

## Aula 8

- O sistema de interrupções do PIC32

José Luís Azevedo, Bernardo Cunha, Tomás O. Silva, P. Bartolomeu

# Interrupções no PIC32

- O PIC32 pode ser configurado em um de dois modos:
  - **Single-vector mode** – um único vetor (0) para todas as fontes de interrupção, ou seja, identificação da fonte por software
  - **Multi-vector mode** – Interrupções vetorizadas (vetores definidos pelo fabricante para todas as fontes – ver PIC32MX7XX Family Data Sheet – Interrupt Controller)
- **Na placa DETPIC32 o sistema de interrupções está configurado para "multi-vector mode"**
- O sistema de interrupções do PIC32 é baseado num módulo de gestão exterior ao CPU (controlador de interrupções)
  - Até 96 fontes de interrupção (75 no PIC32MX7xx) das quais 5 fontes externas com configuração de transição ativa (*rising* ou *falling edge*)
  - Até 64 vetores (51 no PIC32MX7xx)
- O controlador de interrupções permite, entre outras coisas, a configuração das prioridades de cada fonte
  - Funciona como um **priority encoder** enviando para o CPU o pedido pendente de maior prioridade (identificado com vetor e prioridade)

# Tabela de vetores (multi-vector mode)

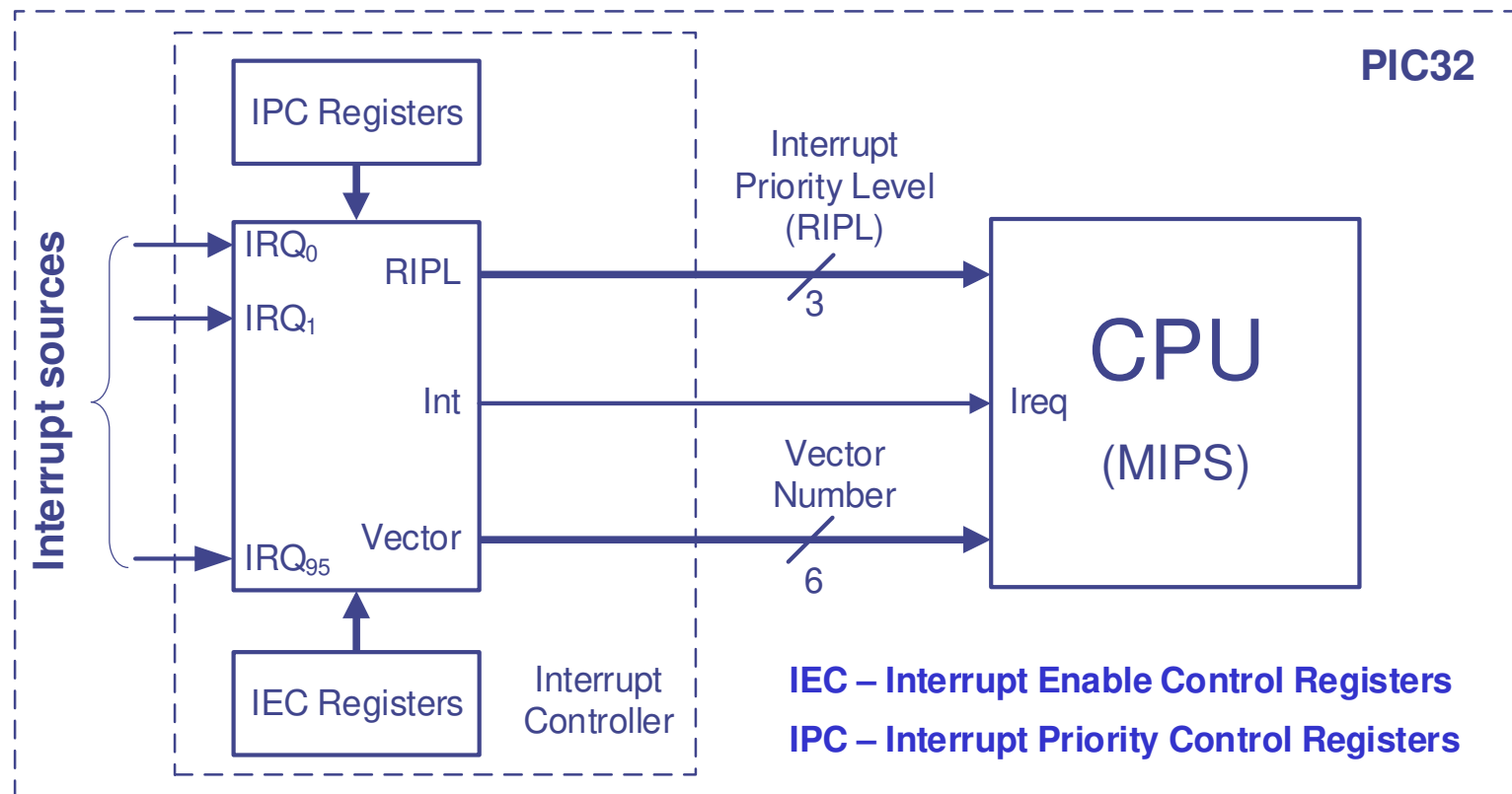
- **Tabela inicializada com instruções de salto para as RSI:** são colocadas na tabela de vetores, instruções de salto para as RSI
- No processamento da interrupção o CPU usa o vetor para calcular o endereço da tabela de vetores e faz um salto para esse endereço (ou seja,  $PC \leq$  endereço calculado)
- Cada posição da tabela tem, em geral, uma instrução de salto incondicional para a RSI a executar
- O espaçamento entre elementos da tabela pode ser configurado para 32, 64, 128, 256 ou 512 bytes (valor por defeito: 32 = 0x20)

Vector 0	→	0x9D000200	j	0x9D001C00	→	Salto para a RSI situada no endereço 0x9D001C00
		0x9D000204	nop			
Vector 1	→	0x9D000220	j	0x9D001D08		
		0x9D000224	nop			
Vector 2	→	0x9D000240	j	0x9D0020C4	→	Salto para a RSI situada no endereço 0x9D0020C4
		0x9D000244	nop			

- **Exemplo:** vetor=2, base\_address=0x9D000200, espaçamento=32  
 $\text{endereço\_tabela} = \text{vetor} * 0x20 + 0x9D000200 = 0x9D000240$

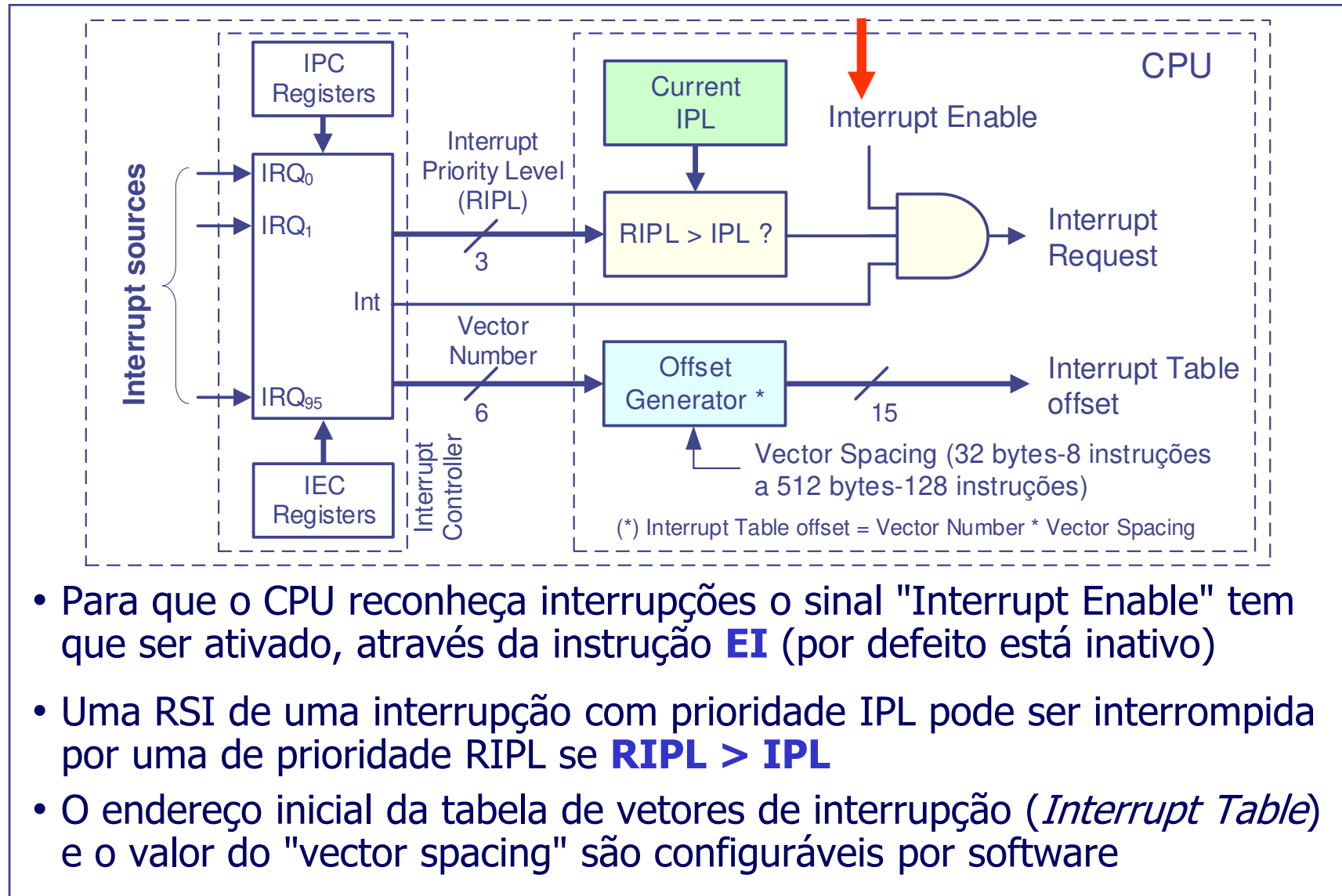
# Interrupções no PIC32

- Fontes de interrupção:
  - Internas: até 91, de periféricos internos; externas: 5



- O pedido pendente com maior prioridade é encaminhado para o CPU (identificado pelo vetor e pela prioridade – RIPL)

# Interrupções no PIC32



- Para que o CPU reconheça interrupções o sinal "Interrupt Enable" tem que ser ativado, através da instrução **EI** (por defeito está inativo)
- Uma RSI de uma interrupção com prioridade IPL pode ser interrompida por uma de prioridade RIPL se **RIPL > IPL**
- O endereço inicial da tabela de vetores de interrupção (*Interrupt Table*) e o valor do "vector spacing" são configuráveis por software

# Interrupções no PIC32

- **IEC0, IEC1, IEC2** – Interrupt Enable **Control Registers**
  - Registos através dos quais se pode habilitar / desativar (*enable / disable*) qualquer fonte de interrupção. Cada módulo do PIC32 que pode gerar interrupções usa 1 ou mais bits destes registos
- **IPC0, IPC1, ..., IPC12** – Interrupt Priority **Control Registers**
  - Registos através dos quais se pode configurar, com 3 bits, a prioridade de cada uma das fontes de interrupção (0 a 7)
- **IFS0, IFS1, IFS2** – Interrupt Flag **Status Registers**
  - Flags de sinalização da ocorrência de interrupções, de todas as fontes possíveis. Cada módulo do PIC32 que pode gerar interrupções usa 1 ou mais bits destes registos
- **INTCON** – Interrupt **Control Register**
  - Configura a polaridade da transição ativa das fontes de interrupção externa (rising edge / falling edge)

# Interrupções no PIC32

- Cada fonte de interrupção tem associado um conjunto de bits de configuração e de status
- **Interrupt Enable Bit** – bit definido num dos registos **IECx** (Interrupt Enable Control Registers), através do qual se pode fazer o *enable* ou o *disable* de uma dada fonte de interrupção. O nome do bit é, normalmente, formado pela sigla identificativa da fonte, terminada com o sufixo **IE** (e.g. **T1IE**, *Timer1 Interrupt Enable*)
- **Priority Level** – conjunto de 3 bits definido num dos registos **IPCx** (Interrupt Priority Control Registers), designado através da sigla da fonte com o sufixo **IP** (e.g. **T1IP**, *Timer1 Interrupt Priority*)
  - 7 níveis de prioridade (1 a 7); a prioridade 0 significa fonte *disabled*
- **Interrupt Flag** – bit definido num dos registos **IFSx** (Interrupt Flag Status Registers) e designado com a sigla da fonte com o sufixo **IF** (e.g. **T1IF**, *Timer1 Interrupt Flag*). Este bit é ativado automaticamente quando ocorre uma interrupção. A desativação é da responsabilidade do programador

## Exemplo de uma tabela de vetores no PIC32

#vector\_7 (INT1, External Interrupt 1)

0x9D0002E0            0x0B40074D    j            0x9D001D34

0x9D0002E4            0x00000000    nop

#vector\_8 (T2 - Timer2)

0x9D000300            0x0B4006C3    j            0x9D001B0C

0x9D000304            0x00000000    nop

#vector\_19 (INT4 - External Interrupt 4)

0x9D000460            0x0B40077A    j            0x9D001DE8

0x9D000464            0x00000000    nop

- Na placa DETPIC32 o endereço inicial da tabela de vetores (a que corresponde o vetor 0) é 0x9D000200.



# Rotina de Serviço à Interrupção no PIC32

**TABLE 7-1: INTERRUPT IRQ, VECTOR AND BIT LOCATION**

Interrupt Source <sup>(1)</sup>	IRQ Number	Vector Number	Interrupt Bit Location			
			Flag	Enable	Priority	Sub-Priority
Highest Natural Order Priority						
CT – Core Timer Interrupt	0	0	IFS0<0>	IEC0<0>	IPC0<4:2>	IPC0<1:0>
CS0 – Core Software Interrupt 0	1	1	IFS0<1>	IEC0<1>	IPC0<12:10>	IPC0<9:8>
CS1 – Core Software Interrupt 1	2	2	IFS0<2>	IEC0<2>	IPC0<20:18>	IPC0<17:16>
INT0 – External Interrupt 0	3	3	IFS0<3>	IEC0<3>	IPC0<28:26>	IPC0<25:24>
T1 – Timer1	4	4	IFS0<4>	IEC0<4>	IPC1<4:2>	IPC1<1:0>

Fonte

Interrupt Function

Vector Number

Function Name

```
void _int_( 4 ) isr_t1(void)
{
    ...
    IFS0bits.T1IF = 0; // Reset T1 Interrupt Flag
}
```

TABLE 7-1: [PIC32MX7XX Family Data Sheet#page 74 - 76]

## Exemplo de programação com interrupções no PIC32

```
int main(void)
{
    configIO();           // Config IO and Interrupt
                          // Controller
    EnableInterrupts();   // Enable Interrupt System. Macro
                          // definida em detpic32.h como:
                          //      asm volatile("ei")

    while(1)
    {
        ...
    }
    return 0;
}

// IO Configuration function
void configIO(void)
{
    ...
    IFS0bits.T1IF = 0;    // Reset Timer 1 interrupt flag
    IPC1bits.T1IP = 2;    // Set priority level to 2
    IEC0bits.T1IE = 1;    // Enable Timer 1 interrupts
    ...
}
```

# Exemplo de programação com interrupções no PIC32

```
// Interrupt Service routine - Timer1
void __int__( 4 ) isr_t1(void)
{
    ...
    IFS0bits.T1IF = 0;    // Reset Timer 1 Interrupt Flag
}

// Interrupt Service routine - External Interrupt 1
void __int__( 7 ) isr_ext_int1(void)
{
    ...
    IFS0bits.INT1IF = 0; // Reset External Interrupt 1 Flag
}

// Interrupt Service routine - External Interrupt 4
void __int__( 19 ) isr_ext_int4(void)
{
    ...
    IFS0bits.INT4IF = 0; // Reset External Interrupt 4 Flag
}
```