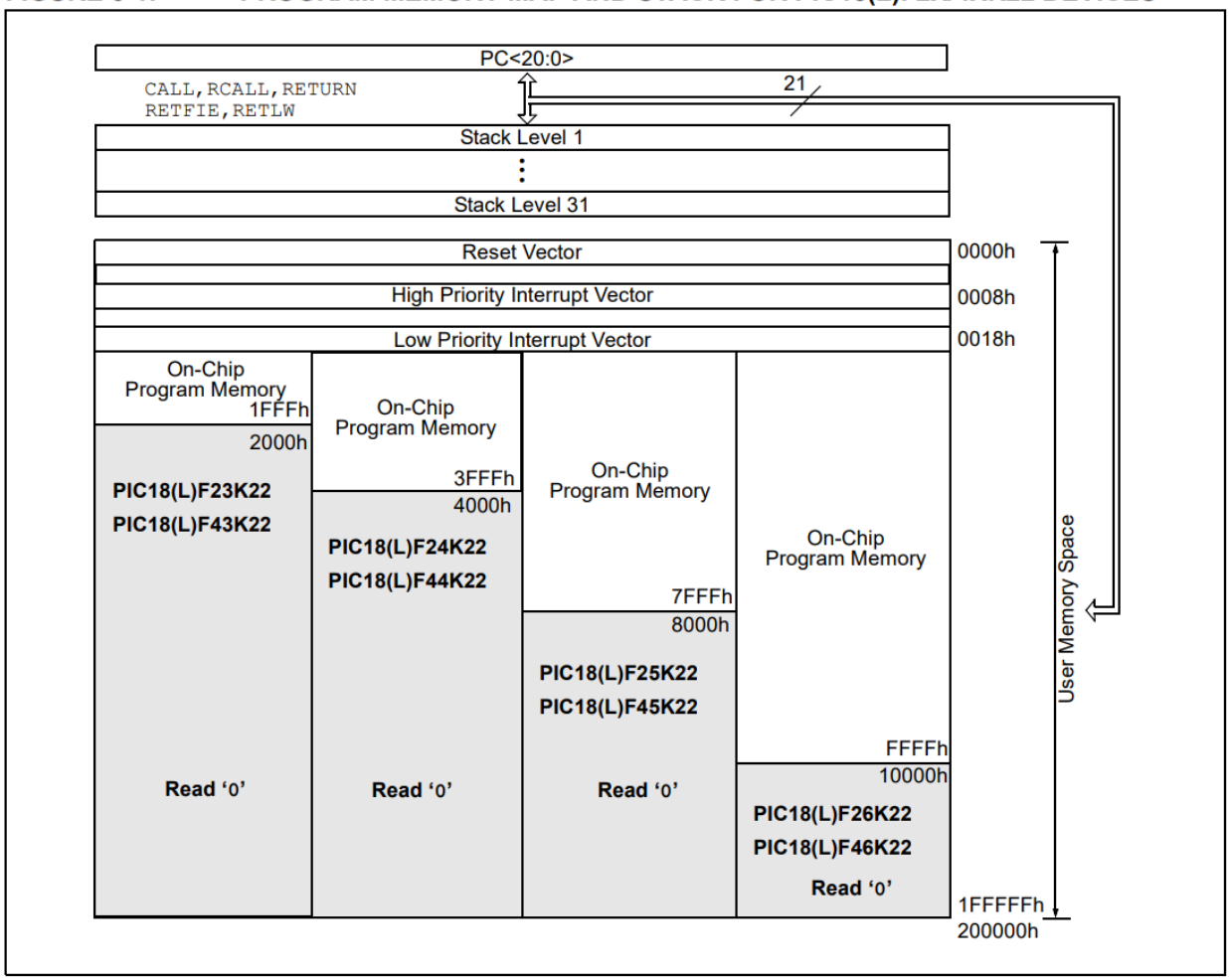
Interrupções

As interrupções são eventos não esperados pelo microcontrolador. Vimos que para o microcontrolador identificar o estado lógico do pino durante o acionamento, ele precisa estar constantemente lendo aquele pino. Mas o que acontece se o programador definir outra rotina para o microcontrolador onde não é possível ficar lendo um pino específico, mas que mesmo assim ele precisaria identificar aquele acionamento? Utilizamos interrupções.

Em outras palavras, uma interrupção permite o tratamento imediato a um evento de maior prioridade.

A interrupção interrompe o processo do microcontrolador. Ela altera o contador de programa (PC) para os endereços de rotina de interrupção. É como se fosse uma rotina separada do microcontrolador.

FIGURE 5-1: PROGRAM MEMORY MAP AND STACK FOR PIC18(L)F2X/4XK22 DEVICES



Isso é visto também no código. A interrupção chama automaticamente um tipo de função especial dentro do código. É nela que você definirá o programa a ser seguido em caso de uma interrupção. Ao fim deste programa, o PC retorna automaticamente de onde parou.

Interrupções também 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.

Dentro do código, as funções recebem um nome especial, são elas

```
// 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
}
```

O PIC18F45K22 possui 19 registradores dedicados à interrupção. Portando, se trata de um assunto complicado e que será estudado ao longo de todo o curso. Neste módulo estudaremos o básico de interrupções.

O registrador RCON

Este registrador tem como função principal salvar eventos de RESET do microcontrolador. Mas o bit 7 deste registrador contém a funcionalidade IPEN, que habilita os níveis de interrupção (alto e baixo). Este registrador foi criado porque microcontroladores da família PIC16F não possui esses níveis de interrupção. E isso ajudaria a deixar o software mais portável.

Então para nossos códigos, vamos manter esta funcionalidade sempre habilitada

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

Como criar uma interrupção

Uma interrupção deve ser iniciada por um periférico que suporta interrupção. Periféricos são todas as funcionalidades presentes em seu microcontrolador, como timers, conversores, comunicação, módulo CCP, entre outros. Por isso que interrupção é um tópico que deve ser estudado ao longo de todo o curso.

Interrupção por INT0

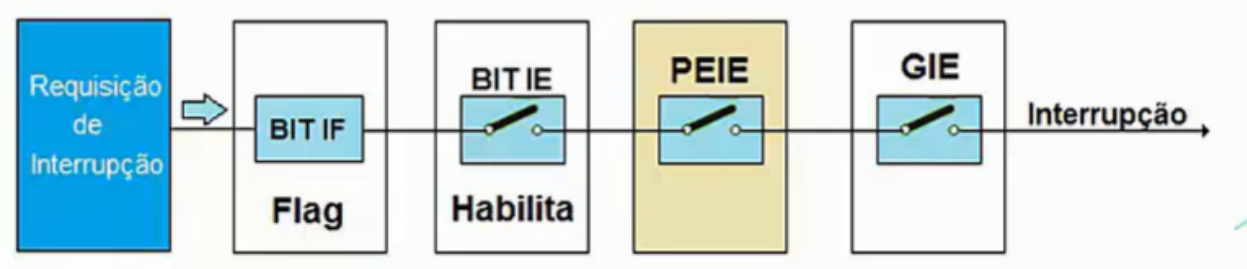
As interrupções do tipo INT são interrupções externas. O PIC18F45k22 possui três pinos (INT0, INT1 INT2) dedicados à essas interrupções externas que pode ser pulso de um sensor externo, acionamento, etc. Quando qualquer um desses pinos são colocados em nível lógico alto, ocorre a requisição da interrupção.

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	

Este tipo de interrupção não possui nível de prioridade. Ela está sempre em alta prioridade.

Sequência de acionamento da interrupção

- A requisição chega;
- O bit IF é automaticamente setado a 1;
- O bit IE funciona como uma chave neste ponto, a interrupção só ocorre se ele for habilitado;
- O PEIE funciona como uma chave neste ponto, a interrupção só ocorre se ele for habilitado;
- O GIE funciona como uma chave neste ponto, a interrupção só ocorre se ele for habilitado;
- A interrupção ocorre



Bits de controle

Todos os periféricos do MCU que geram interrupções possuem estes três flags. Sempre que você ver estes termos, é porque refere-se a interrupção. Eles devem ser configurados via software:

IF → Flag de interrupção: Evento que ocorreu.

IP → Prioridade de interrupção: Bit de maior prioridade.

IE → Habilitação da Interrupção.

O INT0 só não possui o IP, pois, como dito, não possui nível de interrupção. Mas possui o bit INTEDG0 que atua na forma que o pulso será lido.

Com estas informações, vamos configurar a interrupção externa:

```
RCONbits.IPEN = 1;

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

INTCONbits.INT0IE = 1;
INTCONbits.INT0IF = 0;
INTCON2bits.INTEDG0 = 1;
```

Função interrupção

Agora, precisamos configurar a função de interrupção.

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

Com isso a interrupção já está configurada. Podemos adicionar isso em um algoritmo para visualizar este processo

```
#include"../Bibliotecas/config.h"
#include "../Bibliotecas/LCD4bits.h"

#define _XTAL_FREQ 8000000

void __interrupt(high_priority) INT0_INT(void){
    lcdClean();
    lcdString("Interrompi");
    __delay_ms(2000);
    INTCONbits.INT0IF = 0;
}

void main(void) {
    ANSELD = 0;
    TRISD = 0;

    lcdInit();

    RCONbits.IPEN = 1;

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

    INTCONbits.INT0IE = 1;
    INTCONbits.INT0IF = 0;
```

```
    INTCON2bits.INTEDG0 = 1;

    while(1){
        lcdSetCursor(1, 1);
        lcdString("Rotina normal...");
        __delay_ms(500);
    }
}
```

É importante limpar o flag IF manualmente. Mesmo com a interrupção setada, esse flag não é limpo, fazendo com que o programa fique preso na interrupção.

Outro exemplo

A função interrupção é uma função como qualquer outra, você pode criar a rotina que quiser com ela

```
void __interrupt(high_priority) INT0_INT(void){
    static int incrementa = 0;
    char screenBuff[3];

    incrementa++;
    sprintf(screenBuff, "%03d", incrementa);

    lcdClean();
    lcdString("INT = ");
    lcdString(screenBuff);

    __delay_ms(2000);
    lcdClean();
    INTCONbits.INT0IF = 0;
}
```

Prioridade de interrupções

Utilizando funções tanto de alta quanto de baixa prioridade. A interrupção configurada como alta prioridade pode ser executada a qualquer momento, enquanto a baixa prioridade só é executada quando não houver uma alta prioridade sendo executada.

```
#include "../Bibliotecas/LCD4bits.h"
#include "../Bibliotecas/config.h"
#include <stdio.h>

#define _XTAL_FREQ 8000000
```

```
void __interrupt(high_priority) INT0_interrupt(void){
    PORTDbits.RD7 = 1;
    __delay_ms(5000);
    PORTDbits.RD7 = 0;
    INTCONbits.INT0IF = 0;
}

void __interrupt(low_priority) INT1_interrupt(void){
    lcdClean();
    char x = 0;

    lcdComando(1, 'I');
    __delay_ms(1000);
    lcdComando(1, 'N');
    __delay_ms(1000);
    lcdComando(1, 'T');
    __delay_ms(1000);
    lcdComando(1, '1');
    __delay_ms(1000);
    lcdClean();
    INTCON3bits.INT1IF = 0;
}

void main(void){

    ANSELD = 0;
    lcdInit();

    RCONbits.IPEN = 1;

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

    INTCONbits.INT0IE = 1;
    INTCON2bits.INTEDG0 = 1;

    INTCON3bits.INT1E = 1;
    INTCON2bits.INTEDG1 = 1;
    INTCON3bits.INT1IP = 0;


    while(1){
        lcdSetCursor(1, 1);
        lcdString("Ola mundo");
        __delay_ms(500);
    }
}
```

Interrupções de mesma prioridade

Você pode trabalhar com níveis de prioridade iguais para interrupções diferentes. Para diferenciá-las, utilize seu flag (IF).

```
void __interrupt(high_priority) INT0_interrupt(void){

    // INT0
    if(INTCONbits.INT0IF){
        PORTDbits.RD7 = 1;
        __delay_ms(5000);
        PORTDbits.RD7 = 0;
        INTCONbits.INT0IF = 0;
    }

    // INT1
    if(INTCON3bits.INT1IF){
        lcdClean();
        lcdComando(1, 'I');
        __delay_ms(1000);
        lcdComando(1, 'N');
        __delay_ms(1000);
        lcdComando(1, 'T');
        __delay_ms(1000);
        lcdComando(1, '1');
        __delay_ms(1000);
        lcdClean();
        INTCON3bits.INT1IF = 0;
    }
}
```