Módulo 5 – Interrupções

Uma interrupção é um mecanismo que permite que um evento externo (ou até mesmo interno) pare temporariamente a execução normal do programa principal para lidar com uma tarefa de maior prioridade.

As vezes, precisamos tratar de eventos no momento em que eles ocorrem. Na execução padrão do sistema, cada evento tem seu momento de ser executado (todos os eventos ficam em loop dentro do while). Mas é possivel definir uma rotina que é executada independente do momento que será acionada por um evento. Os eventos que geram interrupções no PIC18F45k22 são:

- Estouro do timer - permite criar interrupções periódicas;
- Interrupção externa - gera uma interrupção por um pulso externo chegando aos pinos INTx (INT0, INT1, INT2);
- PORTB – gera interrupção por uma mudança no PORTB;
- Interrupção por conversão A/D – Gera uma interrupção após um evento ADC;
- Interrupção por comunicação serial – Enviar ou receber dados pela serial (UART, SPI e I2C);
- Interrupção por leitura da EEPROM interna

Para cada um destes eventos, é possível criar uma rotina de software que é executada a parte. As funções interrupções são chamadas automaticamente. Depois, o programa volta a sua execução de onde parou:

```
// Função de interrupção de alta prioridade
void __interrupt(high_priority) INT_NAME(void){
      // Tratamento
}

// Função de interrupção de baixa prioridade
void __interrupt(low_priority) INT_NAME(void){
      // Tratamento
}
```

Escrever código dentro das funções de interrupções significa que aquele trecho de código ficará armazenado nos vetores de interruções do microcontrolador.

Para alguns microcontroladores, interrupções possuem níveis de prioridade, podendo ser de alta ou baixa prioridade. Uma interrupção de alta prioridade pode parar uma interrupção de baixa prioridade que já estava sendo executada.

Como criar uma interrupção

Cada periférico (timer, uart, INT) que for compatível com interrupções terá 3 bits principais para habilitar sua interrução:

- IF – Flag de interrupção – Sinaliza que o evento ocorreu
- IP – Prioridade de interrupção – Define um nivel de prioridade
- IE – Habilita a interrupção

Estes bits estão espalhados pelos 19 registradores relacionados a interrupções. Vamos pegar, por exemplo, a habilitação da interrupção por um pulso externo: INT1.

Primeiro habilitamos os níveis de prioridades, esta é um função geral:

```
RCONbits.IPEN = 1;
```

Os pinos que possuem esta interrupção disponível são:

21	18	RB0	AN12			SRI		CCP4 FLT0		$\overline{SS}$2		INT0	Y	
22	19	RB1	AN10	C12IN3-				P1C		SCK2 SCL2		INT1	Y	
23	20	RB2	AN8		CTED1			P1B		SDI2 SDA2		INT2	Y	

Encontra-se, no capítulo de interrupções do datasheet, os bits IF, IP e IE para a interrupção por INT1:

```
INTCON3bits.INT1IP = 1;
INTCON3bits.INT1IE = 1;
INTCON3bits.INT1IF = 1;
```

Também é necessário habilitar as interrupções globais:

```
INTCONbits.GIE = 1;
INTCONbits.PEIE = 1;
```

Por fim, basta tratar a interrupção dentro do vetor interrupt:

```
// Função de interrupção de alta prioridade
void __interrupt(high_priority) INT_NAME(void){
      // Tratamento da Interrupção
      INTCON3bits.INT1IF = 0;
}
```

Veja que é necessário resetar (colocar em 0) o flag IF ao fim da interrupção, pois é ele quem de fato acionará a interrupção dentro do PIC (quando estiver em 1). Isto vale para qualquer evento de interrupção, não somente o INT1.

Você pode ter uma função interrupção de alta prioridade e outra função de baixa prioridade em seu código sem nenhum problema. Mas ao se deparar com o problema de precisar tratar duas interruções diferentes de uma mesma prioridade, você não poderá escrever duas funções interrupção de uma mesma prioridade.

Para resolver este problema, utilize o flag IF para saber qual das interrupções foi acionada e trave toda a rotina dentro de um if else comum:

```
void __interrupt(high_priority) INT0_interrupt(void){
      // Caso a interrupção seja INT0
      if(INTCONbits.INT0IF){
      }

      // Caso a interrupção seja INT1
      else if(INTCON3bits.INT1IF){
      }
}
```