      x11,0(x10)
38h: rem     x11,x11,x12
3Ch: bne     x11,x0,loop

actualiza_display:
        (...)

??h: j      loop

INT:      (...)

```

Instrução executada



PC



Execução do código

X10	0x0000 0020
X11	
X12	

X2/SP	0x8000 0000
PC	0x0000 0020
SEPC	
STVEC	INT
SSTATUS	
SIE	0x0000 0020

```

MAIN: 00h: li      sp,0x80000000
        # define a rotina que irá atender as interrupções
008h: la      x10,INT
010h: csrrw   x0,stvec,x10

        # ativa as interrupções
14h: li      x10,0b0100000
1Ch: csrrw   x0,sie,x10
PC → 20h: csrrsi x0,sstatus,0b10

24h: la      x10,N_VOLTAS
2Ch: li      x12,10
loop: 34h: lw      x11,0(x10)
38h: rem     x11,x11,x12
3Ch: bne     x11,x0,loop

actualiza_display:
        (...)
??h: j      loop

INT:      (...)
  
```

Execução do código

X10	0x0000 0020
X11	
X12	

X2/SP	0x8000 0000
PC	0x0000 0024
SEPC	
STVEC	INT
SSTATUS	0bx...x xx1x
SIE	0x0000 0020

Instrução executada

PC

Faz set ao bit 1 (GIE).

O resto permanece inalterado

```

MAIN: 00h: li      sp,0x80000000
        # define a rotina que irá atender as interrupções
008h: la      x10,INT
010h: csrrw   x0,stvec,x10

        # ativa as interrupções
014h: li      x10,0b0100000
01Ch: csrrw   x0,sie,x10
020h: csrrsi  x0,sstatus,0b10

        PC → 024h: la      x10,N_VOLTAS
02Ch: li      x12,10
loop: 034h: lw      x11,0(x10)
038h: rem     x11,x11,x12
        x11,x0,loop
        y:
loop

INT:      (...)
  
```

Exec
código

Recorde:
N_VOLTAS é o
endereço da variável
(ponteiro para a variável)

X10	N_VOLTAS
X11	
X12	

X2/SP	0x8000 0000
PC	0x0000 002C
SEPC	
STVEC	INT
SSTATUS	0bx...x xx1x
SIE	0x0000 0020

```
MAIN: 00h: li      sp,0x80000000
        # define a rotina que irá atender as interrupções
008h: la      x10,INT
010h: csrrw   x0,stvec,x10

        # ativa as interrupções
014h: li      x10,0b0100000
01Ch: csrrw   x0,sie,x10
020h: csrrsi  x0,sstatus,0b10

        Instrução executada
        PC → 024h: la      x10,N_VOLTAS
        02Ch: li      x12,10
loop:  034h: lw      x11,0(x10)
        038h: rem    x11,x11,x12
        03Ch: bne    x11,x0,loop

        atualiza_display:
                (...)
        ??h: j      loop

INT:      (...)
```

Execução do código

X10	N_VOLTAS
X11	
X12	10

X2/SP	0x8000 0000
PC	0x0000 0034
SEPC	
STVEC	INT
STATUS	0bx...x xx1x
SIE	0x0000 0020

```

MAIN: 00h: li      sp,0x80000000
        # define a rotina que irá atender as interrupções
008h: la      x10,INT
010h: csrrw   x0,stvec,x10

        # ativa as interrupções
014h: li      x10,0b0100000
01Ch: csrrw   x0,sie,x10
020h: csrrsi  x0,sstatus,0b10

024h: la      x10,N_VOLTAS
02Ch: li      x12,10
034h: lw      x11,0(x10)
038h: rem     x11,x11,x12
03Ch: bne     x11,x0,loop

atualiza_display:
        (...)

??h: j        loop

INT:    (...)

```

Execução do código

X10	N_VOLTAS
X11	
X12	10

X2/SP	0x8000 0000
PC	0x0000 0034
SEPC	
STVEC	INT
SSTATUS	0bx...x xx1x
SIE	0x0000 0020

```

MAIN: 00h: li      sp,0x80000000
        # define a rotina que ...
008h: la      x10,INT
010h: csrrw   x0,stvec,x10
        # ativa as interrupções
014h: li      x10,0b0100000
01Ch: csrrw   x0,sie,x10
020h: csrrsi  x0,sstatus,0b10

024h: la      x10,N_VOLTAS
02Ch: li      x12,10
034h: lw      x11,0(x10)
038h: rem     x11,x11,x12
03Ch: bne     x11,x0,loop

actualiza_display:
        (...)

??h: j       loop

INT:      (...)

```

Instrução em
execução



Interrupção 5

Vamos assumir que durante a execução da instrução no endereço 34h, ocorre uma interrupção externa do periférico 5 (contador de voltas)

Execução do código

Vamos assumir que o valor atual é 0

X10	
X11	0
X12	10

X2/SP	0x8000 0000
PC	0x0000 0038
SEPC	
STVEC	INT
SSTATUS	0bx...x xx1x
SIE	0x0000 0020

Instrução em execução

PC

```
MAIN: 00h: li      sp,0x80000000
        # define a rotina que ...
08h: la      x10,INT
10h: csrrw   x0,stvec,x10

        # ativa as interrupções
14h: li      x10,0b0100000
1Ch: csrrw   x0,sie,x10
20h: csrrsi  x0,sstatus,0b10

24h: la      x10,N_VOLTAS
2Ch: li      x12,10
34h: lw      x11,0(x10)
38h: rem     x11,x11,x12
3Ch: bne     x11,x0,loop

actualiza_display:
        (...)

??h: j       loop

INT:      (...)
```

Vamos assumir que durante a execução da instrução no endereço 34h, ocorre uma interrupção externa do periférico 5 (contador de voltas)

Passo 1: terminar a execução da instrução

Interrupção 5

Execução do código

X10	N_VOLTAS
X11	0
X12	10

X2/SP	0x8000 0000
PC	INT
SEPC	0x0000 0038
STVEC	INT
SSTATUS	0bx...xx1x xx0x
SIE	0x0000 0020

```

MAIN: 00h: li      sp,0x80000000
        # define a rotina que trata as interrupções
08h: la      x10,INT
10h: csrrw   x0,stvec,x10

        # ativa as interrupções
14h: li      x10,0b0100000
1Ch: csrrw   x0,sie,x10
20h: csrrsi  x0,sstatus,0b10

24h: la      x10,N_VOLTAS

```

Vamos assumir que durante a execução da instrução no endereço 34h, ocorre uma interrupção externa do periférico 5 (contador de voltas)

Passo 1: terminar a execução da instrução

Passo 2: chamar a rotina de tratamento de interrupções

Endereço da próxima instrução: a primeira da rotina de tratamento de interrupções

Endereço de retorno da rotina de tratamento de interrupções

A. Salva o valor de GIE em SPIE (SPIE ← GIE)
 B. Mascara todas as interrupções (GIE ← 0)
 (SPIE == SSTATUS[5], GIE == SSTATUS[1])

 **Interrupção 5**

Exemplo

X10	N_VOLTAS
X11	0
X12	10

X2/SP	0x7FFF FFF8
PC	INT+4
SEPC	0x0000 0038
STVEC	INT
SSTATUS	0bx...xx1x xx0x
SIE	0x0000 0020

```
N_VOLTAS:    .word    0
```

Instrução executada
rotina de tratamento de interrupções

```
INT:          addi    sp,sp,-8          # salvaguarda do contexto
```

PC → `sw x10,4(sp)`

`sw X11,0(sp)`

verificar qual a interrupção

```
csrrw x10,scause,x0
```

```
addi x11,x10,-5
```

```
bne x11,x0,other_int
```

se for a interrupção 5...

```
la x10,N_VOLTAS
```

```
lw x11,0(x10)          # do número
```

```
addi x11,x11, 1        # de voltas
```

```
sw x11,0(x10)
```

repõe o contexto e sai

```
lw x11,0(sp)
```

```
lw x10,4(sp)
```

```
addi sp,sp,8
```

```
sret          # saída da interrupção
```

```
other_int: ...
```



Exemplo

```
N_VOLTAS:    .word    0
```

rotina de tratamento de interrupções

```
INT:         addi    sp,sp,-8          # salvaguarda do contexto
```

```
            sw      x10,4(sp)
```

```
            sw      X11,0(sp)
```

verificar qual a interrupção

```
            csrrw   x10,scause,x0
```

```
            addi    x11,x10,-5
```

```
            bne     x11,x0,other_int
```

se for a interrupção 5...

```
            la      x10,N_VOLTAS
```

```
            lw      x11,0(x10)        # do número
```

```
            addi    x11,x11, 1        # de voltas
```

```
            sw      x11,0(x10)
```

repõe o contexto e sai

```
            lw      x11,0(sp)
```

```
            lw      x10,4(sp)
```

```
            addi    sp,sp, 8
```

```
            sret
```

saída da interrupção

```
other_int: ...
```



A causa da interrupção é o periférico 5, logo segue em frente

Incrementa N_VOLTAS

Repõe o contexto

Instrução
executada
PC



X10	
X11	
X12	
X2/SP	0x8000 0000
PC	
SEPC	0x0000 0038
STVEC	INT
SSTATUS	0bx...xx1x xx0x
SIE	0x0000 0020

A ocorrência das instruções pode modificar os valores com endereço menor que SP

Dai as seguintes regras:

1. O PUSH de valores para a pilha deve começar sempre pela reserva
2. O POP de valores apenas deve modificar o SP depois dos valores terem sido lidos
3. Todas as palavras em endereços menores que SP são consideradas lixo

```
N_VOLTAS: .word 0
```

rotina de tratamento de interrupções

```
INT: addi    sp,sp,-8          # salvaguarda do contexto
      sw     x10,4(sp)
      sw     x11,0(sp)
      # verificar qual a interrupção
      csrrw  x10,scause,x0
      addi   x11,x10,-5
      bne    x11,x0,other_int
      # se for a interrupção 5...
      la     x10,N_VOLTAS
      lw     x11,0(x10)        # do número
      addi   x11,x11, 1        # de voltas
      sw     x11,0(x10)
      # repõe o contexto e sai
      lw     x11,0(sp)
      lw     x10,4(sp)
      addi   sp,sp,8
      sret                                     # saída da interrupção
```



Instrução
executada
PC

```
other_int: ...
```

X2/SP	0x8000 0000
PC	?
SEPC	0x0000 0038
STVEC	INT
SSTATUS	0bx...xx1x xx0x
SIE	0x0000 0020

Exemplo

X10	N_VOLTAS
X11	0
X12	10

X2/SP	0x8000 0000
PC	0x0000 0038
SEPC	0x0000 0038
STVEC	INT
SSTATUS	0bx...xx1x xx1x
SIE	0x0000 0020

PC ← SEPC

GIE ← SPIE

Instrução em
execução
other_int: ...

```

N_VOLTAS:    .word    0

# rotina de tratamento de interrupções
INT:         addi     sp,sp,-8          # salvaguarda do contexto
              sw      x10,4(sp)
              sw      X11,0(sp)
              # verificar qual a interrupção
              csrrw   x10,scause,x0
              addi    x11,x10,-5
              bne     x11,x0,other_int
              # se for a interrupção 5...
              la      x10,N_VOLTAS
              lw      x11,0(x10)        # do número
              addi    x11,x11, 1        # de voltas
              sw      x11,0(x10)
              # repõe o contexto e sai
              lw      x11,0(sp)
              lw      x10,4(sp)
              addi    sp,sp,8
              sret
              # saída da interrupção
  
```



Execução código

O valor lido foi 0!
(contudo, devido à ocorrência
da interrupção, uma nova
leitura resultaria no valor 1)

X10	
X11	0
X12	10

X2/SP	0x8000 0000
PC	0x0000 0038
SEPC	
STVEC	INT
SSTATUS	0bx...x xx1x
SIE	0x0000 0020

```

MAIN: 00h: li      sp,0x80000000
        # define a rotina que irá atender as interrupções
008h: la      x10,INT
010h: csrrw   x0,stvec,x10

        # ativa as interrupções
014h: li      x10,0b0100000
01Ch: csrrw   x0,sie,x10
020h: csrrsi  x0,sstatus,0b10

024h: la      x10,N_VOLTAS
02Ch: li      x12,10
loop:  34h: lw      x11,0(x10)
PC → 38h: rem      x11,x11,x12
      3Ch: bne     x11,x0,loop

actualiza_display:
        (...)

??h: j      loop

INT:    (...)

```

O programa principal resume a execução sem ter qualquer percepção da existência de uma interrupção (com exceção, é claro, de uma possível alteração do valor da variável N_VOLTAS)

Execução do código

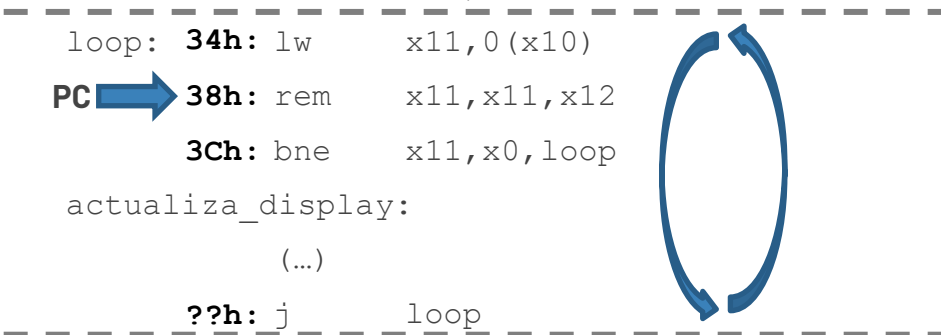
O programa fica num loop à espera da ocorrência de uma interrupção.

Se o processador for muito rápido (tipicamente é), o loop ocorre MUITAS vezes antes da variável alterar de estado.

```
MAIN: 00h: li      sp,0x80000000
        # define a rotina que irá atender as interrupções
08h: la      x10,INT
10h: csrrw    x0,stvec,x10

        # ativa as interrupções
14h: li      x10,0b0100000
1Ch: csrrw    x0,sie,x10
20h: csrrsi   x0,sstatus,0b10

24h: la      x10,N_VOLTAS
2Ch: li      x12,10
-----
loop: 34h: lw      x11,0(x10)
PC → 38h: rem     x11,x11,x12
      3Ch: bne     x11,x0,loop
      actualiza_display:
      (...)
      ??h: j       loop
-----
INT:      (...)
```



Wait for interrupt

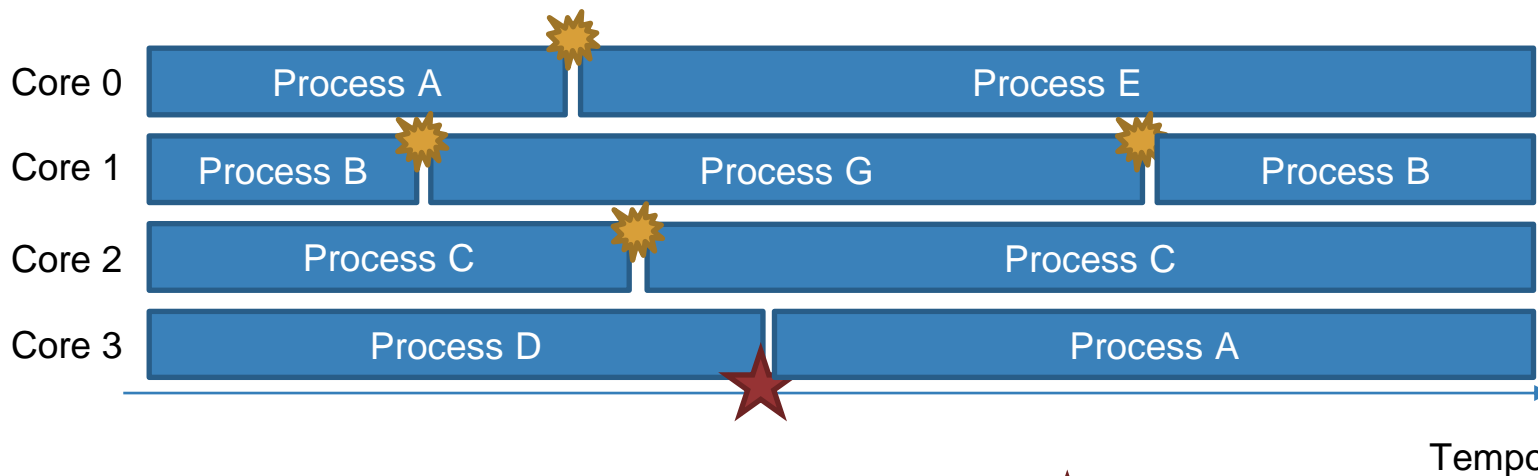
Tratamento de interrupções

- Geralmente os micro-controladores têm uma instrução de *sleep* até à ocorrência de uma interrupção.
- O sleep pode ser associado a:
 - um evento do temporizador – permite que o processador fique em modo *ultra low-power* durante um tempo pré-determinado (ex: sleep for 1s)
 - Um evento de uma interrupção externa – permite que o processador fique em espera até à ocorrência de uma interrupção.
- No caso do RISC-V existe uma instrução de WFI (wait for interrupt).
- Em aplicações com Sistema Operativo, o sleep é implementado tirando o processo de execução

Wait for interrupt

Tratamento de interrupções

- Em aplicações com Sistema Operativo, o “sleep” é implementado tirando o processo de execução



Preempção: o tempo que o SO deu ao processo para executar chegou ao final.

- O processo sai de execução e vai para uma fila de espera.
- O SO retira um processo da fila de espera (pode ser o mesmo) e coloca-o em execução.



Wait for Interrupt: O processo informa o SO que está à espera de um evento externo. O SO coloca o processo a “dormir”. Quando a interrupção chegar, o processo acorda e vai para a fila de espera. Eventualmente, o mecanismo de preempção irá colocar o processo novamente em execução.

Execução do código

Espera que uma interrupção ocorra. Mas não há nenhuma garantia que seja associada ao periférico 5

```
MAIN:      li      sp,0x80000000
           # define a rotina que irá atender as interrupções
           la      x10,INT
           csrrw   x0,stvec,x10

           # ativa as interrupções
           li      x10,0b0100000
           csrrw   x0,sie,x10
           csrrsi  x0,sstatus,0b10

           la      x10,N_VOLTAS
           li      x12,10
loop:      lw      x11,0(x10)
           rem     x11,x11,x12
           bne     x11,x0,loop

actualiza_display:
           (...)
           j      loop

INT:      (...)
```

Tabela de vetores de interrupção/exceção

Tratamento de interrupções e exceções

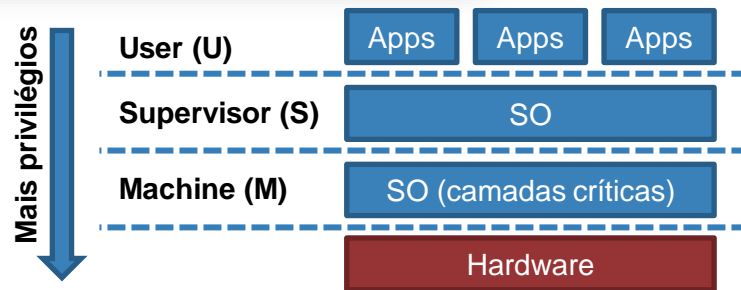
- O RISC-V permite também o uso de uma tabela de vetores de interrupções.
- Nesse caso, o registo **stvec** guarda a base para a tabela de vectores de interrupções
- Cada entrada na tabela deve conter um jump para a rotina de tratamento de interrupções

VANTAGEM:

- Existe uma rotina de tratamento de interrupções para cada tipo de interrupção/periférico diferente

		FFFF FFFF FFFF FFFFh
	...	
Espaço para 4B, i.e., 1 instrução	INT 3: jal x0,int3	stvec+3x4
Espaço para 4B, i.e., 1 instrução	INT 2: jal x0,int2	stvec+2x4
Espaço para 4B, i.e., 1 instrução	INT 1: jal x0,int1	stvec+1x4
Espaço para 4B, i.e., 1 instrução	INT 0: jal x0,int0	stvec+0x4
		0000 0000 0000 0000h

- O RISC-V suporta diferentes níveis de privilégio
 - 0 - User (U): aplicações
 - 1 - Supervisor (S): Camadas menos críticas do SO
 - 2 - Hypervisor (em definição)
 - 3 - Machine (M): Acesso total ao processador



- Diferentes modos de privilégio têm acesso a instruções diferentes:
 - A generalidade das instruções definidas em ACOM são de nível U
 - O acesso aos registos especiais (CSRs) em geral só é permitido em níveis mais baixos (mais privilégios)
 - Em alguns casos, a configuração de registos especiais tem de ser realizada chamando uma função de sistema operativo (realizado através da geração de uma exceção).
- A acesso aos periféricos geralmente está restrito aos níveis de privilégio superiores
- Os mecanismos de gestão e tratamento de interrupções são definidos por camadas
 - Interrupções ao nível de *user* (registos `uepc`, `ucause`, `utvec`, ...)
 - Interrupções ao nível de *supervisor* (registos `sepc`, `scause`, `stvec`, ...)
 - Interrupções ao nível de *machine* (registos `mepc`, `mcause`, `mtvec`, ...)

Invocação manual de uma interrupção/exceção

- As instruções `ecall` e `ebreak` permitem a transferência do controlo de execução para o sistema operativo (SO).
- O processo é efetuado através da geração de uma exceção:
 - No caso da instrução `ecall`, a identificação da exceção (identificado no registo `SCAUSE`) é passado pelo registo `x17=a7`
 - No caso da instrução `ebreak`, a causa será sempre a introdução de um ponto de paragem no programa (usado pelos *debuggers*)

Exemplo:

saída do programa

```
li      a7,10
```

```
ecall
```

Chama o debugger (e.g., gdb)

```
ebreak
```

Invocação manual de uma interrupção/exceção

- As instruções `ecall` e `ebreak` permitem a transferência do controlo de execução para o sistema operativo (SO).
- O processo é efetuado através da geração de uma exceção:
 - No caso da instrução `ecall`, a identificação da exceção (identificado no registo `SCAUSE`) é passado pelo registo `x17=a7`
 - No caso da instrução `ebreak`, a causa será sempre a introdução de um ponto de paragem no programa

O equivalente nos vossos computadores é a instrução:
`INT 3`
(call exception #3: breakpoint)

Exemplo:

```
# saída do programa  
li      a7,10  
ecall
```

```
# Chama o debugger (e.g., gdb)  
ebreak
```