Prática 3:

Interrupções e timers

3.1 – Introdução e objetivos

Na prática anterior foi visto que a função main fica continuamente monitorando o teclado avaliando se alguma tecla foi pressionada através da leitura contínua dos pinos do PIC que são ligados ao teclado. Esta estratégia tem um grande problema: o PIC fica "ocupado" com esta tarefa de monitoração e assim não pode fazer outras tarefas. Ter um microcontrolador dedicado a esta única atividade não é algo viável e representa um verdadeiro desperdício de recursos.

Para resolver esta questão é possível usar os importantes recursos de interrupção e timers. De modo simplificado, uma interrupção é um desvio da execução das instruções de um programa ocasionados por um evento. De modo bastante ilustrativo, considere que agora o código da aplicação anterior (leitura do teclado matricial) seja reescrito e dividido em três partes confirme ilustra a Figura 4.1. Uma das partes continua sendo a função principal main que é continuamente executada para realizar as atividades de uma determinada aplicação qualquer. Uma segunda parte, aqui chamada de "rotina de timer" deverá conter uma função para - periodicamente - varrer as colunas de um teclado. A terceira parte, aqui chamada de "rotina de interrupção", é executada toda vez que uma tecla é pressionada e identifica qual tecla foi pressionada. Esta última só é chamada quando uma tecla é pressionada.

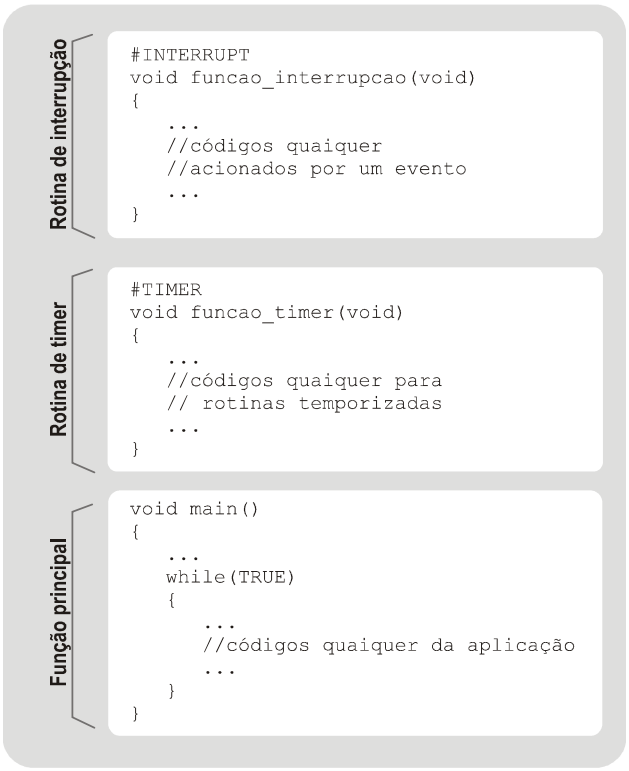

Figura 4.1 - Esquema de divisão de funções de um programa que trabalha com timers e interrupções.

As rotinas de interrupção e timer são especiais e por isto geralmente são precedidas por um código que começa com # indicando ao compilador que a rotina subsequente deve ser tratada como uma rotina de timer ou interrupção e por isto tem um tratamento especial.

No exemplo hipotético da leitura de teclas de um teclado matricial, imagine que a função main não execute mais nenhuma função para leitura do teclado. Assim, ela está disponível para execução de outras atividades. Assim que o programa é iniciado (ou quer dizer, o microcontrolador é eletricamente ligado), os códigos dentro de main são executados. Contudo, o compilador executa periodicamente a função de timer em um período de tempo configurável e fixo. Nesta função pode ser inserido os códigos que varrem as linhas do teclado. Considere, por exemplo, que o timer é programado para executar a cada 200ms. Isto significa que a cada 200ms, o processador pará automaticamente de executar as rotinas da main e executa os códigos contidos na rotina timer. Assim, a cada 200ms o código da função main é interrompido para a varredura de uma linha do teclado.

Já a rotina de interrupção é executada toda vez que acontece um determinado evento. Neste caso, a interrupção poderia ser configurada para ser executada toda vez que um novo valor é enviado aos pinos de entrada do PIC. Isto quer dizer que apenas quando um tecla for pressionada (enviando a um dos pinos de leitura do PIC um nível lógico alto) a rotina de verificação de qual tecla foi pressionada é executada. Quando nenhuma tecla é pressionada, esta rotina não é executada deixando os códigos contidos em main serem executados. Deste modo, neste exemplo, as colunas são varridas com uma certa periodicidade enquanto que em outros momentos o PIC está livre para processar os códigos contidos em main. Ainda, a rotina de detecção de tecla pressionada só é executada quando uma tecla é pressionada pois ela é ativada pela mudança de um nível lógico em alguns dos pinos de leitura do PIC. As próxima subseções tratam com mais detalhes o uso de interrupções e timers. Após isto é sugerido uma aplicação que emprega dois diferentes tipos de timers simultaneamente e duas diferentes interrupções que são acionadas pelo pressionamento de botões.

3.2 – Teoria de interrupções

Para trabalhar com interrupções é preciso entender os registradores associados a elas já que é através da leitura e gravação (R/W) das flags destes registradores que podermos manipular estes recursos. Desde já é fundamental entender que cada interrupção tem pelo menos três bits (ou flags) de configuração:

i. um bit que é setado (pelo próprio hardware do microcontrolador) toda vez que o evento associado à interrupção acontece. Assim, deve-se monitorar constantemente este bit para saber se alguma interrupção aconteceu pois ele indica a ocorrência de uma interrupção que é avisada através do set desta flag;
ii. um bit de habilitação que deve ser configurado pelo programador pra indicar se a interrupção deve estar ativa ou não e;
iii. um bit de prioridade para indicar se a interrupção deve ser tratada como de baixa ou alta prioridade. Interrupções de baixa prioridade não são executadas enquanto interrupções de alta prioridade estão em progresso. A flag IPEN (*interrupt priority enable*) contida no sétimo bit do registrador RCON habilita (IPEN=1) o tratamento das interrupções considerando seus níveis de prioridade ou desabilita (IPEN=0) as prioridades tratando todas como de igual relevância.

É importante também destacar que o sistema de interrupção possui um bit de ativação global que quando ligado, permite que as interrupções específicas programadas possam ser executadas. Se o bit de interrupção global estiver desabilitado, ainda que os bits de configurações das interrupções estejam corretamente programadas, nenhuma interrupção deve acontecer já que o bit de habilitação global inibe o funcionamento de qualquer interrupção. Para entender melhor este processo, são descritos alguns importantes registradores de controle de interrupções na sequência. O exemplo de aplicação ajudara o leitor a entender de forma prática como este processo acontece.

Tabela 4.1 - Descrição do registrador **INTCON** (endereço 0xFF2).

bit 7 (R/W)	*Bit 6 (R/W)*	*bit 5 (R/W)*	*bit 4 (R/W)*	*bit 3 (R/W)*	*bit 2 (R/W)*	*bit 1 (R/W)*	*bit 0 (R/W)*
GIE/GIEH	**PEIE/GIEL**	**TMR0IE**	**INT0IE**	**RBIE**	**TMR0IF**	**INT0IF**	**RBIF**

- **GIE/GIEH:** bit de interrupção global. Se 1 permite a execução das interrupções habilitadas pelo usuário. Se 0 desabilita todas as interrupções.
- **PEIE/GIEL:** bit de interrupção global para as interrupções acionadas pelos periféricos de hardware do PIC. Se 1 permite a execução das interrupções de hardware habilitadas pelo usuário. Se 0 desabilita todas as interrupções.
- **TMR0IE**: habilita o uso da interrupção de timer 0 (TMR0).
- **INT0IE**: habilita ou desabilita a função de interrupção externa 0 (também chamada de INT0) do PIC. A interrupção externa é acionada quando um determinado pino do PIC tem seu valor de tensão alterado por borda acionando assim esta interrupção. No 18F4550 este pinos são o 33, 34 e 35 (INT0, INT1 e INT2, respectivamente).
- **RBIE**: assim como acontece nas interrupções externas INT0, INT1 e INT2, a porta B (pinos 4 a 7) pode também ser configurada para ser sensível a uma mudança de nível em seus pinos. Se RBIE=1 esta função é habilitada.
- **TMR0IF**: esta flag é setada pelo próprio hardware do PIC após um determinado período de tempo fixo. Assim, toda vez que este tempo é alcançado, esta flag é setada indicando que acaba de ocorrer um evento de interrupção de timer 0. Note que a flag TMR0IE deve ser habilitada para que TMR0IF possa indicar um estouro de timer. Quando TMR0IF=0 indica que este tempo ainda não foi alcançado. As funções de timer serão descritas adiante neste mesmo capítulo.
- **INT0IF:** esta flag é setada pelo próprio hardware do PIC quando acontece uma mudança de nível lógico no pino externo (pino 33 do 18F4550) do PIC indicando que aconteceu uma interrupção de borda em um pino do PIC. Depois de processada a interrupção, o programador deve limpar esta flag para que possa ser detectada uma nova interrupção. Quando INT0IF=0 indica que nenhuma interrupção externa aconteceu. Esta flag só é alterada quando a interrupção externa é habilitada fazendo-se INT0IE=1.
- **RBIF:** ele é setado se algum dos pinos (RB7 a RB4) do PIC tem recebido um dado novo. Isto só acontece se esta interrupção é habilitada fazendo-se RBIE=1. Se RBIF=0 indica que não há nenhuma alteração no nível lógico de nenhum dos pinos RB7 a RB4.

Existem ainda os registradores INTCON2 e INTCON3 que, a exemplo do INTCON mostrado na Tabela 4.1, configuram o timer 1 e 2 além das interrupções externas 1 e 2. Destaque deve ser dado ao INTCON2 que configura algumas importantes ações de algumas interrupções. Este registrador é mostrado na Tabela 4.2. Os demais registradores citados podem ser consultados no datasheet do 18F4550.

Tabela 4.2 - Descrição resumida do registrador **INTCON2** (endereço 0xFF1).

bit 7 (R/W)	*Bit 6 (R/W)*	*bit 5 (R/W)*	*bit 4 (R/W)*	*bit 3*	*bit 2 (R/W)*	*bit 1*	*bit 0 (R/W)*
RBPU'	**INTEDG0**	**INTEDG1**	**INTEDG2**	**-**	**TMR0IP**	**-**	**RBIP**

- **RBPU:** bit para ativar ou desativar as resistências pull-up que a porta B possui.
- **INTEDG0**: determina que tipo de borda deve ativar a interrupção externa 0. Se INTEDG0=1 uma borda de subida no pino 33 irá acionar a interrupção externa fazendo com que a flag INT0IF do registrador INTCON seja setada (INT0IF=1) desde que esta interrupção tenha sido ativada fazendo-se INT0IE=1. Se INTEDG0=0, uma borda de descida deve acionar esta interrupção.
- **INTEDG1 e INTEDG2**: tem significado similar a INTEDG0. Contudo, tratam das interrupções externas 1 e 2 (INT1 e INT2).
- **TMR0IP**: determina se o timer 0 terá alta (TMR0IP=1) ou baixa (TMR0IP=0) prioridade.
- **RBIP**: determina se a interrupção ocasionada por mudança de borda nos pinos RB7 a RB4 terá alta (RBIP=1) ou baixa (RBIP =0) prioridade.

Os registradores não se limitam apenas ao INTCON, INTCON2 e INTCON3. Existem ainda outros importantes registradores tais como o PIR1 e PIR2 (*peripheral interrupt request* - registradores que indicam a requisição de uma determinada interrupção chamada por um periférico), PIE1 e PIE2 (*peripheral interrupt enable* - registradores que habilitam ou desabilitam as prioridades associadas aos periféricos), e, por fim, IPR1, IPR2 (*interrupt priority register* - registradores que determinam a prioridade de alguns interrupções acionadas por periféricos). Neste caso, os periféricos mais comuns são o conversor analógico-digital (interrupção indicando uma conversão finalizada), unidades USART e EUSART para transmissão de dados serial (indicando a recepção de um novo dado no buffer de recepção ou o fim do envio de um dado), USB (interrupção de requisição de conexão USB com novo dispositivo), memória EEPROM (interrupção indicando que uma operação de escrita na memória está pronta) dentre outras opções de periféricos. Todos estes registradores e suas flags são detalhadamente descritos no datasheet do 18F4550.

3.3 – Teoria de timers

O timer é um tipo especial de interrupção que é acionado por tempo. Em termos mais específicos, trata-se de um registrador contador que recebe um sinal de clock e conta até certo valor. Uma vez alcançado este valor, ele causa o estouro de timer gerando uma interrupção. O tempo pode ser controlado pelo clock ou pelo valor de contagem. Por exemplo, o timer 0 do 18F4550 é de 8 bits por padrão (mas ele pode ser configurado para também trabalhar como sendo 16 bits) e pode ser alimentado por um clock externo ou pelo próprio clock que alimenta internamente a CPU do microcontrolador. Este clock pode ser dividido em várias vezes através de um divisor de clock chamado de prescaler. Assim, o timer 0 no modo padrão irá contar de 0 (8 bits) até 255 a cada intervalo de clock gerado para o contador do timer 0. Quando ele atingir o valor 0, irá estourar gerando a interrupção.

Para controlar o tempo, deve ser configurado o prescaler gerando um sinal de clock adequado ou o valor de contagem inicial deve ser modificado de 0, que é seu valor de inicialização padrão, para qualquer outro valor que o usuário deseja. Por exemplo, imagine que o timer 0 é alimentado pelo sinal da CPU que é de 20MHz/4=5MHz. Considere ainda que foi configurado um preescaler na proporção de 1:64 o que quer dizer que o clock que alimenta a contagem de timer 0 será 5MHz/64=78,125Khz ou 12,8 us. Assim, se ele contar de 0 até 255 (256x12,8us) o timer irá estourar a cada 3,2ms. Se desejar que o timer estoure a cada 2ms, a contagem deve ir de 0 até 155 (gerando 156 x 12,8us = 2ms). Para ilustrar estas configurações a Tabela 4.3 exibe o registrador T0CON

Tabela 4.3 - Descrição do registrador **TOCON** (endereço 0xFD5) que controla o timer 0 do 18F4550.

bit 7 (R/W)	*Bit 6 (R/W)*	*bit 5 (R/W)*	*bit 4 (R/W)*	*bit 3 (R/W)*	*bit 2 (R/W)*	*bit 1 (R/W)*	*bit 0 (R/W)*
TMR0ON	**T08BIT**	**T0CS**	**T0SE**	**PSA**	**T0PS**		

- **TMR0ON:** habilita o início da contagem do timer 0 quando TMR0ON=1 ou se TMR0ON=0 pára a contagem do timer 0.
- **T08BIT:** se T08BIT=1 o timer 0 é configurado como um contador de 8bits contando de 0 até 255. Caso contrário, é um contador de 16 bits contando de 0 até 65.535.
- **T0CS:** seleciona a fonte de clock que deve ser usada para decrementar a contagem do contador do timer 0. Se T0CS=1 é usado o clock inserido no pino TOCKI. Se TOCS=0 é usado o mesmo clock que alimenta a CPU do PIC.
- **T0SE:** indica que tipo de transição (borda de subida ou descida) deve ser usada em consideração para decremento quando se usa um clock externo no pino T0CKI para alimentar o timer.
- **PSA:** liga (PSA=0) ou desliga (PSA=1) o preescaler para dividir o clock que irá alimentar o contador do timer 0.
- **TOPS:** estes 3 bits selecionam o quanto o clock deve ser divido antes de chegar ao contador do timer zero segundo a relação: 111= :256; 110=1:128; 101=1:64; 100=1:32; 011=1:16; 010=1:8; 001=1:4 e 000=1:2.

O valor de contagem do timer 0 é armazenado no registrador de 8 bits TMR0L (quando a contagem vai de 0 a 255) ou nos registradores TMR0H e TMR0L (quando a contagem vai de 0 até 65.535). Estes registradores estão nos endereços 0xFD7 e 0xFD6, respectivamente. Quando a contagem é estourada (chega a zero), a flag TMR0IF é setada indicando o estouro de timer 0 e gerando a interrupção. Com isto, a CPU do PIC deixa de processar o que estava processando e passa a processar a rotina que foi indicada para tratar a interrupção de timer 0. Depois que a rotina de interrupção de timer 0 é finalizada, a CPU volta para o ponto onde parou antes do timer estourar.

No caso do 18F4550, este possui outros timers. O timer 1 tem 16 bits, um timer 2 de 8bits e um timer 3 de 16 bits. O funcionamento destes timers é similar ao do timer 0 tendo cada um deles seus próprios registradores.

3.4 – Exemplo de aplicação

Para ilustrar a implementação de algumas interrupções e timer, considere um circuito com um LED que pisca com uma certa frequência. Um timer deve ser acionado para piscar este LED e a frequência pode variar (aumentar ou diminuir) segundo a seleção de um botão para aumentar a frequência em que o LED pisca e um outro para diminuir a frequência.

Duas interrupções externas estarão ligadas aos botões de incrementar ou decrementar a frequência com que o LED pisca e toda vez que um botão for pressionado, acionando assim uma interrupção, um segundo LED irá piscar para demonstrar a execução das rotinas de interrupção externas. Por fim, um terceiro LED também irá piscar continuamente e com o atraso de 4 segundo na função main. Isto mostrará que mesmo nos instantes de tempo quando a função main estiver em parada por 4 segundo, as rotinas de timer e interrupção poderão ser executadas demonstrando a independência destas funções em relação ao conteúdo executado em main.

3.4.1 – O circuito

A Figura 4.2 ilustra o circuito que deve ser montado para a aplicação deste capítulo. Note que nele temos 2 chaves com circuito que evita a trepidação de chave. e três LEDs conforme descrito anteriormente.

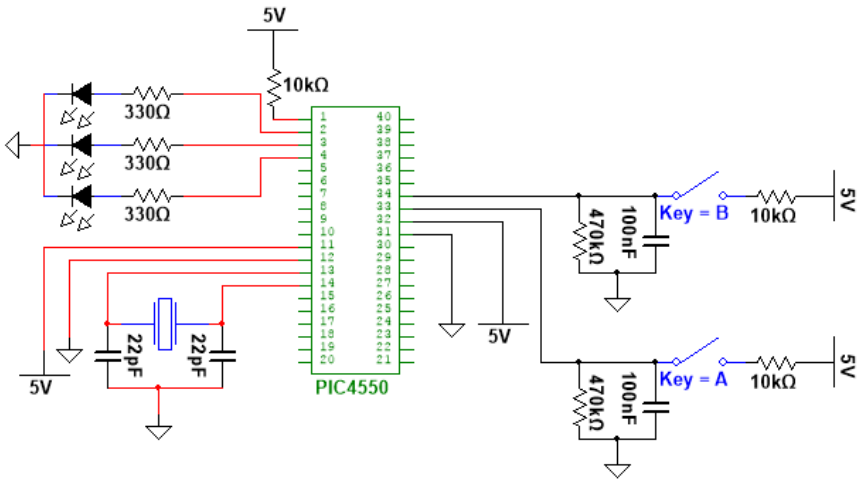

Figura 4.2 - Circuito da aplicação.

3.4.2 – O código

O Código 4.1 ilustra o comportamento da aplicação. Antes, no Wizard do CCS 5, proceda com as etapas:

1) **Seção "Device"**: selecione o PIC 18F4550 e defina um clock de 20MHz;
2) **Seção "Interrupts"**: selecione as opções "Timer 0 overflow (using TIMER0 name)", "Timer 1 overflow", "External interrupt" e, por fim, "External interrupt #1";
3) **Seção "Timers"**: habilite a função "Enable" de Timer 0 (RTCC). Deixe a opção "Source" definida como "Internal" assim como "Bit count" definido como 16 bits. Por fim, no campo "Resolution", escolha a opção 12,8us. Isto gerará um estouro (overflow) a cada 840ms
4) **Seção "I/O Pins"**: selecione os pinos A0, A1 e A2 como "Output" e os pinos B0 e B1 como "Input". Feito isto, finalize o Wizard clicando em "Creat Project".

Código 4.1 - Código gerado automaticamente pelo Wizard do CCS 5 no arquivo ".c" do projeto.

```
1   #include <main.h>
2
3   #INT_TIMER0
4   void  TIMER0_isr(void)
5   {
6   }
7
8   #INT_EXT
9   void  EXT_isr(void)
10  {
11  }
12
13  #INT_EXT1
14  void  EXT1_isr(void)
15  {
16  }
17
18  void main()
19  {
20     setup_timer_0(RTCC_INTERNAL|RTCC_DIV_64);      //840 ms overflow
```

```
21      enable_interrupts(INT_TIMER0);
22      enable_interrupts(INT_EXT);
23      enable_interrupts(INT_EXT1);
24      enable_interrupts(GLOBAL);
25      while(TRUE)
26      {
27         //TODO: User Code
28      }
29   }
```

Nas linhas 3 a 6, 8 a 11 e 13 a 16 são criadas as funções de tratamento das interrupções de timer 0, interrupção externa 0 e interrupção externa 1 respectivamente. Note a presença do # para indicar ao compilador a que tipo de interrupção estas funções estão vinculadas. Todo o código das interrupções deve ser inserido dentro destas rotinas criadas.

Na função main alguns configurações são feitas (estas funções habilitam e desabilitam alguns dos valores das flags descritas nas seções 4.2 e 4.3 automaticamente). A função setup_timer_0 (RTCC_INTERNAL|RTCC_DIV_64) habilita o timer 0 (que algumas vezes é chamado de RTCC) indicando que deve ser usado o clock interno da CPU para alimentar a contagem do timer 0. Ainda, o clock do timer 0 deve ser dividido por 64. Logo, se temos um cristal de 20Mhz que é dividido por 4 (na família 18F sempre o clock é dividido de 4 antes de chegar a CPU) gerando 5MHz. Estes 5MHz são divididos por 64 pelo preescaler gerando uma frequência de 78125Hz ou um período de 12,8us. Como o timer 0 foi registrado para trabalhar em 16 bits (2^{16}=65.536 contagens), teremos ao decorrer das 65.536 contagens um tempo total de 0,838segundos ou, aproximadamente, 840ms.

Preencha as funções criadas conforme demonstra o Código 4.2. Nas linhas 2 e 3 criaram-se duas variáveis globais. A primeira delas mos ajudará a controlar o tempo com que o timer 0 estoura. A segunda contará quantas vezes o timer 0 já estourou.

Código 4.2 - Código da aplicação proposta.

```
1    #include <main.h>
2    int16 controla_tempo;
3    int8 num_estouros;
4
5    #INT_TIMER0
6    void  TIMER0_isr(void)
7    {
8       disable_interrupts(GLOBAL);
9       set_timer0(controla_tempo);
10      num_estouros++;
11      if(num_estouros >= 5)
12      {
13         output_high(PIN_A0);
14         delay_ms(200);
15         num_estouros = 0;
16         output_low(PIN_A0);
17      }
18      enable_interrupts(GLOBAL);
19   }
20
21   #INT_EXT
22   void  EXT_isr(void) //ira incrementar a variavel que controla contagem timer
23   {
24      int1 dado;
25      output_high(PIN_A1);
26      dado = input(PIN_B1);
27      if(dado)
28         if (controla_tempo < 55000) //se não esta do limite do incremento
29            controla_tempo = controla_tempo + 5000;
30      output_low(PIN_A1);
31   }
32
33   #INT_EXT1
34   void  EXT1_isr(void)//ira decrementar a variavel que controla contagem timer
35   {
```

```
36      int1 dado;
37      output_high(PIN_A1);
38      dado = input(PIN_B0);
39      if(dado)
40         if (controla_tempo >= 5000) //se não esta do limite do decremento
41            controla_tempo = controla_tempo - 5000;
42      output_low(PIN_A1);
43   }
44
45   void main()
46   {
47      ... // (foram omitidos alguns códigos de inicialização)
48      controla_tempo = 0; //inicializa variavel
49      num_estouros = 0; //inicializa variavel
50      while(TRUE)
51      {
52         output_high(PIN_A2);
53         delay_ms(300);
54         output_low(PIN_A2);
55         delay_ms(4000);
56      }
57   }
```

Na interrupção de timer (linhas 5 a 19) começamos desabilitando todas as interrupções (linha 8) até que o timer finaliza. Isto é feito desabilitando-se as flgs GIE/GIEH e GIEL do registrador INTCON descrito na Tabela 4.1. Assim, enquanto não fosse processados os códigos da rotina de timer 0, nenhuma outra interrupções era chamada até que a última linha de código de timer 0 fosse executada. Esta linha é justamente a habilitação global das interrupções (linha 18). Isto acontece pois enquanto a rotina de timer é processada espera-se que nada interfira no seu processamento até que esta finalize. Na linha 9 o registrador de contagem do timer 0 é iniciado pela variável "controla_tempo". Logo, controlando o valor desta variável pode-se aumentar ou diminuir o valor de estouro do timer 0 que contará não mais de 0 a 65.535 e sim de "controla_tempo" até 65.535. Na linha 10 é contado quantas vezes o timer 0 já estourou e quando temos 5 estouros (linha 11) piscamos o LED ligado a A0 e reiniciamos a contagem de estouros. Assim, se controla_tempo=0, por exemplo, o timer estoura a cada 840ms. Contudo, o LED só ira piscar a cada 5x840ms=4,2seg. Se controla_tempo=20.000, por exemplo, o timer estoura a cada ((65536-20000)/65536)*840=583ms e o LED só ira piscar a cada 5x583ms=2,9seg.

As rotinas de interrupções externas (linhas 21 a 31 e 33 a 43) só são chamadas quando um dos botões é pressionado fazendo assim executar a respectiva rotina ligada a esta interrupção. Os códigos de compilação das linhas 21 e 33 indicam ao compilador que as rotinas abaixo estão associadas as interrupções externas 0 e 1 (INT_EXT e INT_EXT1, respectivamente). Como foi feito na rotina de timer 0, poderia-se também aqui optar por desabilitar todas as outras interrupções (linhas 8 e 18) enquanto esta interrupção externa é processada. Isto evitaria um "encavalamento" de interrupções. Contudo, aqui negligenciamos este processo. Na rotina de interrupção lemos o valor que gerou o bit de interrupção (linhas 26 e 38) e se ele estiver em nível alto, indicando que o botão foi pressionado (linhas 27 e 39), incrementamos ou decrementamos em 5.000 unidades a variável "controla_tempo" (linhas 29 e 41) que inicializará o valor do timer toda vez que ele foi reiniciado (ver linha 9). Contudo, antes, é necessário verificar se esta variável pode ou não ser incrementada ou decrementada (linhas 28 e 40). Um LED ligado na porta A1 pisca toda vez que um botão for pressionado para indicar que a interrupção está sendo executada.

Por fim, nas linhas 50 a 56, a função main pode se ocupar em qualquer atividade. Neste caso, sua tarefa é única e exclusivamente fazer um LED ligado na porta A2 piscar a cada 4 segundos.