MICROPROCESSADORES II (EMA864315)

ENTRADA E SAÍDA (I/O)

1º SEMESTRE / 2019

Alexandro Baldassin

MATERIAL DIDÁTICO

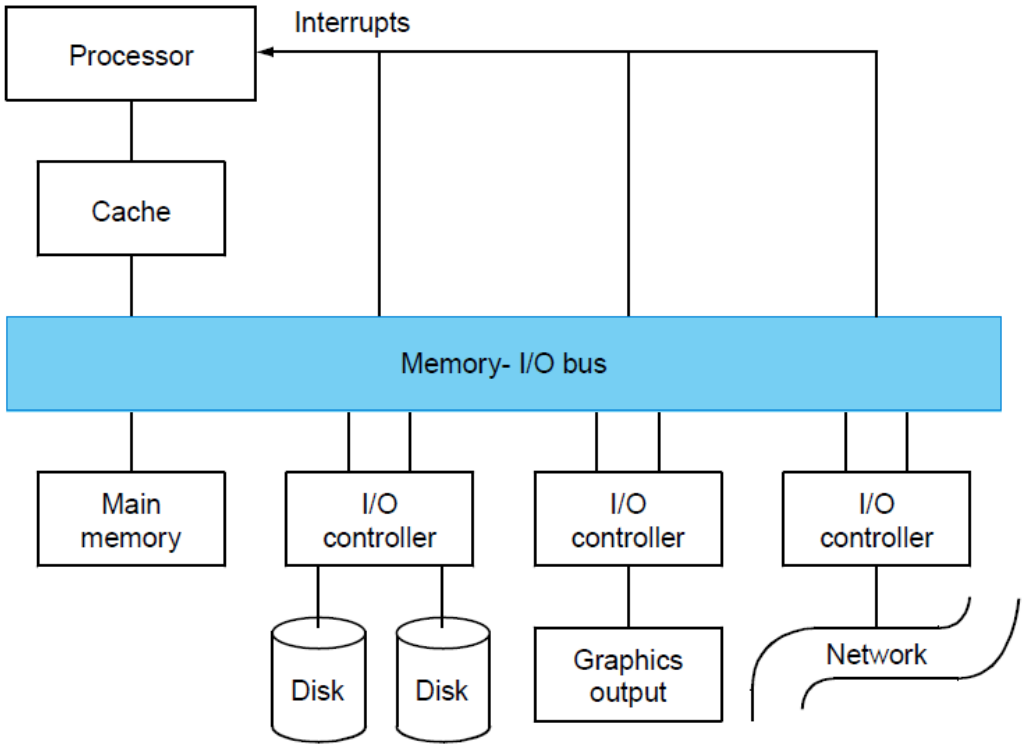
- **Patterson & Hennessy (4a edição)**
  - Capítulo 6 (Tópicos I/O)
    - 6.1 – Introduction
    - 6.5 – Connecting Processors, Memory, and I/O Devices
    - 6.6 – Interfacing I/O Devices to the Processor, Memory, and Operating System

I/O: REQUISITOS

- **O que precisamos para fazer um sistema de I/O funcionar?**
  - Uma forma de **conectar** os dispositivos com o processador e memória
  - Uma forma de **controlar** estes dispositivos (responder e transferir dados)
  - Uma forma de apresentá-los aos programas de usuários, para que possam ser úteis
    - Assunto da disciplina de Sistema Operacional

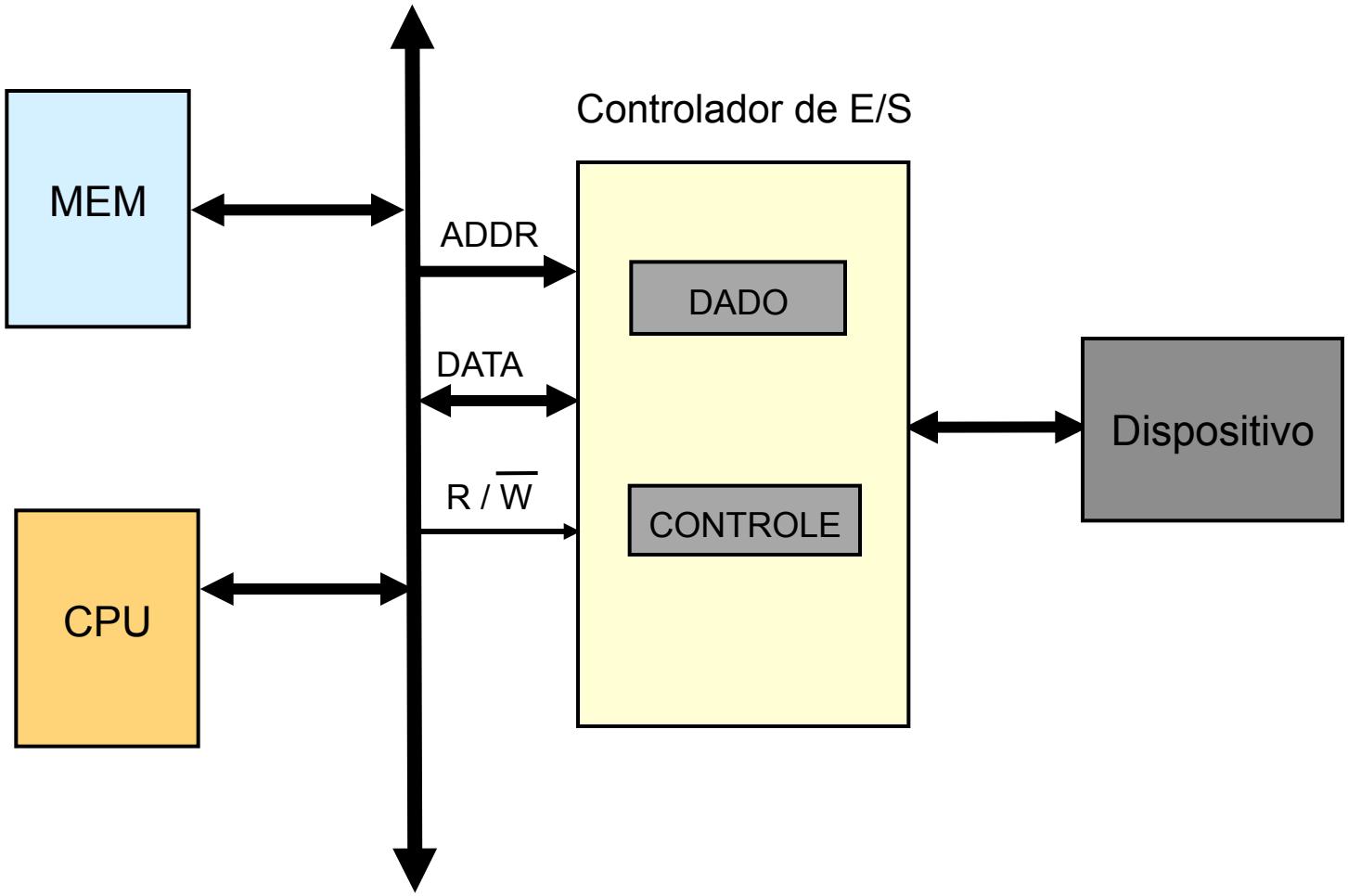
I/O: CONECTIVIDADE

- **Barramento**
  - conjunto de linhas (fios) que interligam os dispositivos do sistema computactional (cpu, memória, periféricos)



Barramentos:
- de dados
- de endereço
- de controle

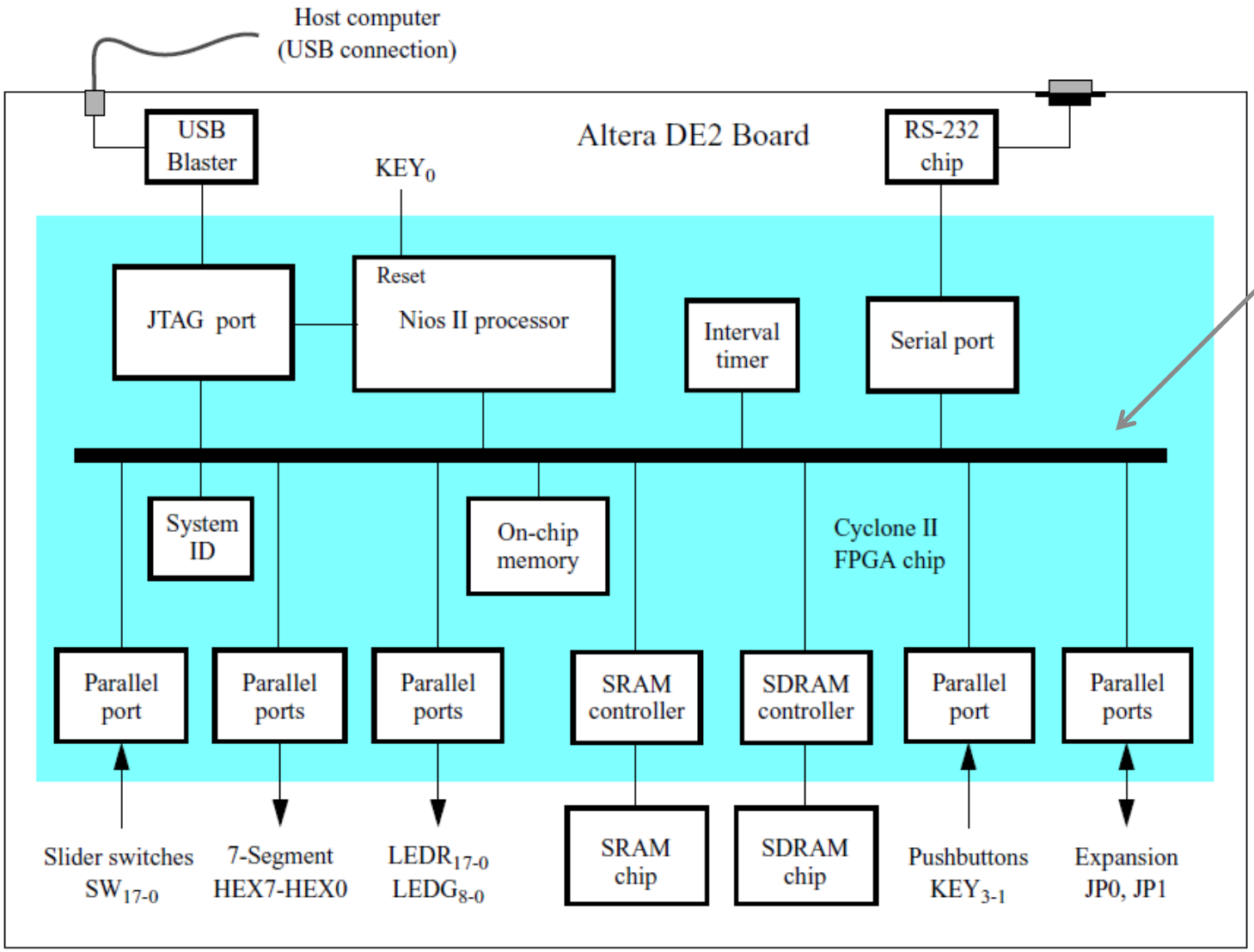
CONTROLADOR DE I/O TÍPICO



TIPOS DE INTERFACE

- **Controlador (porta) serial**
  - Um único sinal, um bit após o outro
  - Padrões: RS-232, RS-485
- **Controlador (porta) paralela**
  - Um dado inteiro por vez (8 bits, etc...)
  - Padrões: Centronics, etc...

DE2 BASIC COMPUTER



barramento Avalon

I/O: CONTROLE

- **Qual o papel do processador nas operações de I/O?**

I/O: CONTROLE

- **Qual o papel do processador nas operações de I/O?**
  - Entrada: Ler uma sequência de bytes
  - Saída: Escrever uma sequência de bytes

- **Para que o processador possa controlar um dispositivo de I/O, ele deve:**

I/O: CONTROLE

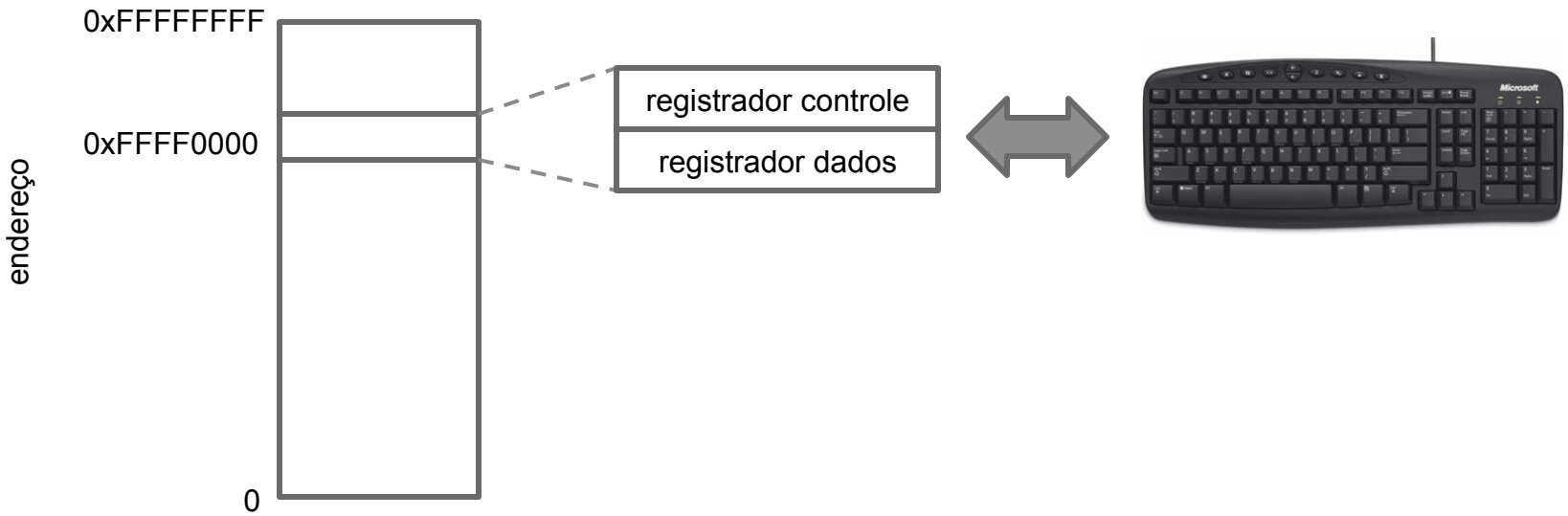
- **Qual o papel do processador nas operações de I/O?**
  - Entrada: Ler uma sequência de bytes
  - Saída: Escrever uma sequência de bytes
- **Para que o processador possa controlar um dispositivo de I/O, ele deve:**
  - Saber como endereçar tal dispositivo
  - Fornecer comandos de controle
- **Dois métodos principais**
  - Instruções específicas (ex: `in` e `out` no IA32)
  - I/O mapeada em memória

I/O MAPEADA EM MEMÓRIA

- **Parte do espaço de endereçamento é dedicado à comunicação com dispositivos de entrada e saída**
  - Instruções de **load** e **store** fazem a comunicação com dispositivos
  - Ao invés de acessar memória física, as instruções acessam registradores dos dispositivos

I/O MAPEADA EM MEMÓRIA

- **Parte do espaço de endereçamento é dedicado à comunicação com dispositivos de entrada e saída**
  - Instruções de **load** e **store** fazem a comunicação com dispositivos
  - Ao invés de acessar memória física, as instruções acessam registradores dos dispositivos



MAPA DA MEMÓRIA DO DE2 BASIC COMPUTER

Base Address	End Address	I/O Peripheral
0x00000000	0x007FFFFF	SDRAM
0x08000000	0x0807FFFF	SRAM
0x09000000	0x09001FFF	On-chip Memory
0x10000000	0x1000000F	Red LED parallel port
0x10000010	0x1000001F	Green LED parallel port
0x10000020	0x1000002F	7-segment HEX3–HEX0 displays parallel port
0x10000030	0x1000003F	7-segment HEX7–HEX4 displays parallel port
0x10000040	0x1000004F	Slider switch parallel port
0x10000050	0x1000005F	Pushbutton parallel port
0x10000060	0x1000006F	JP1 Expansion parallel port
0x10000070	0x1000007F	JP2 Expansion parallel port
0x10001000	0x10001007	JTAG UART port
0x10001010	0x10001017	Serial port
0x10002000	0x1000201F	Interval timer
0x10002020	0x10002027	System ID

Olhar “Basic Computer System for the Altera DE2 Board.pdf”

MAPA DA MEMÓRIA DO DE2 BASIC COMPUTER

Base Address	End Address	I/O Peripheral
0x00000000	0x007FFFFF	SDRAM
0x08000000	0x0807FFFF	SRAM
0x09000000	0x09001FFF	On-chip Memory
0x10000000	0x1000000F	Red LED parallel port
0x10000010	0x1000001F	Green LED parallel port
0x10000020	0x1000002F	7-segment HEX3–HEX0 displays parallel port
0x10000030	0x1000003F	7-segment HEX7–HEX4 displays parallel port
0x10000040	0x1000004F	Slider switch parallel port
0x10000050	0x1000005F	Pushbutton parallel port
0x10000060	0x1000006F	JP1 Expansion parallel port
0x10000070	0x1000007F	JP2 Expansion parallel port
0x10001000	0x10001007	JTAG UART port
0x10001010	0x10001017	Serial port
0x10002000	0x1000201F	Interval timer
0x10002020	0x10002027	System ID

I/O mapeada em memória

Olhar “Basic Computer System for the Altera DE2 Board.pdf”

COMUNICAÇÃO ENTRE CPU E DISPOSITIVOS

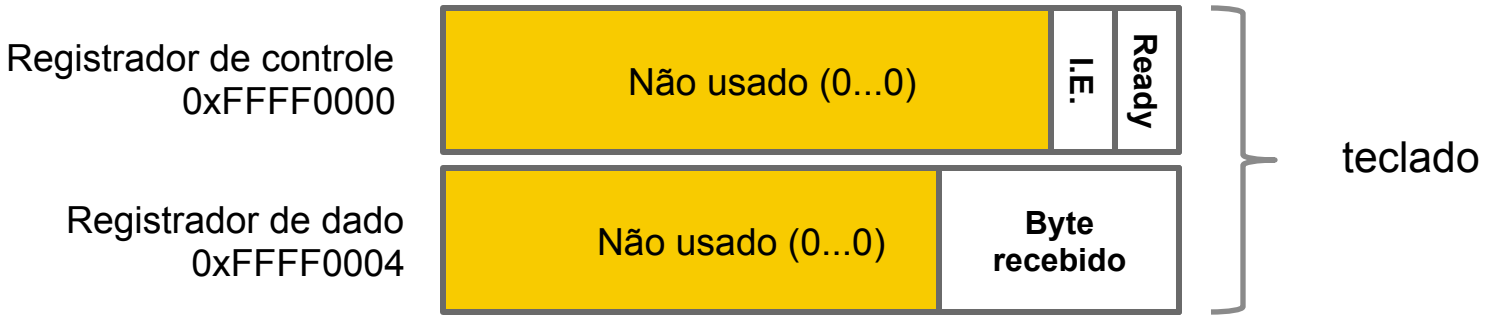
- **Considere um processador de 1GHz...**
  - 1 bilhão de instruções de *load* e *store* por segundo (4GB/s)
  - Dispositivos de I/O podem ter taxa de transferência variando entre 0.01KB/s e 1GB/s
- **Problema**
  - Um dispositivo pode não estar pronto para enviar ou receber dados na velocidade do processador
- **O que fazer em relação a essa disparidade de velocidade?**

COMUNICAÇÃO COM I/O: MÉTODO 1

- **Dispositivo geralmente possui dois registradores**
  - Controle: informa quando está OK para ler ou escrever
  - Dado: contém o dado (a ser lido ou escrito)
- **Processador lê o registrador de controle em um laço, até que o dispositivo esteja pronto**
- **Após sinal de pronto, processador lê ou escreve informação usando o registrador de dado**
  - Essa operação reseta o estado do dispositivo (não pronto)

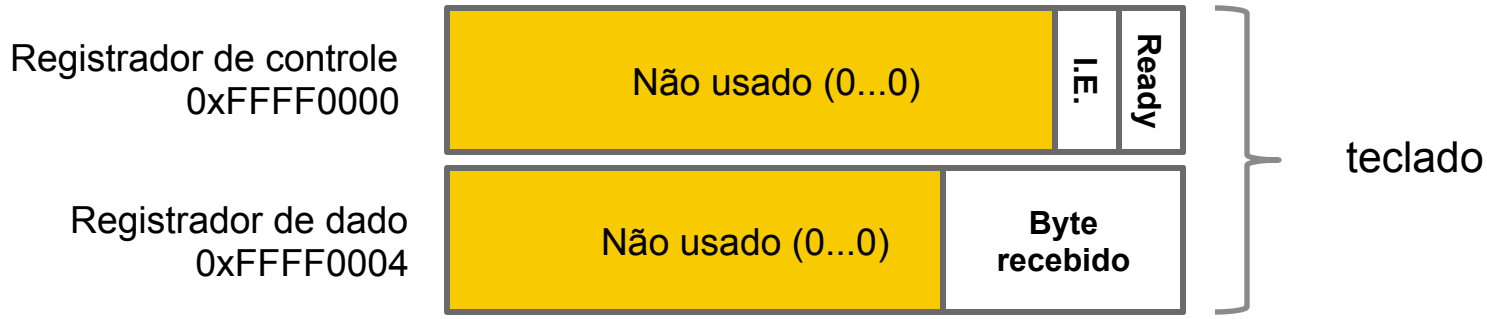
EXEMPLO USANDO MÉTODO 1

- **Considere um terminal (teclado + display) usando I/O mapeada em memória**
  - Leitura do teclado (entrada) – 2 registradores

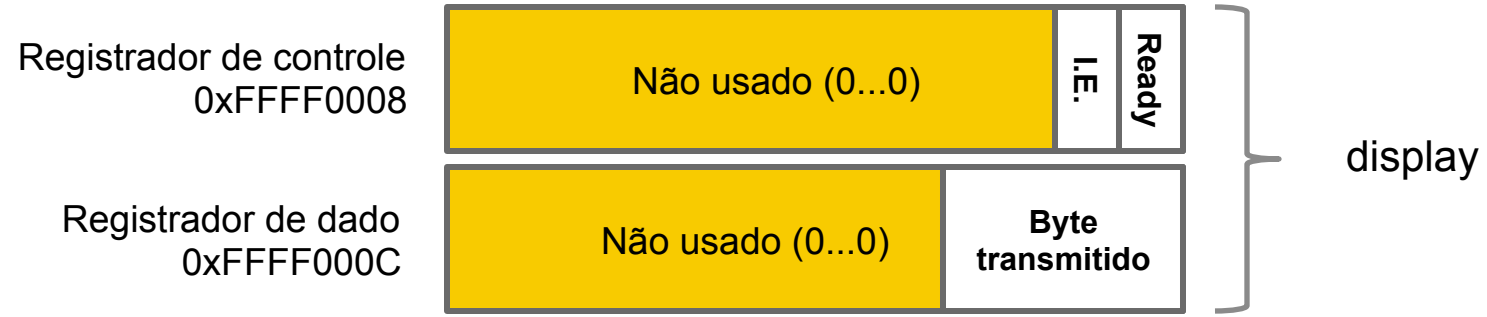


EXEMPLO USANDO MÉTODO 1

- **Considere um terminal (teclado + display) usando I/O mapeada em memória**
  - Leitura do teclado (entrada) – 2 registradores



  - Escrita na tela (saída) – 2 registradores



EXEMPLO USANDO MÉTODO 1

- **Registrador de controle (bit menos significativo)**
  - Teclado: Ready == 1 significa que o caractere no registrador de dado ainda não foi lido
    - quando for lido, Ready vai de 1 => 0
  - Display: Ready == 1 significa que transmissor está apto a aceitar novo caractere
    - bit I.E. discutido mais tarde

- **Registrador de dado (byte menos significativo)**
  - Teclado: último caractere digitado
  - Display: caractere a ser impresso na tela

EXEMPLO USANDO MÉTODO 1

- **Entrada: Lê do teclado para registrador *r4***

EXEMPLO USANDO MÉTODO 1

- **Entrada: Lê do teclado para registrador *r4***

```
            orhi    r2, r0, 0xFFFF      # 0xFFFF0000
WaitLoop:
            ldwio   r3, 0(r2)           # controle
            andi    r3, r3, 0x1
            beq     r3, r0, WaitLoop
            ldwio   r4, 4(r2)           # dado
```

EXEMPLO USANDO MÉTODO 1

- **Entrada: Lê do teclado para registrador *r4***

```
            orhi    r2, r0, 0xFFFF      # 0xFFFF0000
WaitLoop:
            ldwio   r3, 0(r2)           # controle
            andi    r3, r3, 0x1
            beq     r3, r0, WaitLoop
            ldwio   r4, 4(r2)           # dado
```

Note que para realizar leituras (e escritas) de dispositivos em memória mapeada é necessário utilizar instruções especiais (por quê?)

EXEMPLO USANDO MÉTODO 1

- **Saída: Escreve valor em *r2* para a tela**

```
            orhi    r4, r0, 0xFFFF      # 0xFFFF0000
WaitLoop:
            ldwio   r3, 8(r4)           # controle
            andi    r3, r3, 0x1
            beq     r3, r0, WaitLoop
            stwio   r2, 12(r4)          # dado
```

MÉTODO 1 - POLLING

- **Esse método é conhecido como *polling*, ou I/O programada**
  - Processador espera pelo dispositivo em um laço

- **Qual a desvantagem do polling?**

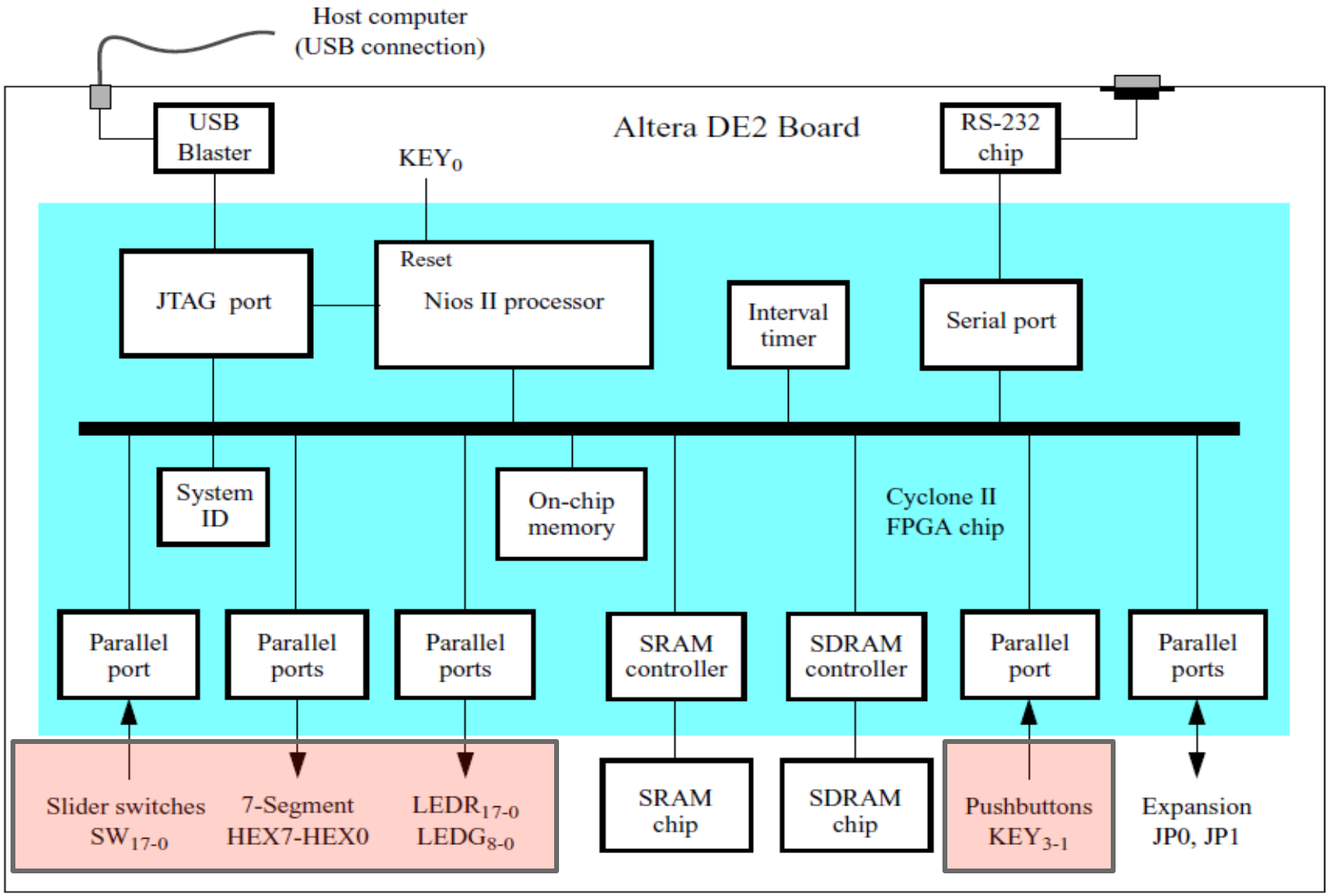
MÉTODO 1 - POLLING

- **Esse método é conhecido como *polling*, ou I/O programada**
  - Processador espera pelo dispositivo em um laço
- **Qual a desvantagem do polling?**
  - Processador gasta grande parte de seu tempo sem fazer nada útil (tempo de resposta a outras aplicações é alta)
- **Gostaríamos que o processador não esperasse. Ideias?**

MÉTODO 1 - POLLING

- **Esse método é conhecido como *polling*, ou I/O programada**
  - Processador espera pelo dispositivo em um laço
- **Qual a desvantagem do polling?**
  - Processador gasta grande parte de seu tempo sem fazer nada útil (tempo de resposta a outras aplicações é alta)
- **Gostaríamos que o processador não esperasse. Ideias?**
  - Fazer com que o dispositivo avise: **mecanismo de interrupção**

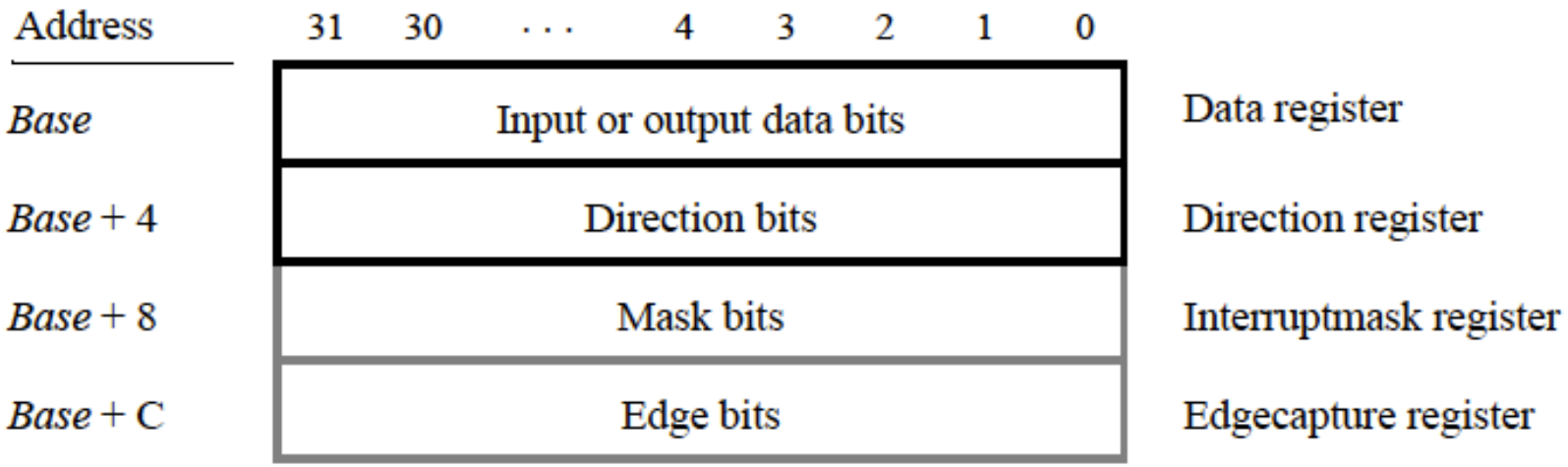
LABORATÓRIO – I/O



LABORATÓRIO – I/O

- **Neste laboratório veremos os dispositivos I/O que usam portas de comunicação paralela**
  - Seção 2.3 da referência *Basic Computer System for the Altera DE2 Board*

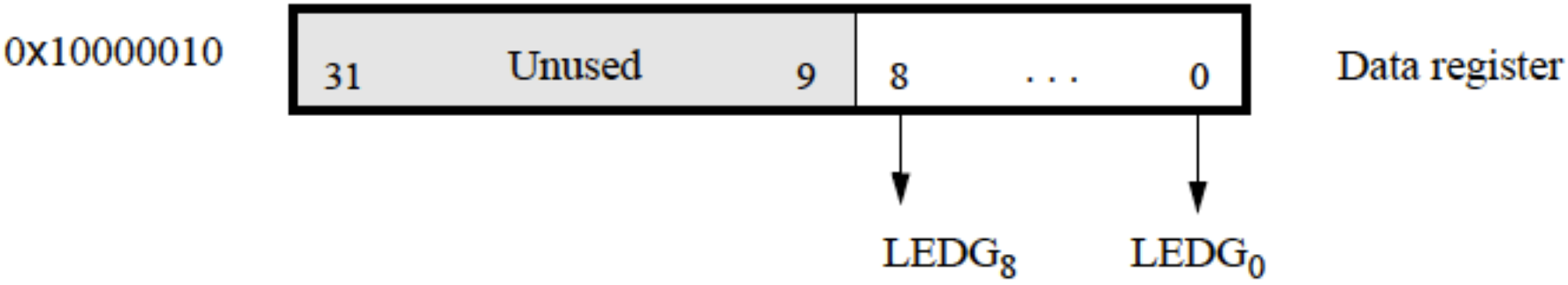
- **A interface PIO (Parallel I/O) pode conter até 4 registradores**



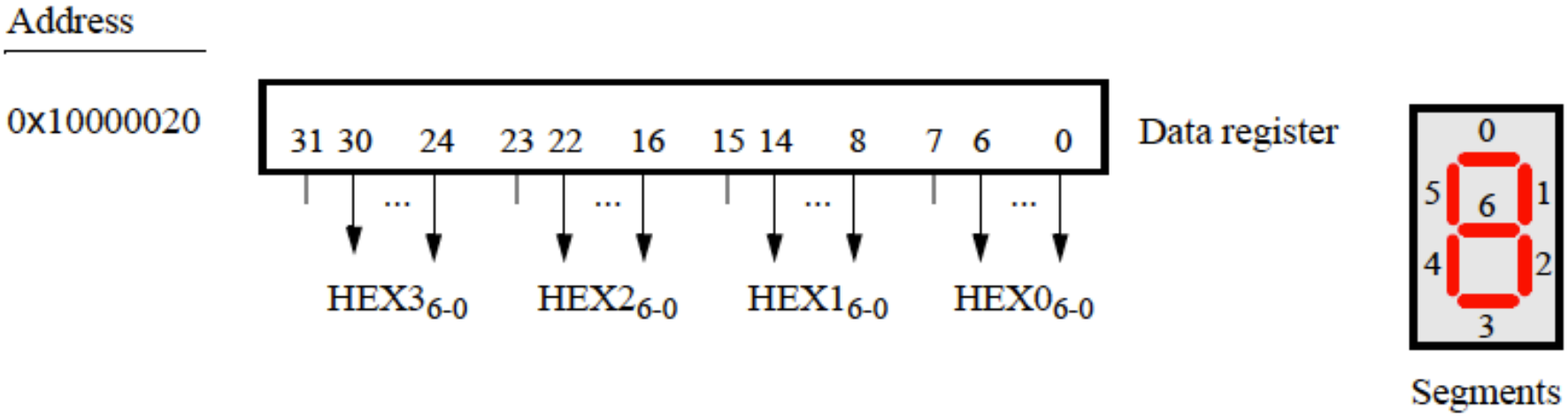
INTERFACES PARALELAS – DE2

- **Slider/Toggle switches**

- **Green LEDs**



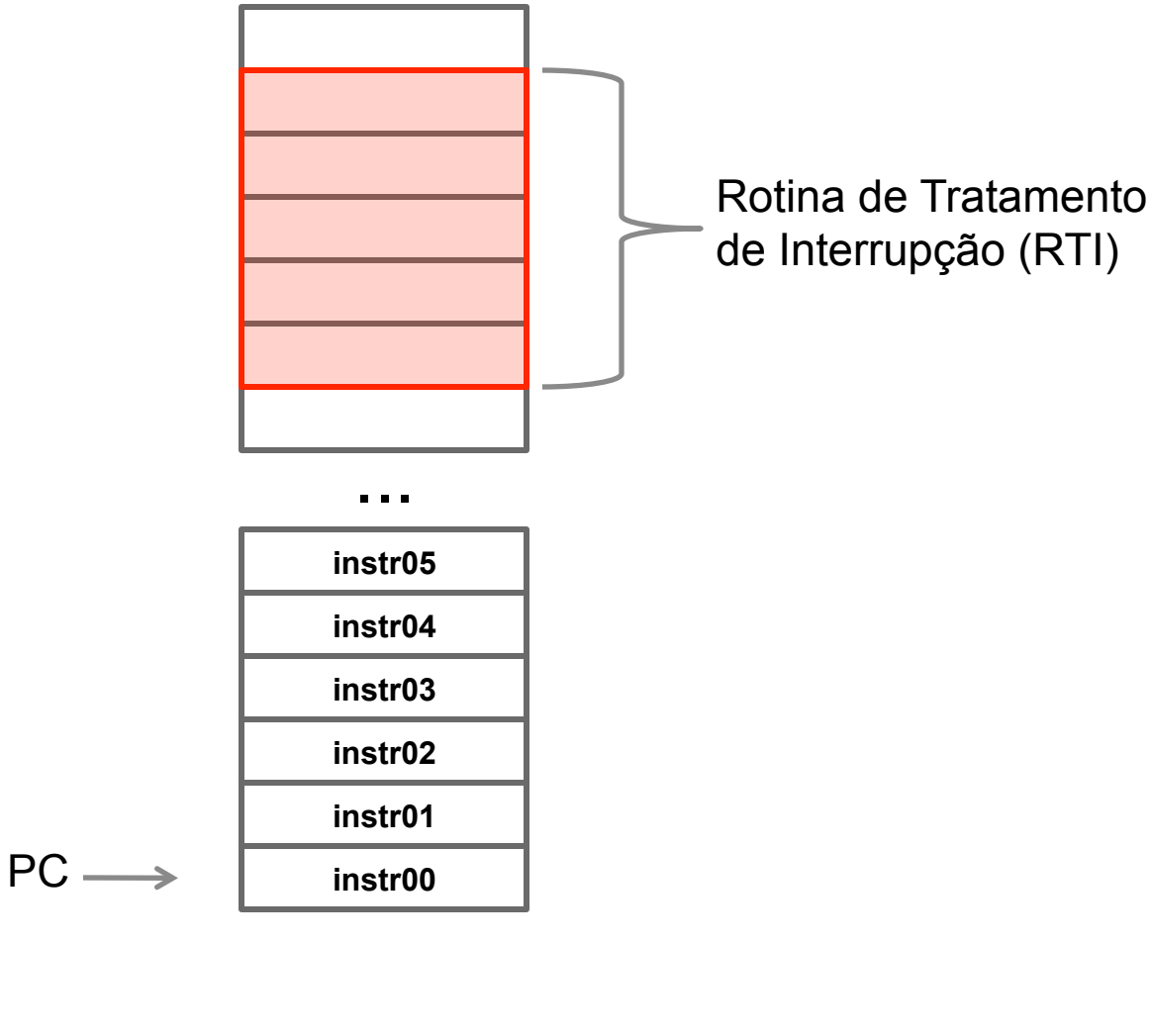
INTERFACE DISPLAY 7-SEG



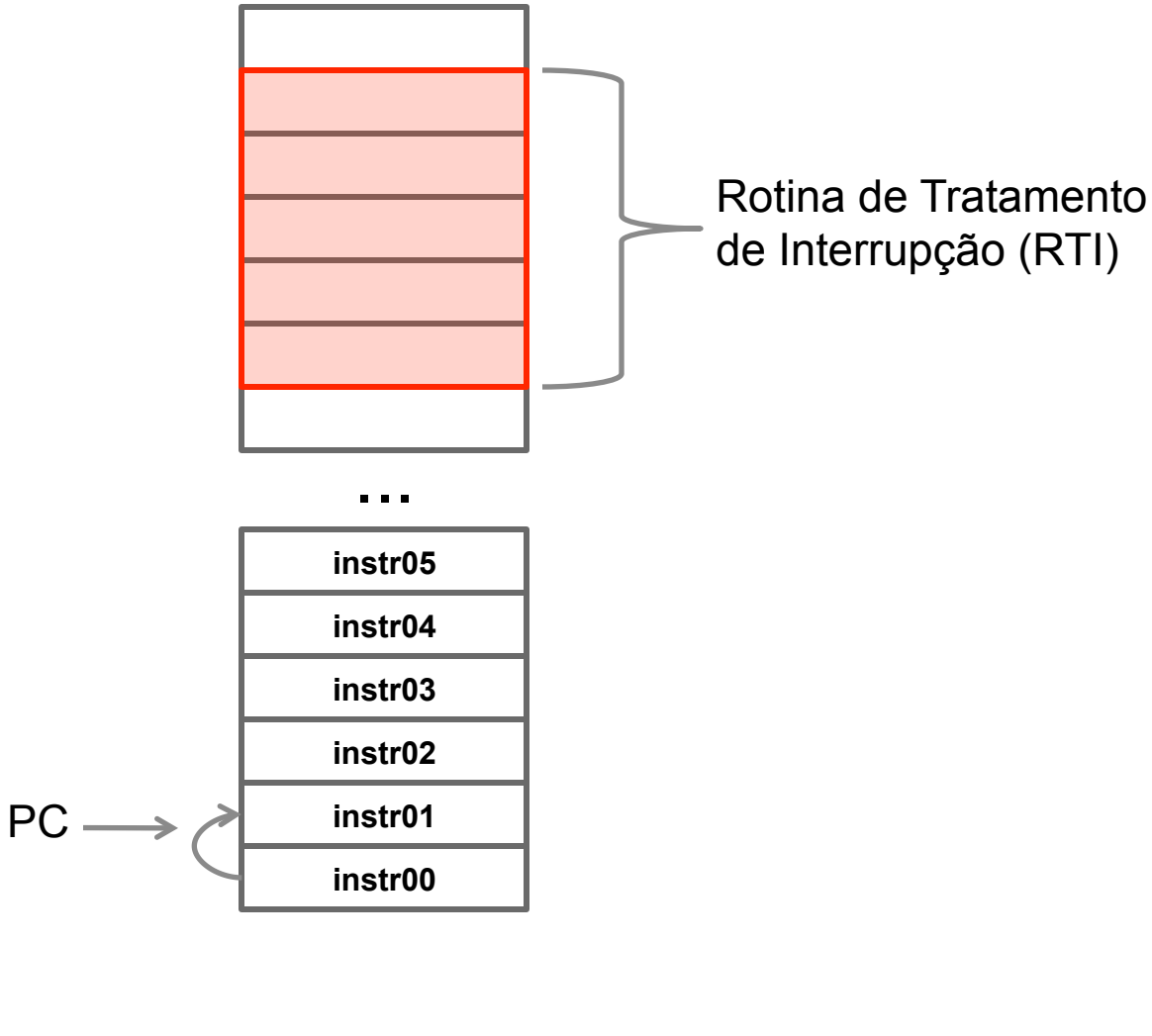
MÉTODO 2 - INTERRUPÇÃO

- **Dispositivo de I/O interrompe execução do processador para indicar que precisa de sua atenção**
  - Interrupção é assíncrona em relação à execução de instruções
- **Necessário alguma forma de identificar qual dispositivo gerou interrupção**
  - Interrupção vetorizada
    - A interrupção possui um ID que identifica a rotina específica para lidar com a interrupção
  - Uma única rotina de tratamento de exceção

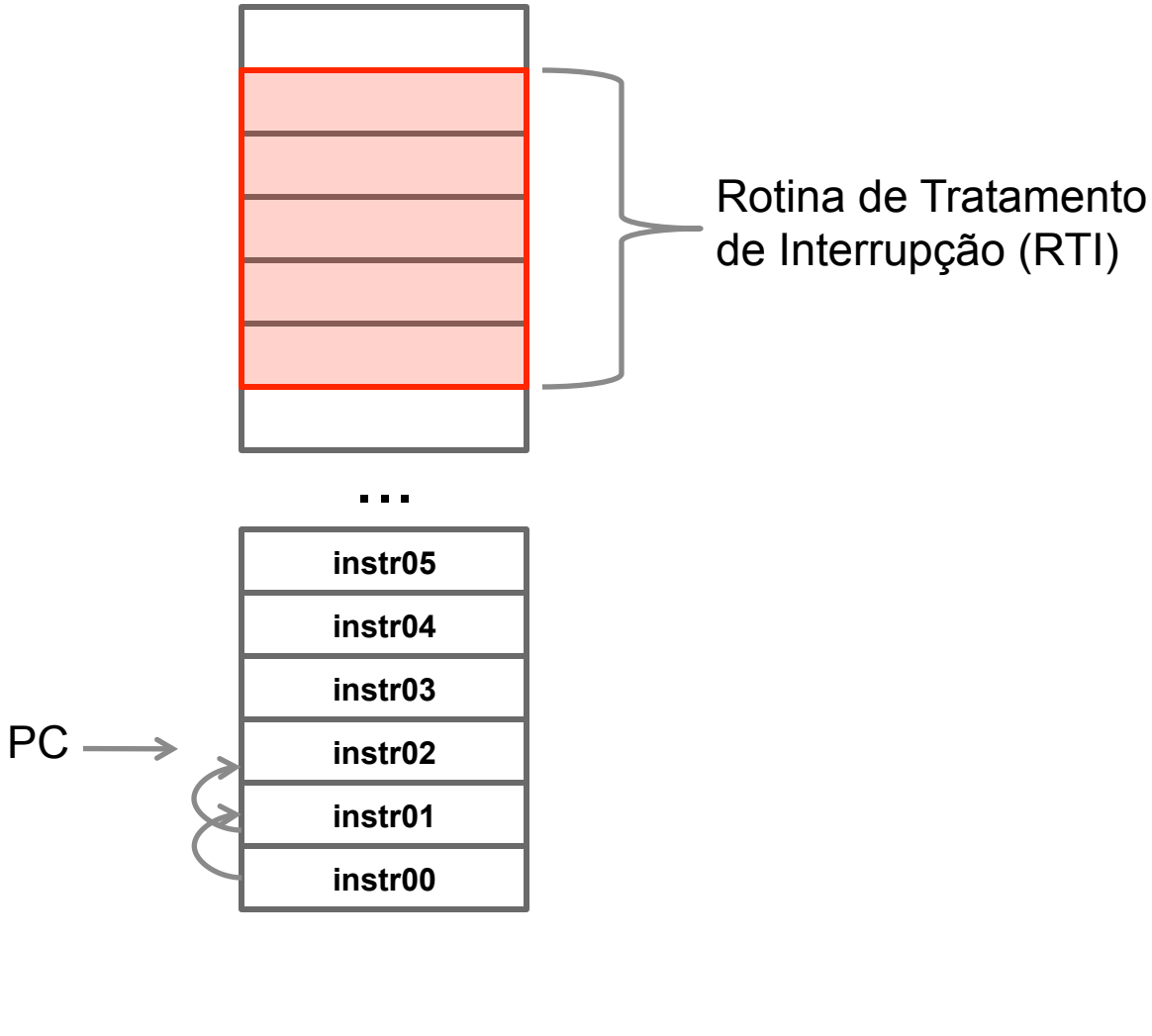
INTERRUPÇÃO



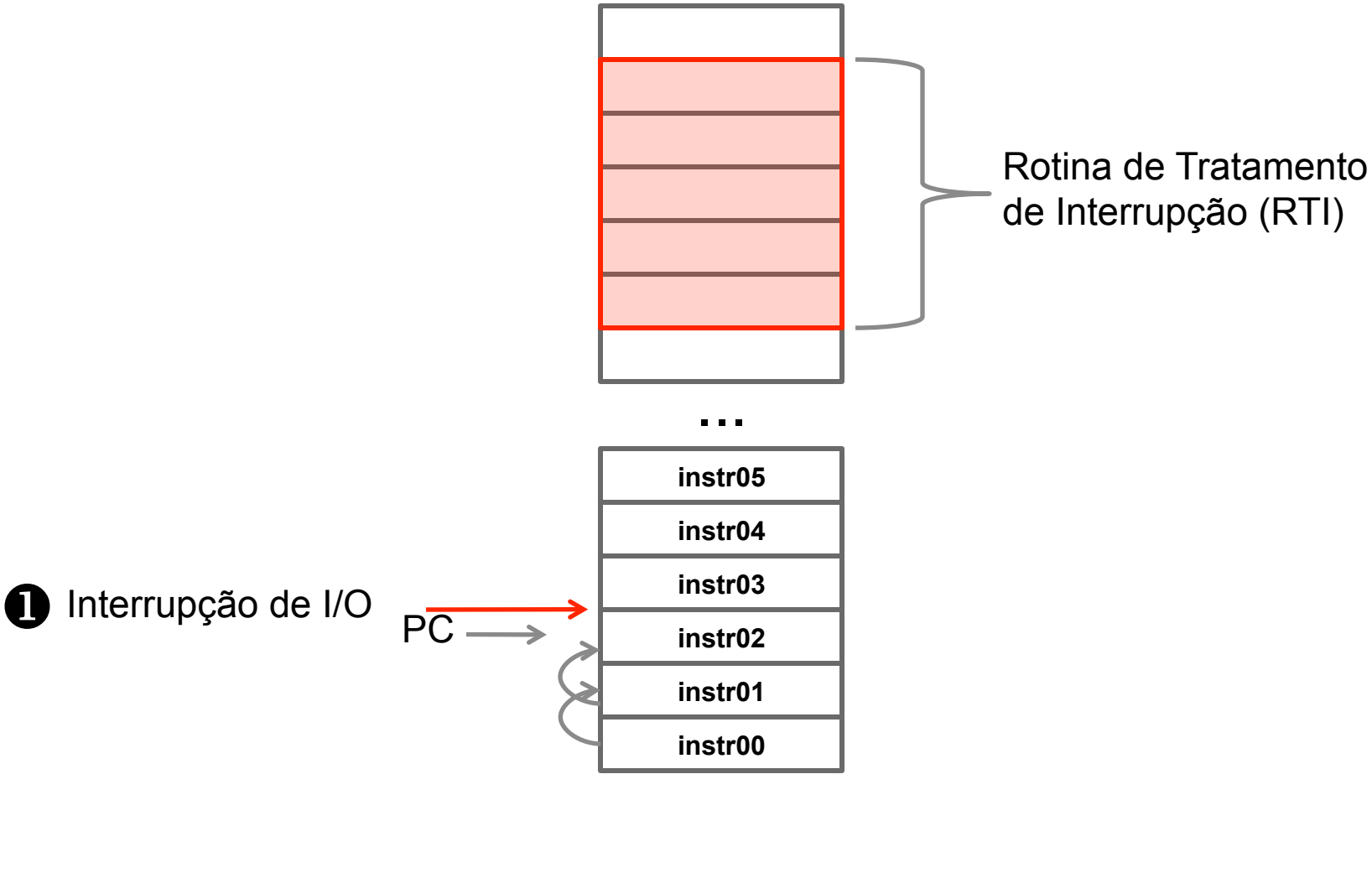
INTERRUPÇÃO



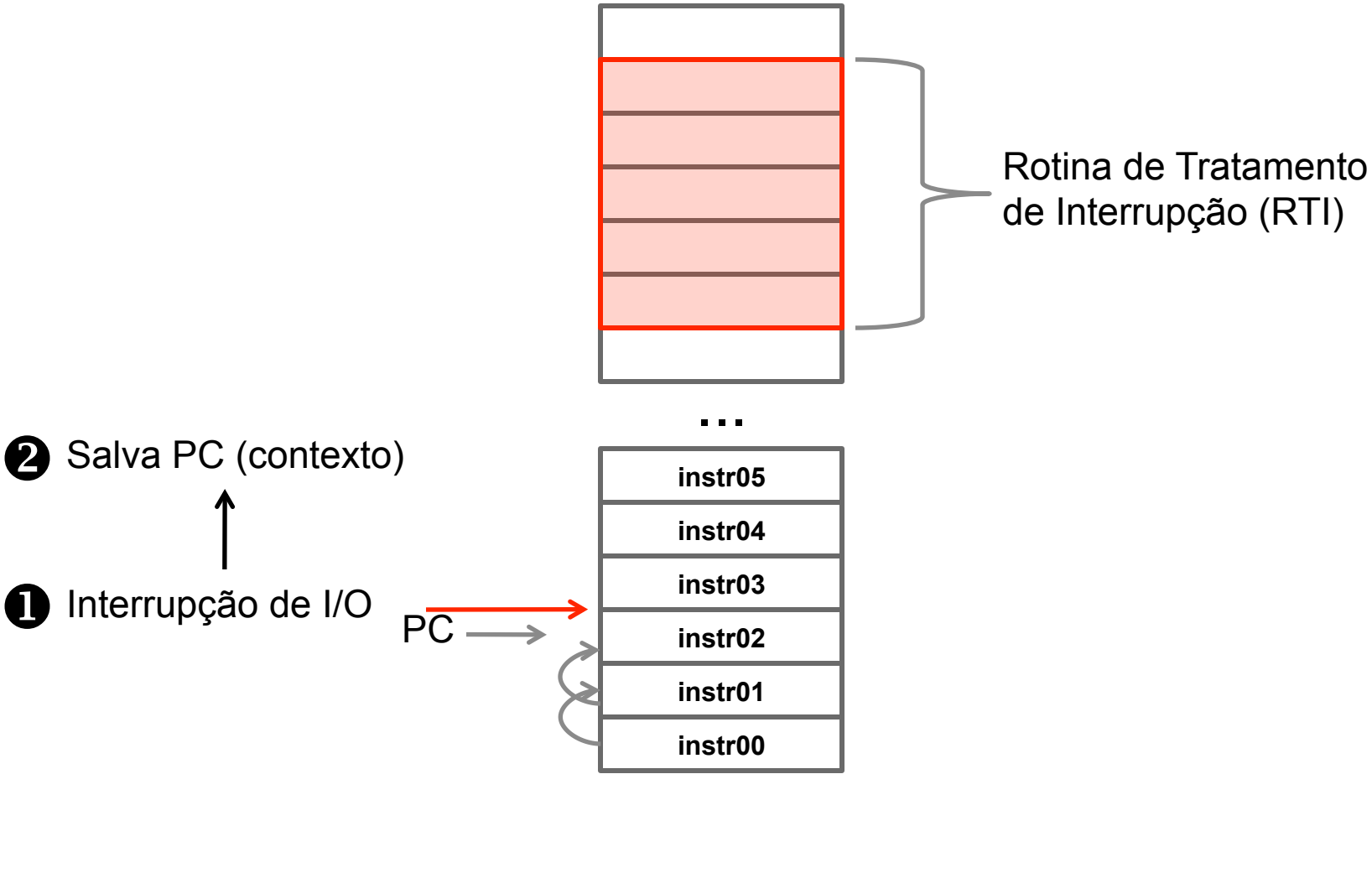
INTERRUPÇÃO



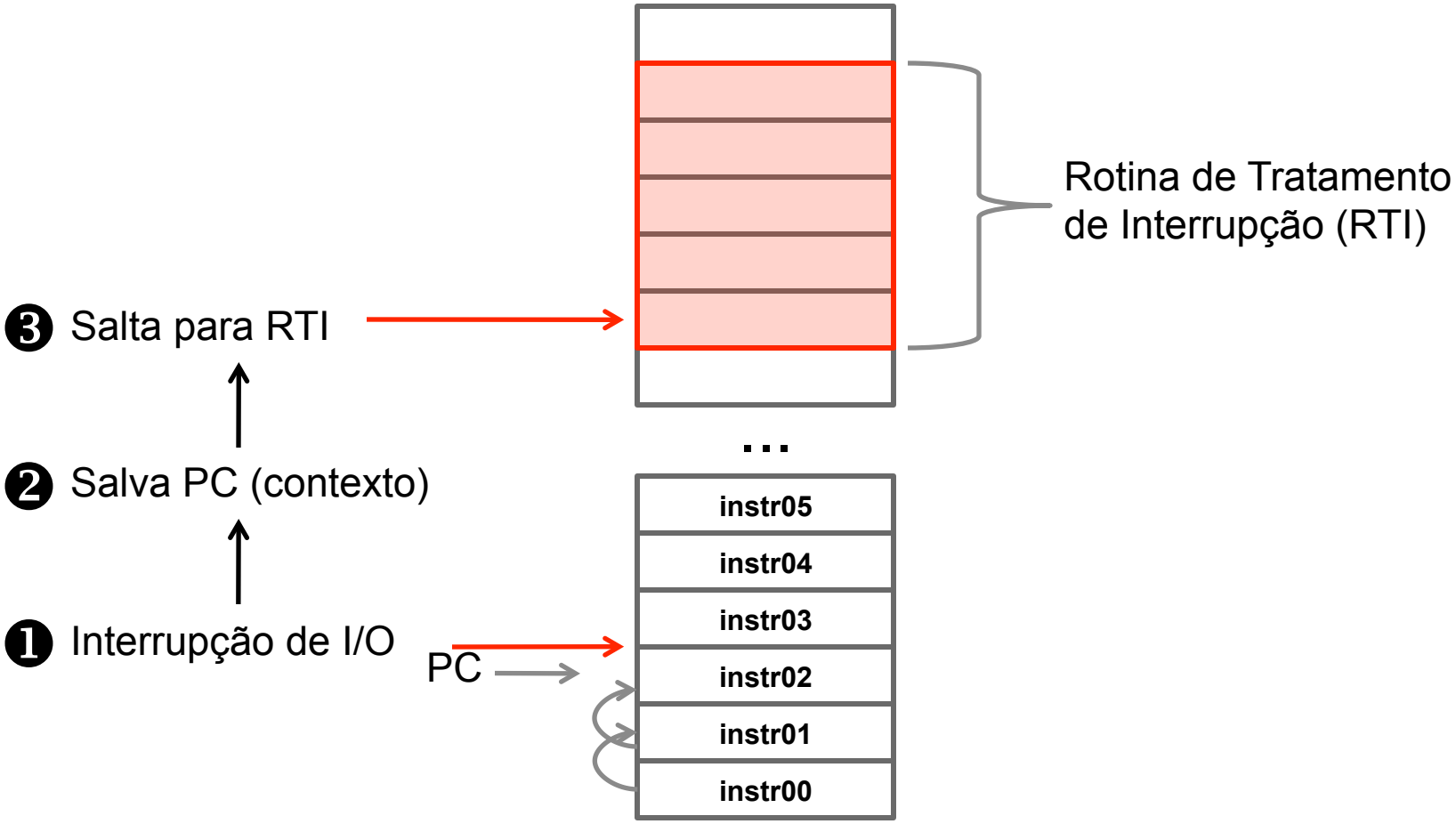
INTERRUPÇÃO



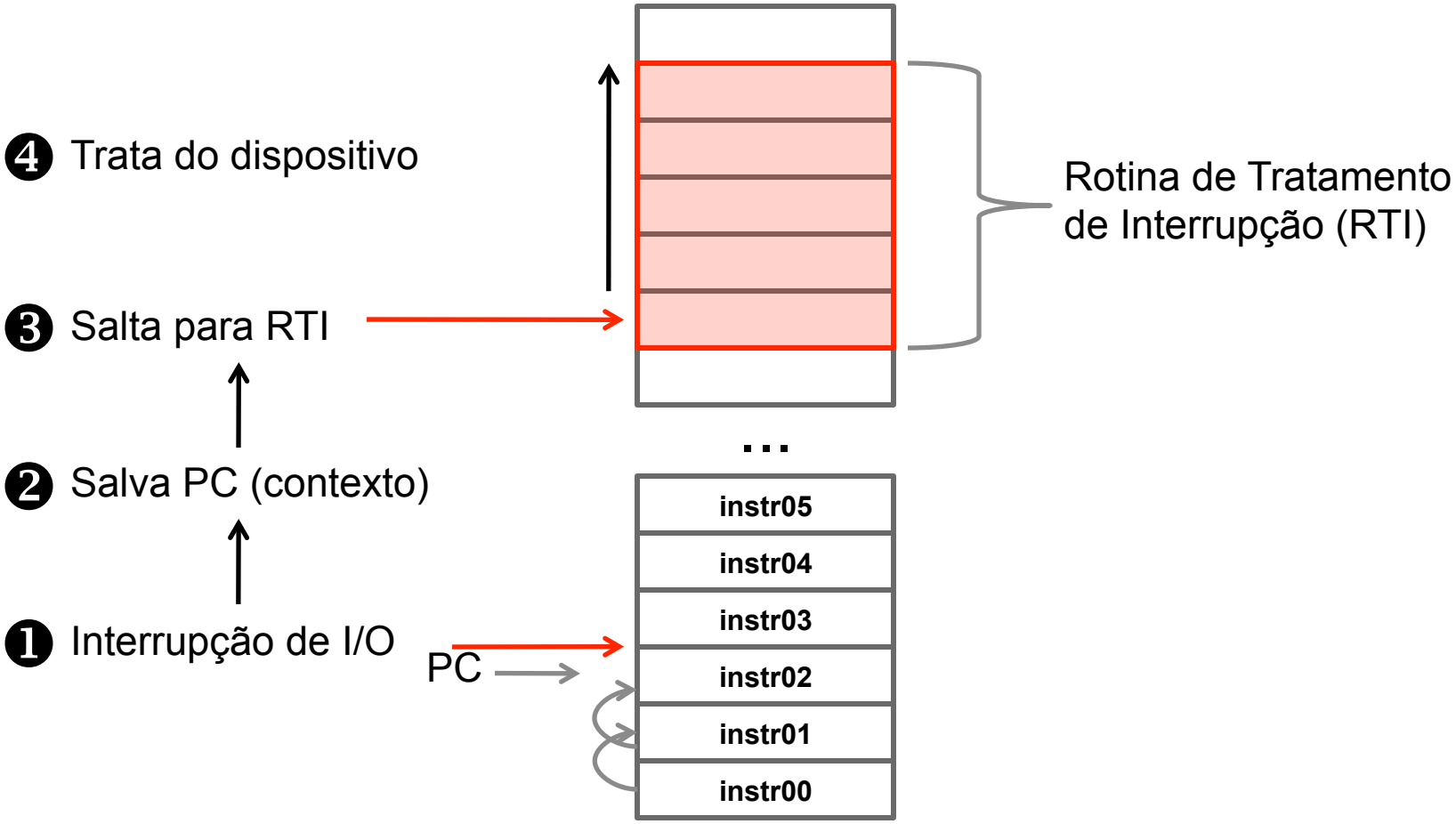
INTERRUPÇÃO



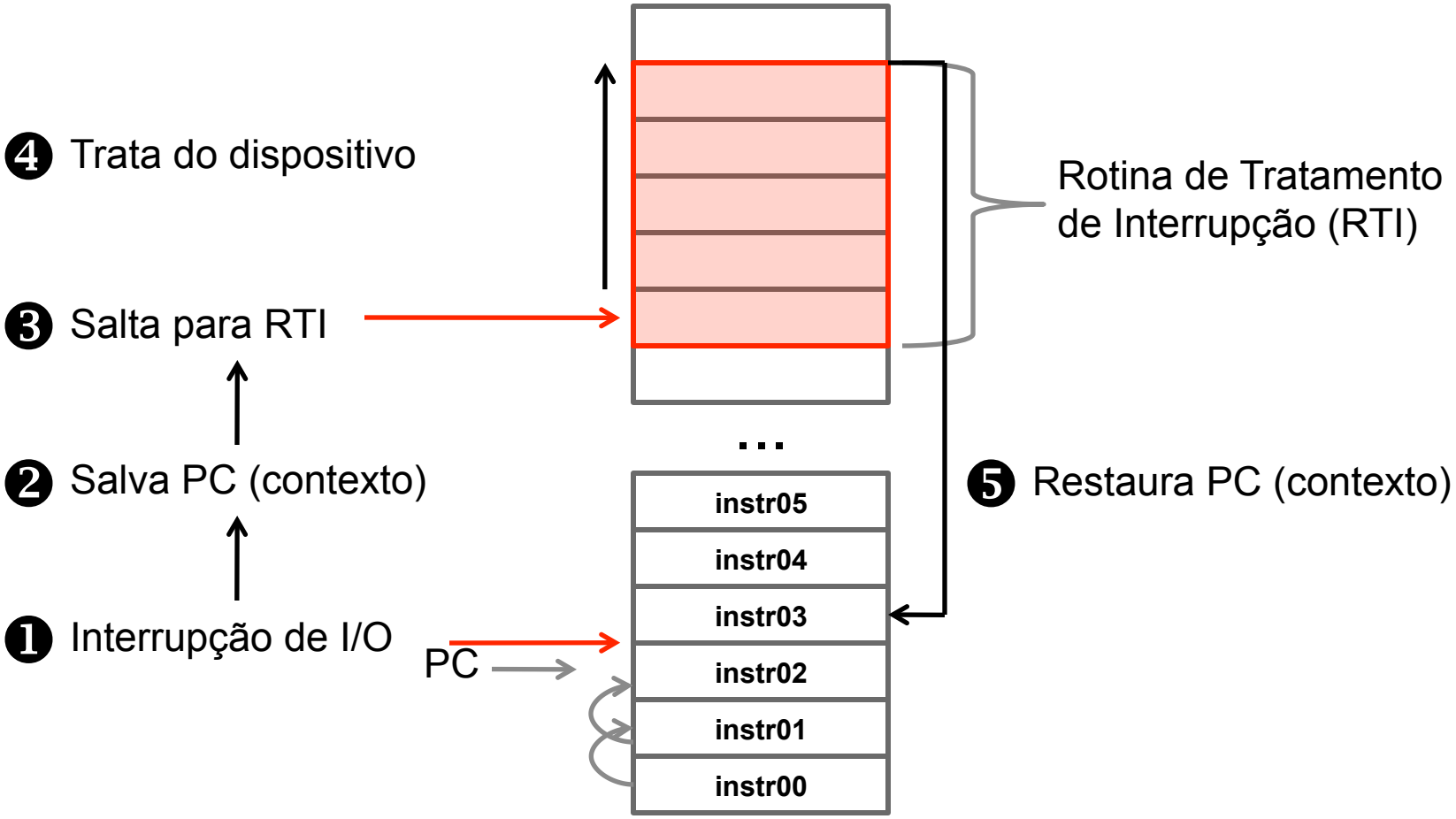
INTERRUPÇÃO



INTERRUPÇÃO

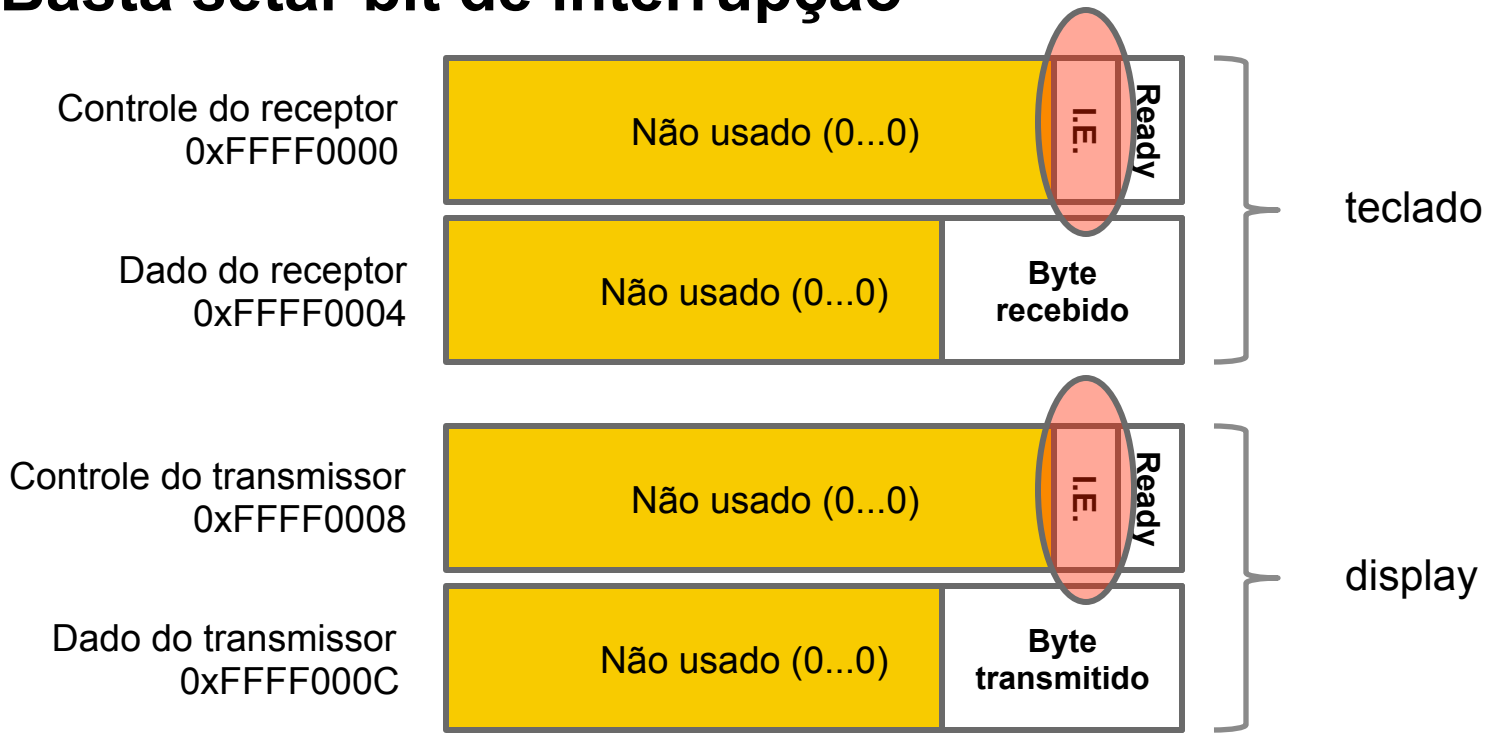


INTERRUPÇÃO



EXEMPLO USANDO INTERRUPÇÃO

- **I.E. = Interrupt Enable**
- **Basta setar bit de interrupção**



INTERRUPÇÃO

- **Vantagens**
  - Libera o processador da necessidade de checar pelo dispositivo constantemente (*polling*)
  - Programa do usuário é suspenso apenas quando a transferência de dados entre CPU e dispositivo é necessária
- **Desvantagens**
  - Hardware relativamente complexo necessário
  - Processador ainda é responsável por toda transferência

EXCEÇÕES NO DE2 BASIC COMPUTER

- **O endereço de reset do DE2 Basic Computer é 0x00000000**
  - Quando o pushbutton 0 (KEY0) é pressionado, o PC é automaticamente ajustado para 0x00000000
- **O endereço para a rotina de tratamento de exceção é 0x00000020**
- **Uma exceção pode ser gerada por:**
  - Instrução *trap*
  - Instrução não implementada
  - Divisão por zero
  - Interrupção de hardware

Mais informações no capítulo 3, página 3-30, da referência "*Nios II Processor Reference – Handbook*"

EXCEÇÕES NO DE2 BASIC COMPUTER

- **Registradores do Nios II relativos a exceções**

Registrador	Nome	Função
r24	et	Temporário para tratamento de exceção
r29	ea	Endereço de retorno da exceção

- **Para retornar da rotina de tratamento de exceção você deve usar a instrução eret**

REGISTRADORES DE CONTROLE – NIOS II

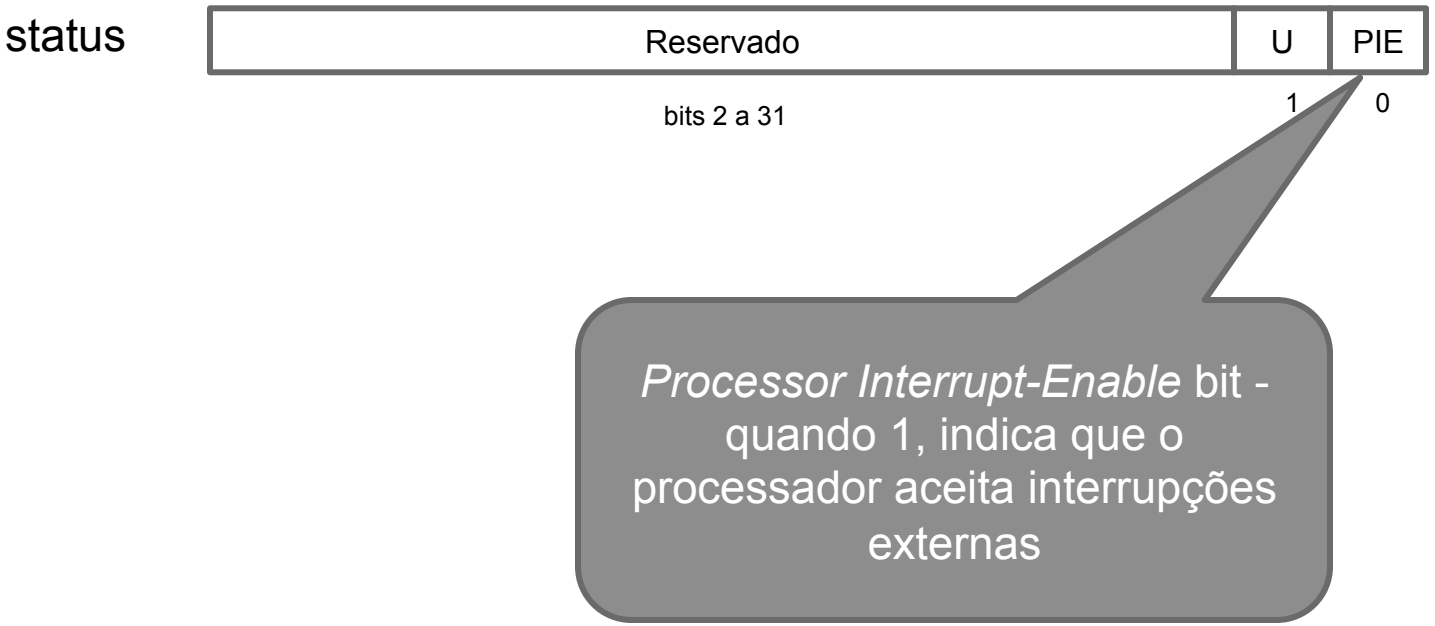
- **Registradores de controle úteis no DE2 Basic Computer**

Registrador	Nome	Função
ctl0	status	armazena vários status da CPU
ctl1	estatus	cópia de ctl0 durante exceção
ctl2	bstatus	cópia de ctl0 durante pausa de depuração
ctl3	ienable	habilitação de interrupção
ctl4	ipending	interrupções pendentes
ctl5	cpuid	identificador único do processador

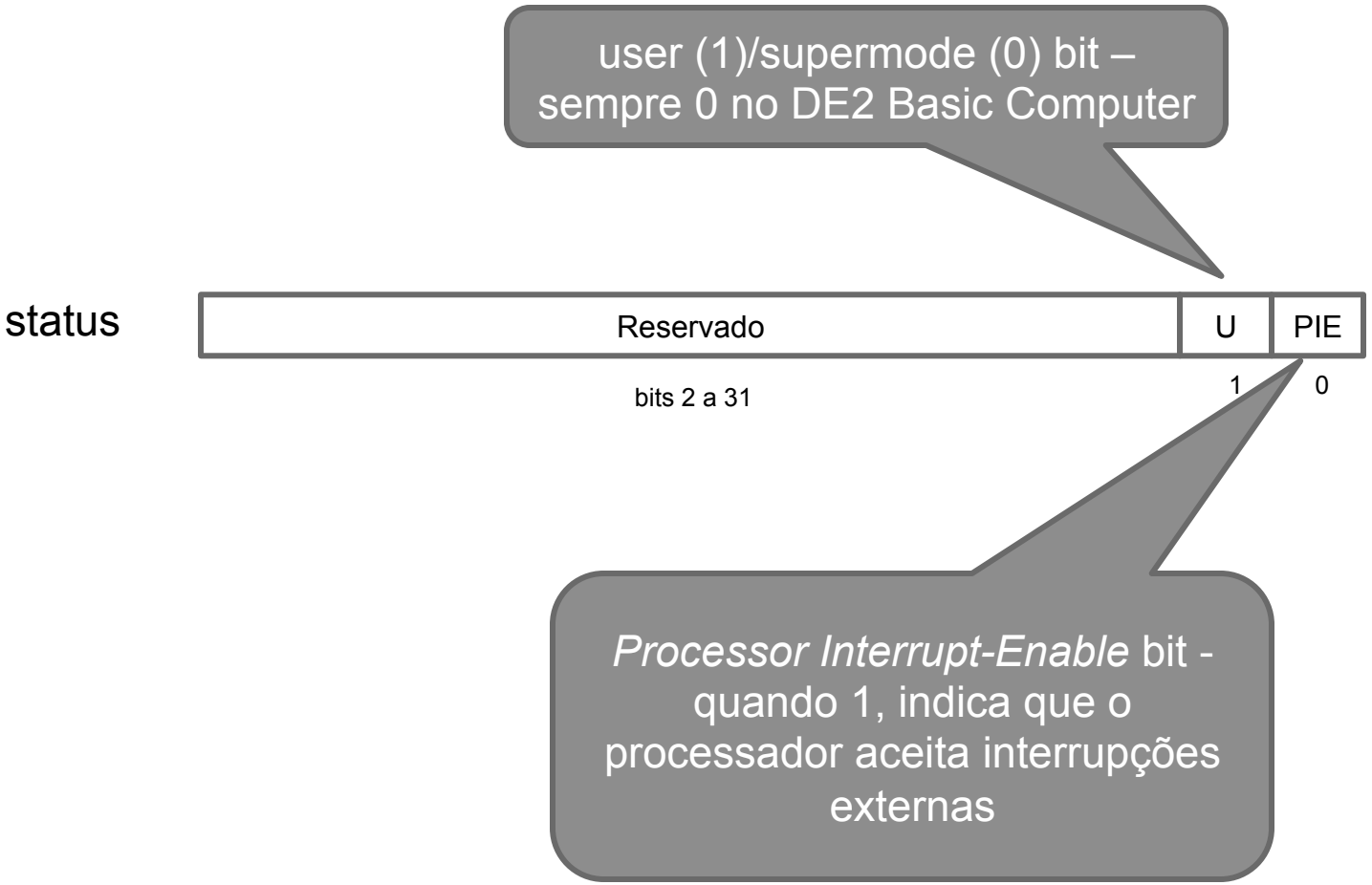
- **A escrita e leitura desses registradores devem ser feitas com as instruções *wrctl* e *rdctl*, respectivamente**

REGISTRADOR *STATUS* NO DE2 BASIC COMPUTER

REGISTRADOR *STATUS* NO DE2 BASIC COMPUTER



REGISTRADOR *STATUS* NO DE2 BASIC COMPUTER



REGISTRADORES DE INTERRUPÇÃO

- **O Nios II possui 32 entradas para pedido de interrupção (*irq0 – irq31*)**

- **A habilitação ou não de algumas dessas interrupções é feita através do registrador *ienable***
  - Se o valor do respectivo bit for 0, a interrupção não será considerada
  - Exemplo: se o bit 2 for igual a 0, o dispositivo ligado à irq2 não poderá gerar interrupção

- **O registrador *ipending* indica quais interrupções estão pendentes (ainda não foram tratadas)**

IRQS DO DE2 BASIC COMPUTER

I/O Peripheral	IRQ #
Interval timer	0
Pushbutton switch parallel port	1
JTAG port	8
Serial port	10
JP1 Expansion parallel port	11
JP2 Expansion parallel port	12

INTERRUPÇÕES NO DE2 BASIC COMPUTER

- **Uma interrupção de hardware é gerada se e somente se as seguintes 3 condições forem verdadeiras**
  1. O bit *PIE* do registrador *status* for 1
  2. Uma linha de pedido de interrupção, `irqn`, estiver ativa
  3. O bit *n* correspondente no registrador *ienable* for 1

- **Após desviar o controle para a rotina de interrupção, o processador limpa o bit *PIE*, desabilitando interrupções futuras**

- **O valor do registrador *ipending* indica quais interrupções estão pendentes**

TRATAMENTO DE EXCEÇÕES

- **Em resposta a um evento de exceção, o processador Nios II automaticamente realiza as seguintes ações:**
  1. Salva o status atual do processador ao copiar o registrador *status* (*ctl0*) para o *estatus* (*ctl1*)
  2. Limpa o bit U do registrador *status* para garantir que o processador esteja em modo supervisor
  3. Limpa o bit PIE do registrador *status* para desabilitar futuras interrupções por hardware
  4. Escreve o endereço da instrução seguinte a que gerou a exceção no registrador *ea* (*r29*)
  5. Transfere a execução para a rotina que trata a execeção (nosso caso, endereço 0x00000020)

Para maiores informações consulte o capítulo 11 do *Introduction to the Altera Nios II Soft Processor*

DETERMINANDO O TIPO DA EXCEÇÃO

- **Lembre-se que uma exceção pode ser causada por diferentes motivos**
  - Instrução *trap*
  - Instrução não implementada
  - Interrupção de hardware

- **Para determinar o tipo de exceção....**
  1. Leia o registrador *ipending* para checar se existe alguma interrupção de hardware – caso afirmativo, desvie para a rotina de tratamento de interrupção de hardware adequada
  2. Leia a instrução que estava sendo executada quando a exceção ocorreu – caso seja a *trap*, desvie para a rotina de tratamento de interrupção de software adequada
  3. Caso contrário a exceção é devida a uma instrução não implementada.

RETORNANDO DA EXCEÇÃO

- **A última instrução da rotina de tratamento de exceção deve ser a *eret***
  - Essa instrução, entre outras coisas, atribui ao PC o valor do registrador *ea*

- **Atenção!!!**
  - Se a exceção for devida a uma interrupção de hardware, o valor do registrador *ea* deve ser decrementado de 4
  - A instrução sendo executada quando do aparecimento da interrupção não é completada antes da rotina de exceção ser invocada

EXEMPLO DE TRATAMENTO DE EXCEÇÃO

```
.org      0x20
/* Exception handler */
          rdctl   et, ipending                  /* Check if external interrupt occurred */
          beq     et, r0, OTHER_EXCEPTIONS      /* If zero, check exceptions */
          subi    ea, ea, 4                     /* Hardware interrupt, decrement ea to execute the interrupted */
                                                /*   instruction upon return to main program */
          andi    r13, et, 2                    /* Check if irq1 asserted */
          beq     r13, r0, OTHER_INTERRUPTS     /* If not, check other external interrupts */
          call    EXT_IRQ1                      /* If yes, go to IRQ1 service routine */
OTHER_INTERRUPTS:
/* Instructions that check for other hardware interrupts should be placed here */
          br      END_HANDLER                   /* Done with hardware interrupts */
OTHER_EXCEPTIONS:
/* Instructions that check for other types of exceptions should be placed here */
END_HANDLER:
          eret                                  /* Return from exception */
.org      0x100
/* Interrupt-service routine for the desired hardware interrupt */
EXT_IRQ1:
/* Instructions that handle the irq1 interrupt request should be placed here */
          ret       /* Return from the interrupt-service routine */
```

Página 25 do *"Introduction to the Altera Nios II Soft Processor"*

INTERFACE PUSHBUTTON