CAPÍTULO 2

MICROCONTROLADOR PIC16F84

1 - Introdução

O **PIC 16F84** pertence a uma classe de microcontroladores de 8 bits, com uma arquitectura RISC. A estrutura genérica é a do mapa que se segue, que nos mostra os seus blocos básicos.

Memória de programa (FLASH) - para armazenar o programa que se escreveu.

Como a memória fabricada com tecnologia FLASH pode ser programa e limpa mais que uma vez. ela torna-se adequada para o desenvolvimento de dispositivos.

EEPROM - memória dos dados que necessitam de ser salvaguardados quaando a alimentação é desligada. Normalmente é usada para guardar dados importantes que não se podem perder quando a alimentação, de repente, "vai abaixo". Um exemplo deste tipo de dados é a temperatura fixada para os reguladores de temperatura. Se, durante uma quebra de alimentação, se perdessem dados, nós precisaríamos de proceder a um novo ajustamento quando a alimentação fosse restabelecida. Assim, o nosso dispositivo, perderia eficácia.

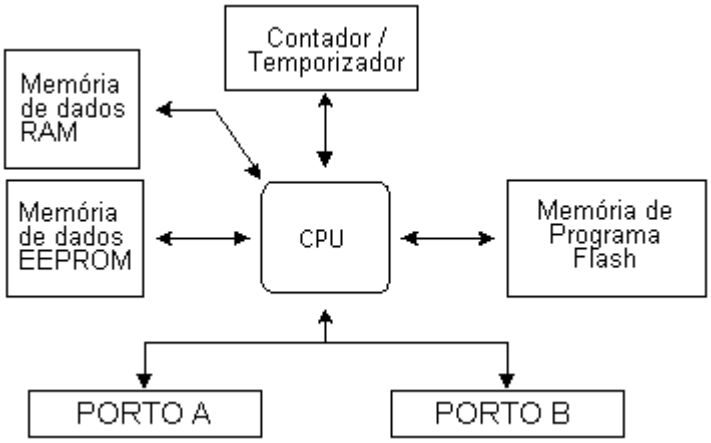
RAM - memória de dados usada por um programa, durante a sua execução.

Na RAM, são guardados todos os resultados intermédios ou dados temporários durante a execução do programa e que não são cruciais para o dispositivo, depois de ocorrer uma falha na alimentação.

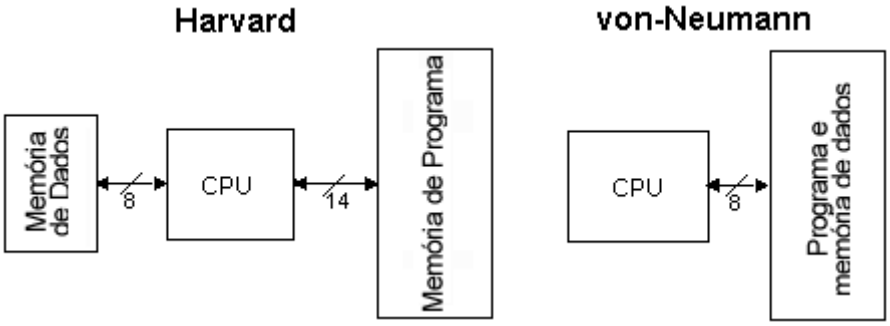
PORTO A e PORTO B são ligações fisicas entre o microcontrolador e o mundo exterior. O porto A tem cinco pinos e o porto B oito pinos.

CONTADOR/TEMPORIZADOR é um registo de 8 bits no interior do microcontrolador que trabalha independentemente do programa. No fim de cada conjunto de quatro ciclos de relógio do oscilador, ele incrementa o valor armazenado, até atingir o valor máximo (255), nesta altura recomeça a contagem a partir de zero. Como nós sabemos o tempo exacto entre dois incrementos sucessivos do conteúdo do temporizador, podemos utilizar este para medir intervalos de tempo, o que o torna muito útil em vários dispositivos.

UNIDADE DE PROCESSAMENTO CENTRAL faz a conexão com todos os outros blocos do microcontrolador. Ele coordena o trabalho dos outros blocos e executa o programa do utilizador.



Esquema do microcontrolador PIC16F84



Arquitecturas Harvard versus von Neumann

2 - CISC, RISC

Já foi dito que o PIC16F84 tem uma arquitectura RISC. Este termo é encontrado, muitas vezes, na literatura sobre computadores e necessita de ser explicada aqui, mais detalhadamente. A arquitectura de Harvard é um conceito mais recente que a de von-Neumann. Ela adveio da necessidade de pôr o microcontrolador a trabalhar mais rapidamente. Na arquitectura de Harvard, a memória de dados está separada da memória de programa. Assim, é possível uma maior fluência de dados através da unidade central de processamento e, claro, uma maior velocidade de funcionamento. A separação da memória de dados da memória de programa, faz com que as instruções possam ser representadas por palavras de mais que 8 bits. O PIC16F84, usa 14 bits para cada instrução, o que permite que que todas as instruções ocupem uma só palavra de instrução. É também típico da arquitectura Harvard ter um reportório com menos instruções que a de von-Neumann's, instruções essas, geralmente executadas apenas num único ciclo de relógio.

Os microcontroladores com a arquitectura Harvard, são também designados por "microcontroladores RISC". RISC provém de Computador com um Conjunto Reduzido de Instruções (Reduced Instruction Set Computer). Os microcontroladores com uma arquitectura von-Neumann são designados por 'microcontroladores CISC'. O nome CISC deriva de Computador com um Conjunto Complexo de Instruções (Complex Instruction Set Computer).

Como o PIC16F84 é um microcontrolador RISC, disso resulta que possui um número reduzido de instruções, mais precisamente 35 (por exemplo, os microcontroladores da Intel e da Motorola têm mais de cem instruções). Todas estas instruções são executadas num único ciclo, excepto no caso de instruções de salto e de ramificação. De acordo com o que o seu fabricante refere, o PIC16F84 geralmente atinge resultados de 2 para 1 na compressão de código e 4 para 1 na velocidade, em relação aos outros microcontroladores de 8 bits da sua classe.

3 - Aplicações

O PIC16F84, é perfeitamente adequado para muitas variedades de aplicações, como a indústria automóvel, sensores remotos, fechaduras eléctricas e dispositivos de segurança. É também um dispositivo ideal para cartões inteligentes, bem como para dispositivos alimentados por baterias, por causa do seu baixo consumo.

A memória EEPROM, faz com que se torne mais fácil usar microcontroladores em dispositivos onde o armazenamento permanente de vários parâmetros, seja necessário (códigos para transmissores, velocidade de um motor, frequências de recepção, etc.). O baixo custo, baixo consumo, facilidade de manuseamento e flexibilidade fazem com que o PIC16F84 se possa utilizar em áreas em que os microcontroladores não eram anteriormente empregues (exemplo: funções de temporização, substituição de interfaces em sistemas de grande porte, aplicações de coprocessamento, etc.).

A possibilidade deste chip de ser programável no sistema (usando somente dois pinos para a transferência de dados), dão flexibilidade do produto, mesmo depois de a sua montagem e teste estarem completos.

Esta capacidade, pode ser usada para criar linhas de produção e montagem, para armazenar dados de calibragem disponíveis apenas quando se proceder ao teste final ou, ainda, para aperfeiçoar os programas presentes em produtos acabados.

4 - Relógio / ciclo de instrução

O relógio (clock), é quem dá o sinal de partida para o microcontrolador e é obtido a partir de um componente externo chamado "oscilador". Se considerasse-mos que um microcontrolador era um relógio de sala, o nosso clock corresponderia ao pêndulo e emitiria um ruído correspondente ao deslocar do pêndulo. Também, a força usada para dar corda ao relógio, podia comparar-se à alimentação eléctrica.

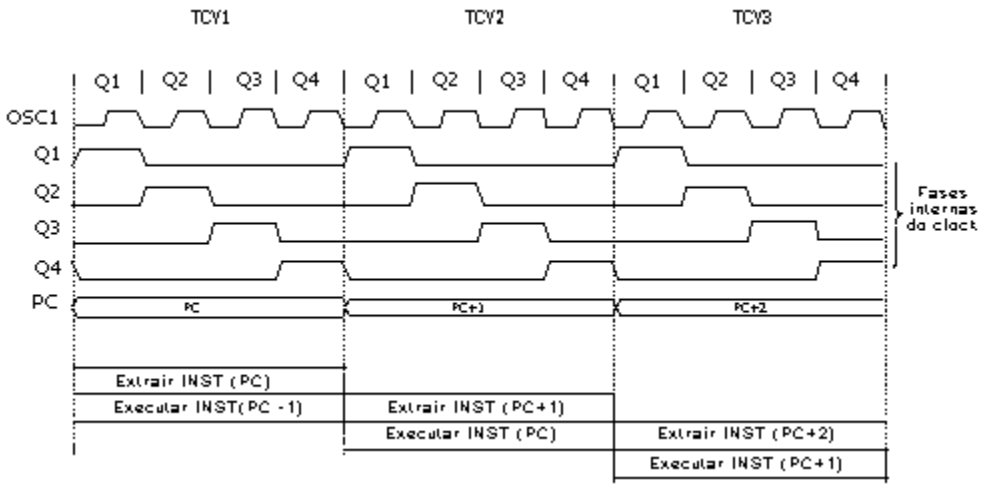
O clock do oscilador, é ligado ao microcontrolador através do pino OSC1, aqui, o circuito interno do microcontrolador divide o sinal de clock em quatro fases, Q1, Q2, Q3 e Q4 que

não se sobrepõem. Estas quatro pulsações perfazem um ciclo de instrução (também chamado ciclo de máquina) e durante o qual uma instrução é executada.

A execução de uma instrução, é antecedida pela extracção da instrução que está na linha seguinte. O código da instrução é extraído da memória de programa em Q1 e é escrito no registo de instrução em Q4.

A descodificação e execução dessa mesma instrução, faz-se entre as fases Q1 e Q4 seguintes. No diagrama em baixo, podemos observar a relação entre o ciclo de instrução e o clock do oscilador (OSC1) assim como as fases Q1-Q4.

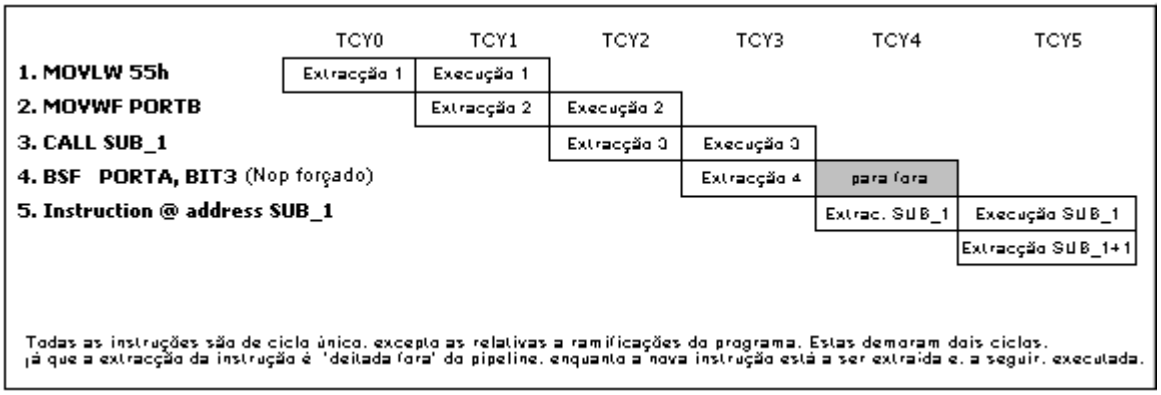
O contador de programa (Program Counter ou PC) guarda o endereço da próxima instrução a ser executada.



5 – Pipelining

Cada ciclo de instrução inclui as fases Q1, Q2, Q3 e Q4. A extracção do código de uma instrução da memória de programa, é feita num ciclo de instrução, enquanto que a sua descodificação e execução, são feitos no ciclo de instrução seguinte. Contudo, devido à sobreposição – pipelining (o microcontrolador ao mesmo tempo que executa uma instrução extrai simultaneamente da memória o código da instrução seguinte), podemos considerar que, para efeitos práticos, cada instrução demora um ciclo de instrução a ser executada. No entanto, se a instrução provocar uma mudança no conteúdo do contador de programa (PC), ou seja, se o PC não tiver que apontar para o endereço seguinte na memória de programa, mas sim para outro (como no caso de saltos ou de chamadas de subrotinas), então deverá

considerar-se que a execução desta instrução demora dois ciclos. Isto acontece, porque a instrução vai ter que ser processada de novo, mas, desta vez, a partir do endereço correcto. O ciclo de chamada começa na fase Q1, escrevendo a instrução no registo de instrução (Instruction Register – IR). A descodificação e execução continua nas fases Q2, Q3 e Q4 do clock.



Fluxograma das Instruções no Pipeline

TCY0 é lido da memória o código da instrução MOVLW 55h (não nos interessa a instrução que foi executada, por isso não está representada por rectângulo).

TCY1 é executada a instrução MOVLW 55h e é lida da memória a instrução MOVWF PORTB.

TCY2 é executada a instrução MOVWF PORTB e lida a instrução CALL SUB_1.

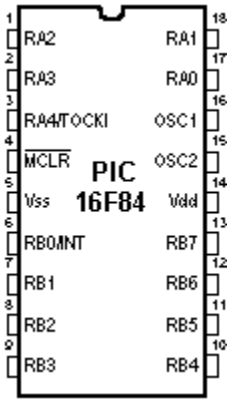
TCY3 é executada a chamada (call) de um subprograma CALL SUB_1 e é lida a instrução BSF PORTA,BIT3. Como esta instrução não é a que nos interessa, ou seja, não é a primeira instrução do subprograma SUB_1, cuja execução é o que vem a seguir, a leitura de uma instrução tem que ser feita de novo. Este é um bom exemplo de uma instrução a precisar de mais que um ciclo.

TCY4 este ciclo de instrução é totalmente usado para ler a primeira instrução do subprograma no endereço SUB_1.

TCY5 é executada a primeira instrução do subprograma SUB_1 e lida a instrução seguinte.

6 - Significado dos pinos

O PIC16F84 tem um total de 18 pinos. É mais frequentemente encontrado num tipo de encapsulamento DIP18, mas, também pode ser encontrado numa cápsula SMD de menores dimensões que a DIP. DIP é uma abreviatura para Dual In Package (Empacotamento em duas linhas). SMD é uma abreviatura para Surface Mount Devices (Dispositivos de Montagem em Superfície), o que sugere que os pinos não precisam de passar pelos orifícios da placa em que são inseridos, quando se solda este tipo de componente.



Os pinos no microcontrolador PIC16F84, têm o seguinte significado:

Pino nº 1, **RA2** Segundo pino do porto A. Não tem nenhuma função adicional.

Pino nº 2, **RA3** Terceiro pino do porto A. Não tem nenhuma função adicional.

Pino nº 3, **RA4** Quarto pino do porto A. O TOCK1 que funciona como entrada do temporizador, também utiliza este pino.

Pino nº 4, **MCLR** Entrada de reset e entrada da tensão de programação Vpp do microcontrolador .

Pino nº 5, **Vss** massa da alimentação.

Pino nº 6, **RB0**, bit 0 do porto B. Tem uma função adicional que é a de entrada de interrupção.

Pino nº 7, **RB1** bit 1do porto B. Não tem nenhuma função adicional.

Pino nº 8, **RB2** bit 2 do porto B. Não tem nenhuma função adicional.

Pino nº 9, **RB3** bit 3 do porto B. Não tem nenhuma função adicional.

Pino nº 10, **RB4** bit 4 do porto B. Não tem nenhuma função adicional.

Pino nº 11, **RB5** bit 5 do porto B. Não tem nenhuma função adicional.

Pino nº 12, **RB6** bit 6 do porto B. No modo de programa é a linha de clock

Pino nº 13, **RB7** bit 7 do porto B. Linha de dados no modo de programa

Pino nº 14, **Vdd** Pólo positivo da tensão de alimentação.

Pino nº 15, **OSC2** para ser ligado a um oscilador.

Pino nº 16, **OSC1** para ser ligado a um oscilador.

Pino nº 17, **RA0** bit 0 do porto A. Sem função adicional.

Pino nº 18, **RA1** bit 1 do porto A. Sem função adicional.

7.1 Gerador de relógio – oscilador

O circuito do oscilador é usado para fornecer um relógio (clock), ao microcontrolador. O clock é necessário para que o microcontrolador possa executar um programa ou as instruções de um programa.

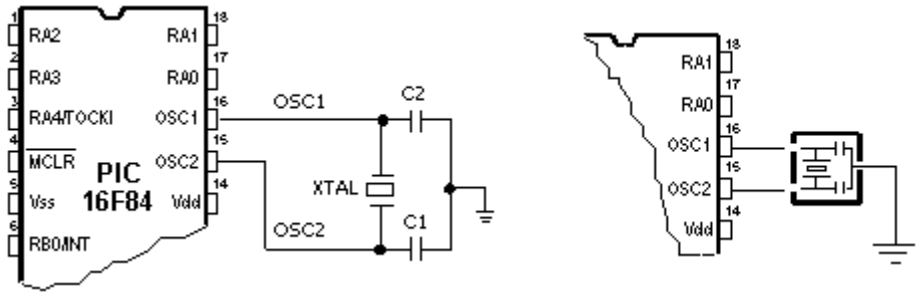
7.1.1 - Tipos de osciladores

O PIC16F84 pode trabalhar com quatro configurações de oscilador. Uma vez que as configurações com um oscilador de cristal e resistência-condensador (RC) são aquelas mais frequentemente usadas, elas são as únicas que vamos mencionar aqui.

Quando o oscilador é de cristal, a designação da configuração é de XT, se o oscilador for uma resistência em série com um condensador, tem a designação RC. Isto é importante, porque há necessidade de optar entre os diversos tipos de oscilador, quando se escolhe um microcontrolador.

7.1.1.1 - Oscilador XT

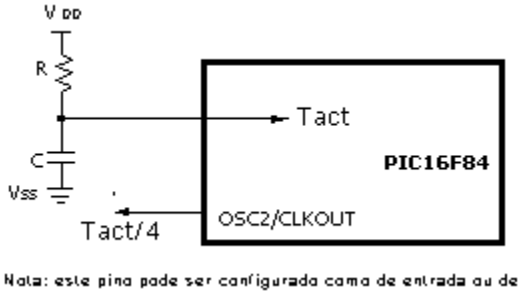
O oscilador de cristal está contido num envólucro de metal com dois pinos onde foi escrita a frequência a que o cristal oscila. Dois condensadores cerâmicos devem ligar cada um dos pinos do cristal à massa. Casos há em que cristal e condensadores estão contidos no mesmo encapsulamento, é também o caso do ressonador cerâmico ao lado representado. Este elemento tem três pinos com o pino central ligado à massa e os outros dois pinos ligados aos pinos OSC1 e OSC2 do microcontrolador. Quando projectamos um dispositivo, a regra é colocar o oscilador tão perto quanto possível do microcontrolador, de modo a evitar qualquer interferência nas linhas que ligam o oscilador ao microcontrolador.



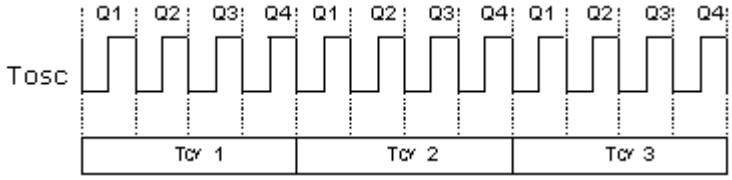
Clock de um microcontrolador a partir de um Clock de um microcontrolador com um ressonador cristal de quartzo

7.1.1.2 - OSCILADOR RC

Em aplicações em que a precisão da temporização não é um factor crítico, o oscilador RC torna-se mais económico. A frequência de ressonância do oscilador RC depende da tensão de alimentação, da resistência R, capacidade C e da temperatura de funcionamento.

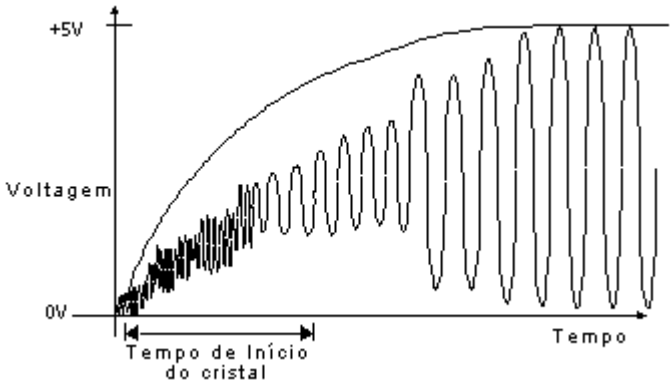


O diagrama acima, mostra como um oscilador RC deve ser ligado a um PIC16F84. Com um valor para a resistência R abaixo de 2,2 K, o oscilador pode tornar-se instável ou pode mesmo parar de oscilar. Para um valor muito grande R (1M por exemplo), o oscilador torna-se muito sensível à humidade e ao ruído. É recomendado que o valor da resistência R esteja compreendido entre 3K e 100K. Apesar de o oscilador poder trabalhar sem condensador externo (C = 0 pF), é conveniente, ainda assim, usar um condensador acima de 20 pF para evitar o ruído e aumentar a estabilidade. Qualquer que seja o oscilador que se está a utilizar, a frequência de trabalho do microcontrolador é a do oscilador dividida por 4. A frequência de oscilação dividida por 4 também é fornecida no pino OSC2/CLKOUT e, pode ser usada, para testar ou sincronizar outros circuitos lógicos pertencentes ao sistema.



Relação entre o sinal de clock e os ciclos de instrução

Ao ligar a alimentação do circuito, o oscilador começa a oscilar. Primeiro com um período de oscilação e uma amplitude instáveis, mas, depois de algum tempo, tudo estabiliza.



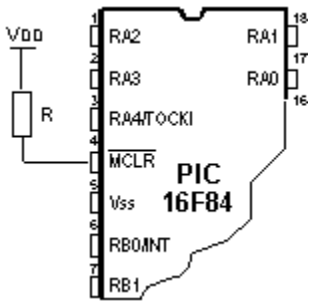
Sinal de clock do oscilador do microcontrolador depois de ser ligada a alimentação

Para evitar que esta instabilidade inicial do clock afecte o funcionamento do microcontrolador, nós necessitamos de manter o microcontrolador no estado de reset enquanto o clock do oscilador não estabiliza. O diagrama em cima, mostra uma forma típica do sinal fornecido por um oscilador de cristal de quartzo ao microcontrolador quando se liga a alimentação.

7.2 Reset

O reset é usado para pôr o microcontrolador num estado conhecido. Na prática isto significa que às vezes o microcontrolador pode comportar-se de um modo inadequado em determinadas condições indesejáveis. De modo a que o seu funcionamento normal seja restabelecido, é preciso fazer o reset do microcontrolador, isto significa que todos os seus registos vão conter valores iniciais pré-definidos, correspondentes a uma posição inicial. O reset não é usado somente quando o microcontrolador não se comporta da maneira que nós queremos, mas, também pode ser usado, quando ocorre uma interrupção por parte de outro dispositivo, ou quando se quer que o microcontrolador esteja pronto para executar um programa .

De modo a prevenir a ocorrência de um zero lógico acidental no pino MCLR (a linha por cima de MCLR significa o sinal de reset é activado por nível lógico baixo), o pino MCLR tem que ser ligado através de uma resistência ao lado positivo da alimentação. Esta resistência deve ter um valor entre 5 e 10K. Uma resistência como esta, cuja função é conservar uma determinada linha a nível lógico alto, é chamada "resistência de pull up".



Utilização do circuito interno de reset

O microcontrolador PIC16F84, admite várias formas de reset:

a) Reset quando se liga a alimentação, POR (Power-On Reset)

b) Reset durante o funcionamento normal, quando se põe a nível lógico baixo o pino MCLR do microcontrolador.

c) Reset durante o regime de SLEEP (dormir).

d) Reset quando o temporizador do watchdog (WDT) transborda (passa para 0 depois de atingir o valor máximo).

e) Reset quando o temporizador do watchdog (WDT) transborda estando no regime de SLEEP.

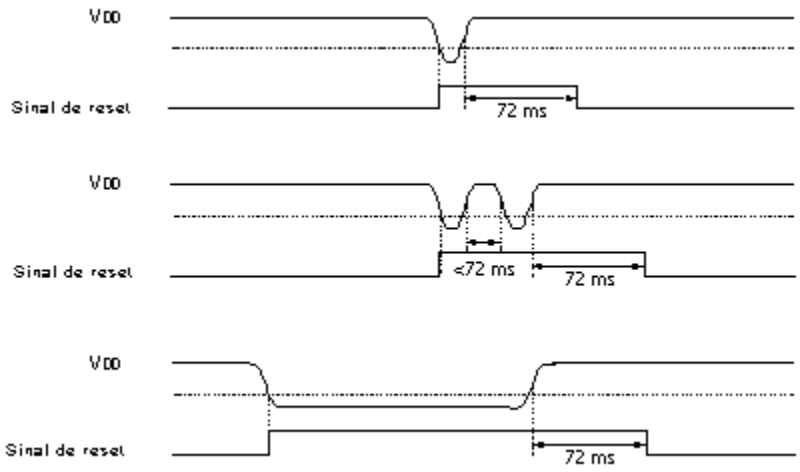
Os reset mais importantes são o a) e o b). O primeiro, ocorre sempre que é ligada a alimentação do microcontrolador e serve para trazer todos os registos para um estado inicial. O segundo que resulta da aplicação de um valor lógico baixo ao pino MCLR durante o funcionamento normal do microcontrolador e, é usado muitas vezes, durante o desenvolvimento de um programa.

Durante um reset, os locais de memória da RAM (registos) não são alterados. Ou seja, os conteúdos destes registos, são desconhecidos durante o restabelecimento da alimentação, mas mantêm-se inalterados durante qualquer outro reset. Ao contrário dos registos normais, os SFR (registos com funções especiais) são reiniciados com um valor inicial pré-definido. Um dos mais importantes efeitos de um reset, é introduzir no contador de programa (PC), o valor zero (0000), o que faz com que o programa comece a ser executado a partir da primeira instrução deste.

Reset quando o valor da alimentação desce abaixo do limite permitido (Brown-out Reset).

O impulso que provoca o reset durante o estabelecimento da alimentação (power-up), é gerado pelo próprio microcontrolador quando detecta um aumento na tensão Vdd (numa faixa entre 1,2V e 1,8V). Esse impulso perdura durante 72ms, o que, em princípio, é tempo suficiente para que o oscilador estabilize. Esse intervalo de tempo de 72ms é definido por um temporizador interno PWRT, com um oscilador RC próprio. Enquanto PWRT estiver activo, o microcontrolador mantém-se no estado de reset. Contudo, quando o dispositivo está a trabalhar, pode surgir um problema não resultante de uma queda da tensão para 0 volts, mas sim de uma queda de tensão para um valor abaixo do limite que garante o correcto funcionamento do microcontrolador. Trata-se de um facto muito provável de ocorrer na prática, especialmente em ambientes industriais onde as perturbações e instabilidade da alimentação ocorrem frequentemente. Para resolver este problema, nós

precisamos de estar certos de que o microcontrolador entra no estado de reset de cada vez que a alimentação desce abaixo do limite aprovado.

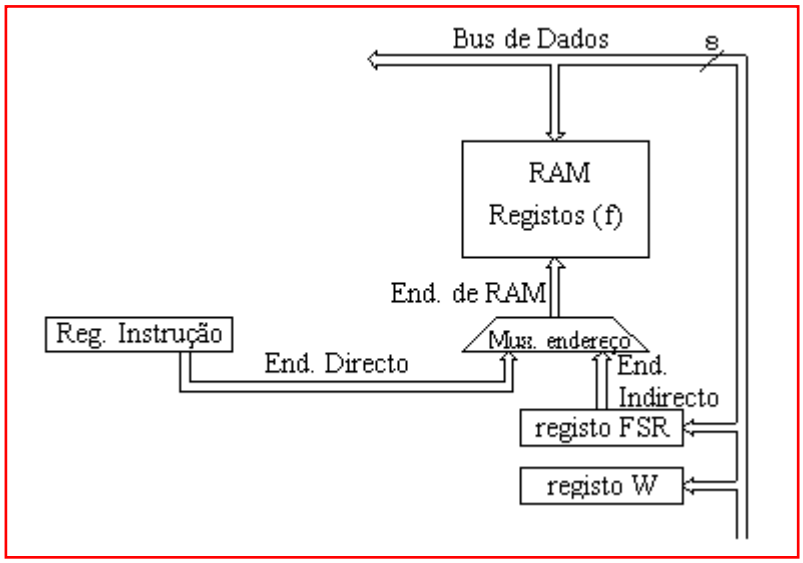


Exemplos de quedas na alimentação abaixo do limite

Se, de acordo com as especificações eléctricas, o circuito interno de reset de um microcontrolador não satisfizer as necessidades, então, deverão ser usados componentes electrónicos especiais, capazes de gerarem o sinal de reset desejado. Além desta função, estes componentes, podem também cumprir o papel de vigiarem as quedas de tensão para um valor abaixo de um nível especificado. Quando isto ocorre, aparece um zero lógico no pino MCLR, que mantém o microcontrolador no estado de reset, enquanto a voltagem não estiver dentro dos limites que garantem um correcto funcionamento.

7.3 Unidade Central de Processamento

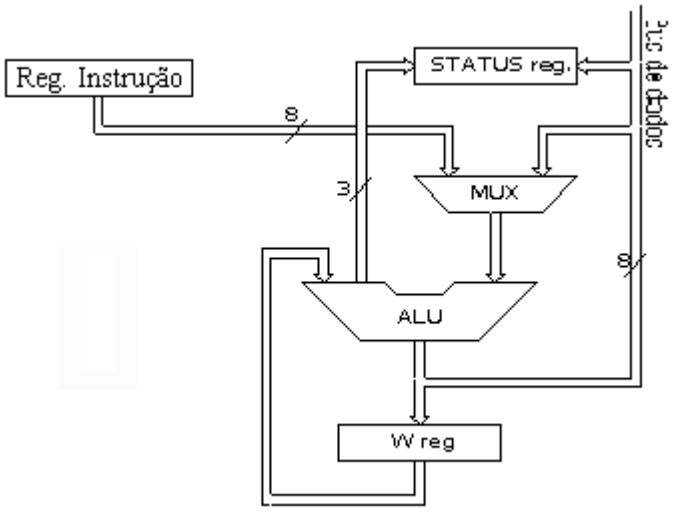
A unidade central de processamento (CPU) é o cérebro de um microcontrolador. Essa parte é responsável por extrair a instrução, descodificar essa instrução e, finalmente, executá-la.



Esquema da unidade central de processamento - CPU

A unidade central de processamento, interliga todas as partes do microcontrolador de modo a que este se comporte como um todo. Uma das sua funções mais importante é, seguramente, descodificar as instruções do programa. Quando o programador escreve um programa, as instruções assumem um claro significado como é o caso por exemplo de MOVLW 0x20. Contudo, para que um microcontrolador possa entendê-las, esta forma escrita de uma instrução tem que ser traduzida numa série de zeros e uns que é o 'opcode' (operation code ou código da operação). Esta passagem de uma palavra escrita para a forma binária é executada por tradutores assembler (ou simplesmente assembler). O código da instrução extraído da memória de programa, tem que ser descodificado pela unidade central de processamento (CPU). A cada uma das instruções do reportório do microcontrolador, corresponde um conjunto de acções para a concretizar. Estas acções, podem envolver transferências de dados de um local de memória para outro, de um local de memória para os portos, e diversos cálculos, pelo que, se conclui que, o CPU, tem que estar ligado a todas as partes do microcontrolador. Os bus de de dados e o de endereço permitem-nos fazer isso.

Unidade Lógica Aritmética (ALU) – A unidade lógica aritmética (ALU – Arithmetic Logic Unit), é responsável pela execução de operações de adição, subtracção, deslocamento (para a esquerda ou para a direita dentro de um registo) e operações lógicas. O PIC16F84 contém uma unidade lógica aritmética de 8 bits e registos de uso genérico também de 8 bits.



Unidade lógica-aritmética e como funciona

Por operando nós designamos o conteúdo sobre o qual uma operação incide. Nas instruções com dois operandos, geralmente um operando está contido no registo de trabalho W (working register) e o outro operando ou é uma constante ou então está contido num dos outros registos. Esses registos podem ser "Registos de Uso Genérico" (General Purpose Registers – GPR) ou "Registos com funções especiais" (Special Function Registers – SFR). Nas instruções só com um operando, um dos operandos é o conteúdo do registo W ou o conteúdo de um dos outros registos. Quando são executadas operações lógicas ou aritméticas como é o caso da adição, a ALU controla o estado dos bits (que constam do registo de estado – STATUS). Dependendo da instrução a ser executada, a ALU, pode modificar os valores bits do Carry (C), Carry de dígito (DC) e Z (zero) no registo de estado - STATUS.

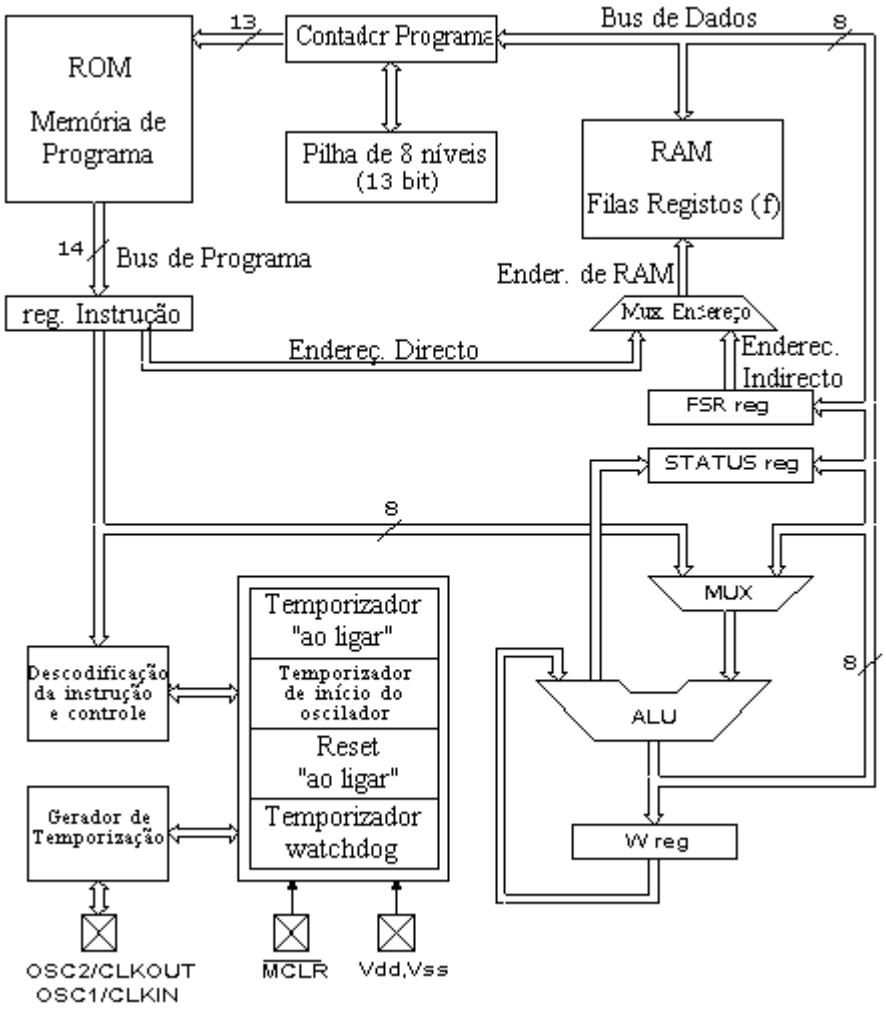
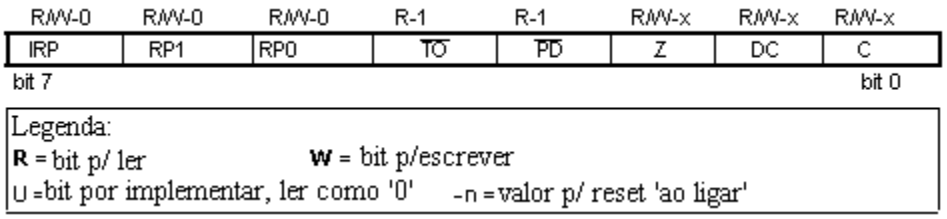


Diagrama bloco mais detalhado do microcontrolador PIC16F84

Registo STATUS



Bit 0 C (Carry) Transporte

Este bit é afectado pelas operações de adição, subtracção e deslocamento. Toma o valor '1' (set), quando um valor mais pequeno é subtraído de um valor maior e toma o valor '0' (reset) quando um valor maior é subtraído de um menor.

1= Ocorreu um transporte no bit mais significativo

0= Não ocorreu transporte no bit mais significativo

O bit C é afectado pelas instruções ADDWF, ADDLW, SUBLW e SUBWF.

Bit 1 DC (Digit Carry) Transporte de dígito

Este bit é afectado pelas operações de adição, subtracção. Ao contrário do anterior, DC assinala um transporte do bit 3 para o bit 4 do resultado. Este bit toma o valor '1', quando um valor mais pequeno é subtraído de um valor maior e toma o valor '0' quando um valor maior é subtraído de um menor.

1= Ocorreu um transporte no quarto bit mais significativo

0= Não ocorreu transporte nesse bit

O bit DC é afectado pelas instruções ADDWF, ADDLW, SUBLW e SUBWF.

Bit 2 Z (bit Zero) Indicação de resultado igual a zero.

Este bit toma o valor '1' quando o resultado da operação lógica ou aritmética executada é igual a 0.

1= resultado igual a zero

0= resultado diferente de zero

bit 3 PD (Bit de baixa de tensão – Power Down)

Este bit é posto a '1' quando o microcontrolador é alimentado e começa a trabalhar, depois de um reset normal e depois da execução da instrução CLRWDT. A instrução SLEEP põe este bit a '0' ou seja, quando o microcontrolador entra no regime de baixo consumo / pouco trabalho. Este bit pode também ser posto a '1', no caso de ocorrer um impulso no pino RB0/INT, uma variação nos quatro bits mais significativos do porto B, ou quando é completada uma operação de escrita na DATA EEPROM ou ainda pelo watchdog.

1 = depois de ter sido ligada a alimentação

0 = depois da execução de uma instrução SLEEP

Bit 4 TO Time-out ; transbordo do Watchdog

Este bit é posto a '1', depois de a alimentação ser ligada e depois da execução das instruções CLRWDT e SLEEP. O bit é posto a '0' quando o watchdog consegue chegar ao fim da sua contagem (overflow = transbordar), o que indica que qualquer coisa não esteve bem.

1 = não ocorreu transbordo

0 = ocorreu transbordo

Bits 5 e 6 RP1:RP0 (bits de selecção de banco de registos)

Estes dois bits são a parte mais significativa do endereço utilizado para endereçamento directo. Como as instruções que endereçam directamente a memória, dispõem somente de sete bits para este efeito, é preciso mais um bit para poder endereçar todos os 256 registos do PIC16F84. No caso do PIC16F84, RP1, não é usado, mas pode ser necessário no caso de outros microcontroladores PIC, de maior capacidade.

01 = banco de registos 1

00 = banco de registos 0

bit 7 IRP (Bit de selecção de banco de registos)

Este bit é utilizado no endereçamento indirecto da RAM interna, como oitavo bit

1 = bancos 2 e 3

0 = bancos 0 e 1 (endereços de 00h a FFh)

O registo de estado (STATUS), contém o estado da ALU (C, DC, Z), estado de RESET (TO, PD) e os bits para selecção do banco de memória (IRP, RP1, RP0). Considerando que a selecção do banco de memória é controlada através deste registo, ele tem que estar presente em todos os bancos. Os bancos de memória serão discutidos com mais detalhe no capítulo que trata da Organização da Memória. Se o registo STATUS for o registo de destino para instruções que afectem os bits Z, DC ou C, então não é possível escrever nestes três bits.

Registo OPTION

R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1
$\overline{\text{RBPU}}$ (1)	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0
bit 7							bit 0

Legenda:
R = bit p/ ler **W** = bit p/ escrever
U =não implementado, ler como '0' -n = valor p/ reset 'ao ligar'

Bits 0 a 2 PS0, PS1, PS2 (bits de selecção do divisor Prescaler)

Estes três bits definem o factor de divisão do prescaler. Aquilo que é o prescaler e o modo como o valor destes três bits afectam o funcionamento do microcontrolador será estudado na secção referente a TMR0.

Bits	TMR0	WDT
000	1 : 2	1 : 1
001	1 : 4	1 : 2
010	1 : 8	1 : 4
011	1 : 16	1 : 8
100	1 : 32	1 : 16
101	1 : 64	1 : 32
110	1 : 128	1 : 64
111	1 : 256	1 : 128

Bit **3 PSA** (Bit de Atribuição do Prescaler)

Bit que atribui o prescaler ao TMR0 ou ao watchdog.

1 = prescaler atribuído ao watchdog

0 = prescaler atribuído ao temporizador TMR0

Bit 4 T0SE (bit de selecção de bordo activo em TMR0)

Se for permitido aplicar impulsos em TMR0, a partir do pino RA4/TOCK1, este bit determina se os impulsos activos são os impulsos ascendentes ou os impulsos descendentes.

1 = bordo descendente

0 = bordo ascendente

Bit 5 TOCS (bit de selecção de fonte de clock em TMR0)

Este pino escolhe a fonte de impulsos que vai ligar ao temporizador. Esta fonte pode ser o clock do microcontrolador (frequência de clock a dividir por 4) ou impulsos externos no pino RA4/TOCKI.

1 = impulsos externos

0 = ¼ do clock interno

Bit 6 INDEDG (bit de selecção de bordo de interrupção)

Se esta interrupção estiver habilitada, é possível definir o bordo que vai activar a interrupção no pino RB0/INT.

1 = bordo ascendente

0 = bordo descendente

Bit 7 RBPU (Habilitação dos pull-up nos bits do porto B)

Este bit introduz ou retira as resistências internas de pull-up do porto B.

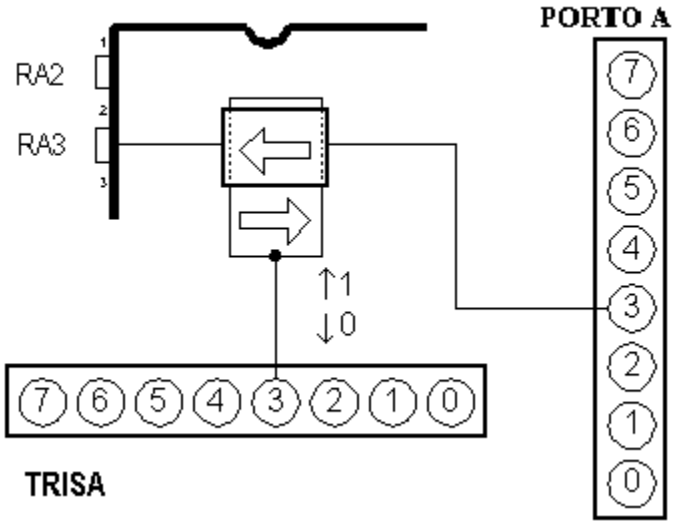
1 = resistências de "pull-up" desligadas

0 = resistências de "pull-up" ligadas

7.4 Portos

Porto, é um grupo de pinos num microcontrolador que podem ser acedidos simultaneamente, e, no qual nós podemos colocar uma combinação de zeros e uns ou ler dele o estado existente. Fisicamente, porto é um registo dentro de um microcontrolador que está ligado por fios aos pinos do microcontrolador. Os portos representam a conexão física da Unidade Central de Processamento (CPU) com o mundo exterior. O microcontrolador usa-os para observar ou comandar outros componentes ou dispositivos. Para aumentar a sua funcionalidade, os mesmos pinos podem ter duas aplicações distintas, como, por exemplo, RA4/TOCKI, que é simultaneamente o bit 4 do porto A e uma entrada externa para o contador/temporizador TMR0. A escolha de uma destas duas funções é feita através dos

registos de configuração. Um exemplo disto é o TOCS, quinto bit do registo OPTION. Ao seleccionar uma das funções, a outra é automaticamente inibida.



Relação entre os registos TRISA e PORTO A

Todos os pinos dos portos podem ser definidos como de entrada ou de saída, de acordo com as necessidades do dispositivo que se está a projectar. Para definir um pino como entrada ou como saída, é preciso, em primeiro lugar, escrever no registo TRIS, a combinação apropriada de zeros e uns. Se no local apropriado de um registo TRIS for escrito o valor lógico "1", então o correspondente pino do porto é definido como entrada, se suceder o contrário, o pino é definido como saída. Todos os portos, têm um registo TRIS associado. Assim, para o porto A, existe o registo TRISA no endereço 85h e, para o porto B existe o registo TRISB, no endereço 86h.

PORTO B

O porto B tem 8 pinos associados a ele. O respectivo registo de direcção de dados chama-se TRISB e tem o endereço 86h. Ao pôr a '1' um bit do registo TRISB, define-se o correspondente pino do porto como entrada e se pusermos a '0' um bit do registo TRISB, o pino correspondente vai ser uma saída. Cada pino do PORTO B possui uma pequena resistência de 'pull-up' (resistência que define a linha como tendo o valor lógico '1'). As resistências de pull-up são activadas pondo a '0' o bit RBPU, que é o bit 7 do registo OPTION. Estas resistências de 'pull-up' são automaticamente desligadas quando os pinos

do porto são configurados como saídas. Quando a alimentação do microcontrolador é ligada, as resistências de pull-up são também desactivadas.

Quatro pinos do PORTO B, RB4 a RB7 podem causar uma interrupção, que ocorre quando qualquer deles varia do valor lógico zero para valor lógico um ou o contrário. Esta forma de interrupção só pode ocorrer se estes pinos forem configurados como entradas (se qualquer um destes 4 pinos for configurado como saída, não será gerada uma interrupção quando há variação de estado). Esta modalidade de interrupção, acompanhada da existência de resistências de pull-up internas, torna possível resolver mais facilmente problemas frequentes que podemos encontrar na prática, como por exemplo a ligação de um teclado matricial. Se as linhas de um teclado ficarem ligadas a estes pinos, sempre que se prime uma tecla, ir-se-á provocar uma interrupção. Ao processar a interrupção, o microcontrolador terá que identificar a tecla que a produziu. Não é recomendável utilizar o porto B, ao mesmo tempo que esta interrupção está a ser processada.

```
clrf   STATUS       ;Banco 0
clrf   PORTB        ;Porto B = 0
bsf    STATUS,RP0   ;Banco 1
movlw  0x0F         ;Definir pinos de entrada e saída
movwf  TRISB        ;Escrever no registo TRISB
```

O exemplo de cima mostra como os pinos 0, 1, 2 e 3 são definidos como entradas e 4, 5, 6 e 7 como saídas.

PORTO A

O porto A (PORTA) está associado a 5 pinos. O registo de direcção de dados correspondente é o TRISA, no endereço 85h. Tal como no caso do porto B, pôr a '1' um bit do registo TRISA, equivale a definir o correspondente pino do porto A, como entrada e pôr a '0' um bit do mesmo registo, equivale a definir o correspondente pino do porto A, como saída.

O quinto pino do porto A tem uma função dupla. Nesse pino está também situada a entrada externa do temporizador TMR0. Cada uma destas opções é escolhida pondo a '1' ou pondo a '0' o bit TOCS (bit de selecção de fonte de clock de TMR0). Conforme o valor deste bit, assim o temporizador TMR0 incrementa o seu valor por causa de um impulso do oscilador interno ou devido a um impulso externo aplicado ao pino RA4/TOCKI.

```
bcf    STATUS,RP0   ;Banco 0
clrf   PORTA        ;Porto A = 0
bsf    STATUS,RP0   ;Banco 1
movlw  0x1F         ;Definir pinos de entrada e  de saída
movwf  TRISA        ;Escrever no registo TRISA
```

Este exemplo mostra como os pinos 0, 1, 2, 3 e 4 são declarados como entradas e os pinos 5, 6 e 7 como pinos de saída.

7.5 Organização da memória

O PIC16F84 tem dois blocos de memória separados, um para dados e o outro para o programa. A memória EEPROM e os registos de uso genérico (GPR) na memória RAM constituem o bloco para dados e a memória FLASH constitui o bloco de programa.

7.5.1 - Memória de programa

A memória de programa é implementada usando tecnologia FLASH, o que torna possível programar o microcontrolador muitas vezes antes de este ser instalado num dispositivo, e, mesmo depois da sua instalação, podemos alterar o programa e parâmetros contidos. O tamanho da memória de programa é de 1024 endereços de palavras de 14 bits, destes, os endereços zero e quatro estão reservados respectivamente para o reset e para o vector de interrupção.

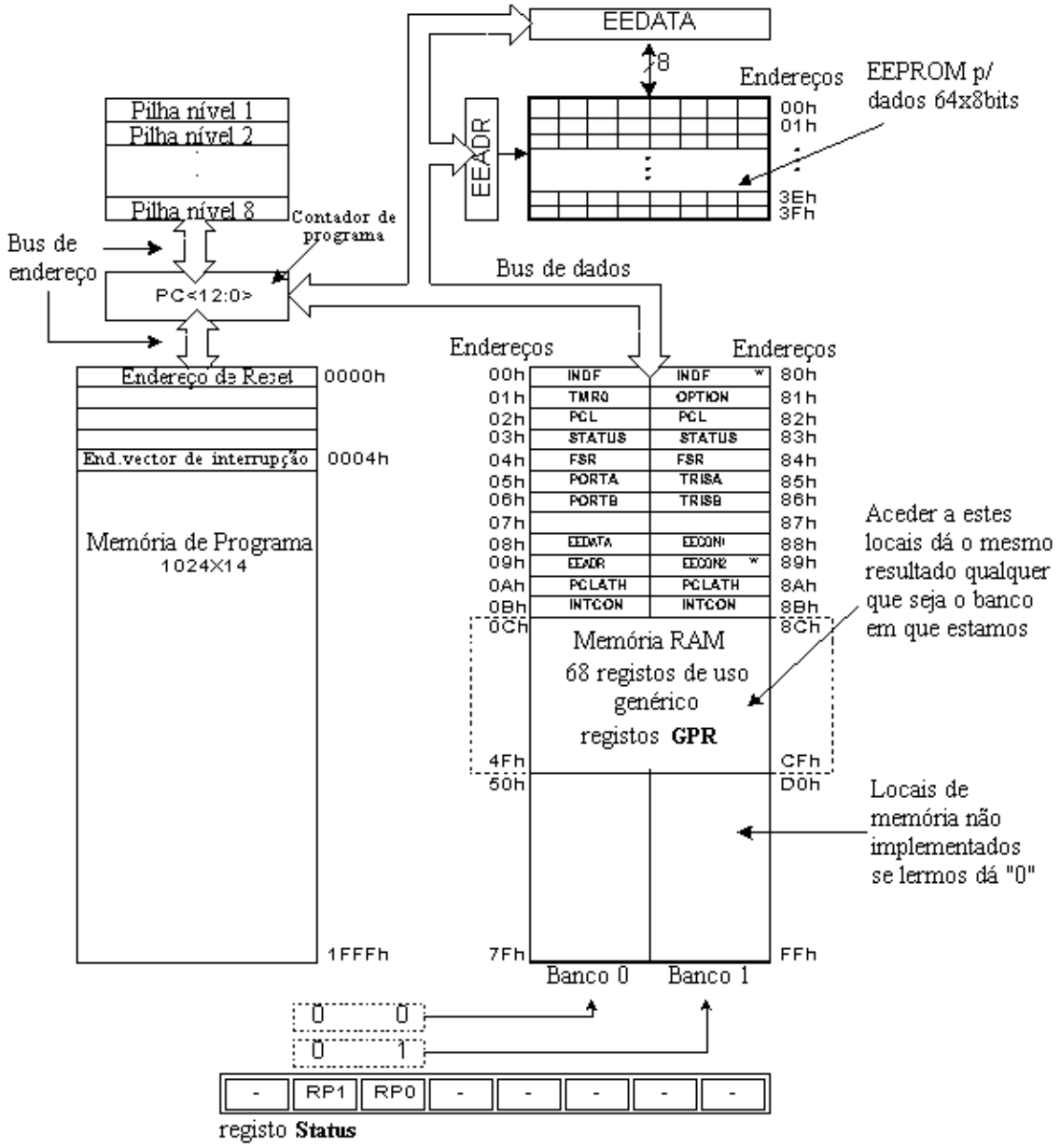
7.5.2 - Memória de dados

A memória de dados compreende memória EEPROM e memória RAM. A memória EEPROM consiste em 64 posições para palavras de oito bits e cujos conteúdos não se perdem durante uma falha na alimentação. A memória EEPROM não faz parte directamente do espaço de memória mas é acedida indirectamente através dos registos EEADR e EEDATA. Como a memória EEPROM serve usualmente para guardar parâmetros importantes (por exemplo, de uma dada temperatura em reguladores de temperatura), existe um procedimento estrito para escrever na EEPROM que tem que ser seguido de modo a evitar uma escrita acidental. A memória RAM para dados, ocupa um espaço no mapa de memória desde o endereço 0x0C até 0x4F, o que corresponde a 68 localizações. Os locais da memória RAM são também chamados registos GPR (General Purpose Registers = Registos

de uso genérico). Os registos GPR podem ser acedidos sem ter em atenção o banco em que nos encontramos de momento.

7.5.3 – Registos SFR

Os registos que ocupam as 12 primeiras localizações nos bancos 0 e 1 são registos especiais e têm a ver com a manipulação de certos blocos do microcontrolador. Estes registos são os SFR (Special Function Registers ou Registos de Funções Especiais).



Organização da memória no microcontrolador PIC16F84

7.5.4 - Bancos de Memória

Além da divisão em 'comprimento' entre registos SFR e GPR, o mapa de memória está também dividido em 'largura' (ver mapa anterior) em duas áreas chamadas 'bancos'. A selecção de um dos bancos é feita por intermédio dos bits RP0 e RP1 do registo STATUS.

Exemplo :

```
bcf STATUS, RP0
```

A instrução BCF "limpa" o bit RP0 (RP0 = 0) do registo STATUS e, assim, coloca-nos no banco 0.

```
bsf STATUS, RP0
```

A instrução BSF põe a um, o bit RP0 (RP0 = 1) do registo STATUS e, assim, coloca-nos no banco 1.

Normalmente, os grupos de instruções muito usados são ligados numa única unidade que pode ser facilmente invocada por diversas vezes num programa, uma unidade desse tipo chama-se genericamente Macro e, normalmente, essa unidade é designada por um nome especifico facilmente compreensível. Com a sua utilização, a selecção entre os dois bancos torna-se mais clara e o próprio programa fica mais legível.

```
BANK0 macro
Bcf STATUS, RP0 ;Selecionar o banco 0 da memória
Endm
BANK1 macro
Bsf STATUS, RP0 ; Selecionar o banco 1 da memória
Endm
```

Os locais de memória 0Ch – 4Fh são registos de uso genérico (GPR) e são usados como memória RAM. Quando os endereços 8Ch – CFh são acedidos, nós acedemos também às mesmas localizações do banco 0. Por outras palavras, quando estamos a trabalhar com os registos de uso genérico, não precisamos de nos preocupar com o banco em que nos encontramos!

7.5.5 – Contador de Programa

O contador de programa (PC = Program Counter), é um registo de 13 bits que contém o endereço da instrução que vai ser executada. Ao incrementar ou alterar (por exemplo no caso de saltos) o conteúdo do PC, o microcontrolador consegue executar todas as instruções do programa, uma após outra.

7.5.6 – Pilha

O PIC16F84 tem uma pilha (stack) de 13 bits e 8 níveis de profundidade, o que corresponde a 8 locais de memória com 13 bits de largura. O seu papel básico é guardar o valor do contador de programa quando ocorre um salto do programa principal para o endereço de um subprograma a ser executado. Depois de ter executado o subprograma, para que o microcontrolador possa continuar com o programa principal a partir do ponto em que o deixou, ele tem que ir buscar à pilha esse endereço e carregá-lo no contador de programa. Quando nos movemos de um programa para um subprograma, o conteúdo do contador de programa é empurrado para o interior da pilha (um exemplo disto é a instrução CALL). Quando são executadas instruções tais como RETURN, RETLW ou RETFIE no fim de um subprograma, o contador de programa é retirado da pilha, de modo a que o programa possa continuar a partir do ponto em que a sequência foi interrompida. Estas operações de colocar e extrair da pilha o contador de programa, são designadas por PUSH (meter na pilha) e POP (tirar da pilha), estes dois nomes provêm de instruções com estas designações, existentes nalguns microcontroladores de maior porte.

7.5.7 – Programação no Sistema

Para programar a memória de programa, o microcontrolador tem que entrar num modo especial de funcionamento no qual o pino MCLR é posto a 13,5V e a voltagem da alimentação Vdd deve permanecer estável entre 4,5V e 5,5V. A memória de programa pode ser programada em série, usando dois pinos 'data/clock' que devem ser previamente separados do dispositivo em que o microcontrolador está inserido, de modo a que não possam ocorrer erros durante a programação.

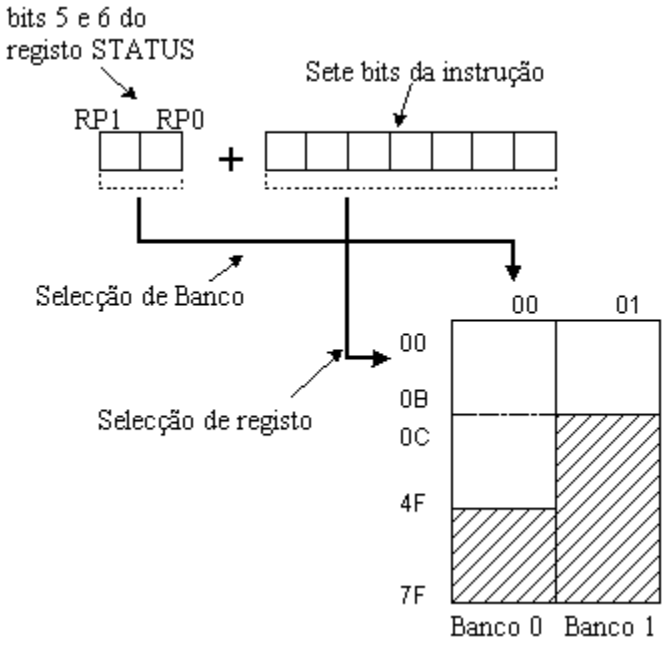
7.5.8 – Modos de endereçamento

Os locais da memória RAM podem ser acedidos directa ou indirectamente.

7.5.9- Endereçamento Directo

O endereçamento directo é feito através de um endereço de 9 bits. Este endereço obtém-se juntando aos sete bits do endereço directo de uma instrução, mais dois bits (RP1 e RP0) do registo STATUS, como se mostra na figura que se segue. Qualquer acesso aos registos especiais (SFR), pode ser um exemplo de endereçamento directo.

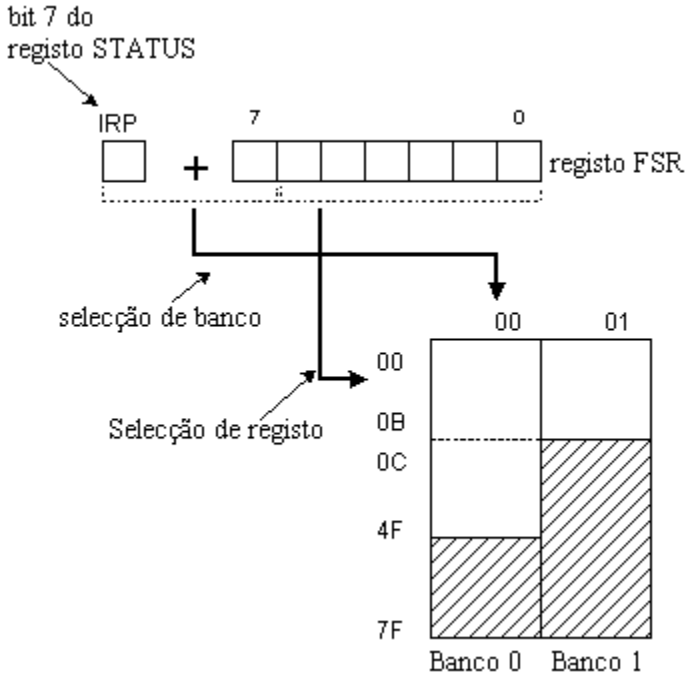
```
Bsf   STATUS,
RP0            ; Banco 1

movlw 0xFF     ; w = 0xFF

movwf TRISA    ; o endereço do registo TRISA é tirado do código da instrução movwf TRISA
```



Endereçamento Directo

7.5.10 - Endereçamento Indirecto

O endereçamento indirecto, ao contrário do directo, não tira um endereço do código instrução, mas fá-lo com a ajuda do bit IRP do registo STATUS e do registo FSR. O local endereçado é acedido através do registo INDF e coincide com o endereço contido em FSR. Por outras palavras, qualquer instrução que use INDF como registo, na realidade acede aos dados apontados pelo registo FSR. Vamos supor, por exemplo, que o registo de uso genérico de endereço 0Fh contém o valor 20. Escrevendo o valor de 0Fh no registo FSR, nós vamos obter um ponteiro para o registo 0Fh e, ao ler o registo INDF, nós iremos obter o valor 20, o que significa que lemos o conteúdo do registo 0Fh, sem o mencionar explicitamente (mas através de FSR e INDF). Pode parecer que este tipo de endereçamento não tem quaisquer vantagens sobre o endereçamento directo, mas existem problemas que só podem ser resolvidos de uma forma simples, através do endereçamento indirecto.



Endereçamento Indirecto

Um exemplo pode ser enviar um conjunto de dados através de uma comunicação série, usando buffers e indicadores (que serão discutidos num capítulo mais à frente, com

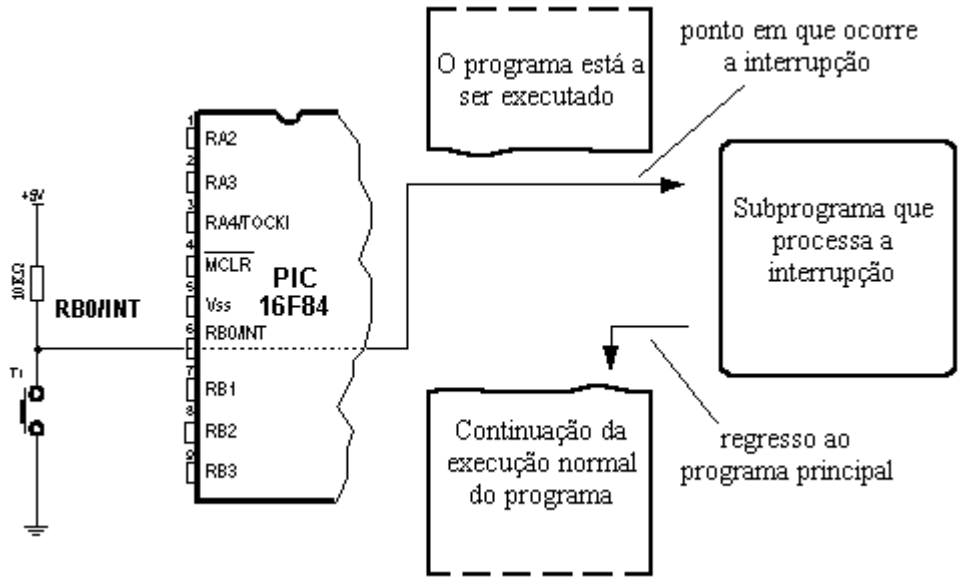
exemplos), outro exemplo é limpar os registos da memória RAM (16 endereços neste caso) como se pode ver a seguir.

```
        Movlw  0x0C          ;definição do endereço de início
        Movwf  FSR           ;FSR aponta p/ o endereço 0x0C
LOOP    clrf   INDF          ;INDF = 0
        incf   FSR           ;endereço = endereço inicial + 1
        btfss  FSR,4         ;todos os locais de memória limpos?
        goto   loop          ;não, para ' loop' de novo
CONTINUE
        :                    ;sim, continuar com o programa
```

Quando o conteúdo do registo FSR é igual a zero, ler dados do registo INDF resulta no valor 0 e escrever em INDF resulta na instrução NOP (no operation = nenhuma operação).

7.6 Interrupções

As interrupções são um mecanismo que o microcontrolador possui e que torna possível responder a alguns acontecimentos no momento em que eles ocorrem, qualquer que seja a tarefa que o microcontrolador esteja a executar no momento. Esta é uma parte muito importante, porque fornece a ligação entre um microcontrolador e o mundo real que nos rodeia. Geralmente, cada interrupção muda a direcção de execução do programa, suspendendo a sua execução, enquanto o microcontrolador corre um subprograma que é a rotina de atendimento de interrupção. Depois de este subprograma ter sido executado, o microcontrolador continua com o programa principal, a partir do local em que o tinha abandonado.



Uma das possíveis fontes de interrupção e como afecta o programa principal

O registo que controla as interrupções é chamado INTCON e tem o endereço 0Bh. O papel do INTCON é permitir ou impedir as interrupções e, mesmo no caso de elas não serem permitidas, ele toma nota de pedidos específicos, alterando o nível lógico de alguns dos seus bits.

Registo INTCON

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
GIE	PEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF
bit 7							bit 0

Legend:
R = bit p/ ler **W** = bit p/ escrever
U = bit não implementado lido como '0' - n = valor no reset 'ao ligar'

Bit 0 RBIF (flag que indica variação no porto B) Bit que informa que houve mudança nos níveis lógicos nos pinos 4, 5, 6 e 7 do porto B.

1= pelo menos um destes pinos mudou de nível lógico

0= não ocorreu nenhuma variação nestes pinos

Bit 1 INTF (flag de interrupção externa INT) Ocorrência de uma interrupção externa

1= ocorreu uma interrupção externa

0= não ocorreu uma interrupção externa

Se um impulso ascendente ou descendente for detectado no pino RB0/INT, o bit INTF é posto a '1' (o tipo de sensibilidade, ascendente ou descendente é definida através do bit INTEDG do registo OPTION). O subprograma de atendimento desta interrupção, deve repor este bit a '0', afim de que a próxima interrupção possa ser detectada.

Bit 2 TOIF (Flag de interrupção por transbordo de TMR0) O contador TMR0, transbordou.

1= o contador mudou a contagem de FFh para 00h

0= o contador não transbordou

Para que esta interrupção seja detectada, o programa deve pôr este bit a '0'

Bit 3 RBIE (bit de habilitação de interrupção por variação no porto B) Permite que a interrupção por variação dos níveis lógicos nos pinos 4, 5, 6 e 7 do porto B, ocorra.

1= habilita a interrupção por variação dos níveis lógicos

0= inibe a interrupção por variação dos níveis lógicos

A interrupção só pode ocorrer se RBIE e RBIF estiverem simultaneamente a '1' lógico.

Bit 4 INTE (bit de habilitação da interrupção externa INT) bit que permite uma interrupção externa no bit RB0/INT.

1= interrupção externa habilitada

0= interrupção externa impedida

A interrupção só pode ocorrer se INTE e INTF estiverem simultaneamente a '1' lógico.

Bit 5 TOIE (bit de habilitação de interrupção por transbordo de TMR0) bit que autoriza a interrupção por transbordo do contador TMR0.

1= interrupção autorizada

0= interrupção impedida

A interrupção só pode ocorrer se TOIE e TOIF estiverem simultaneamente a '1' lógico.

Bit 6 EEIE (bit de habilitação de interrupção por escrita completa, na EEPROM) bit que habilita uma interrupção quando uma operação de escrita na EEPROM termina.

1= interrupção habilitada

0= interrupção inibida

Se EEIE e EEIF (que pertence ao registo EECON1) estiverem simultaneamente a '1', a interrupção pode ocorrer.

Bit 7 GIE (bit de habilitação global de interrupção) bit que permite ou impede todas as interrupções

1= todas as interrupções são permitidas

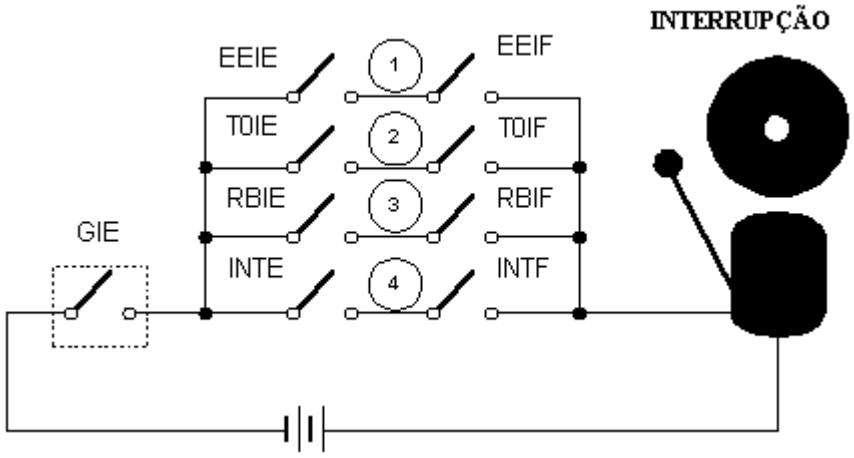
0= todas as interrupções impedidas

O PIC16F84 possui quatro fontes de interrupção:

1. Fim de escrita na EEPROM
2. Interrupção em TMR0 causada por transbordo do temporizador
3. Interrupção por alteração nos pinos RB4, RB5, RB6 e RB7 do porto B.
4. Interrupção externa no pino RB0/INT do microcontrolador

De um modo geral, cada fonte de interrupção tem dois bits associados. Um habilita a interrupção e o outro assinala quando a interrupção ocorre. Existe um bit comum a todas as interrupções chamado GIE que pode ser usado para impedir ou habilitar todas as interrupções, simultaneamente. Este bit é muito útil quando se está a escrever um programa porque permite que todas as interrupções sejam impedidas durante um período de tempo, de tal maneira que a execução de uma parte crítica do programa não possa ser interrompida.

Quando a instrução que faz GIE= 0 é executada (GIE= 0 impede todas as interrupções), todas os pedidos de interrupção pendentes, serão ignorados.



Esquema das interrupções no microcontrolador PIC16F84

As interrupções que estão pendentes e que são ignoradas, são processadas quando o bit GIE é posto a '1' (GIE= 1, todas as interrupções permitidas). Quando a interrupção é atendida, o bit GIE é posto a '0', de tal modo que, quaisquer interrupções adicionais sejam inibidas, o endereço de retorno é guardado na pilha e, no contador de programa, é escrito 0004h – somente depois disto, é que a resposta a uma interrupção começa!

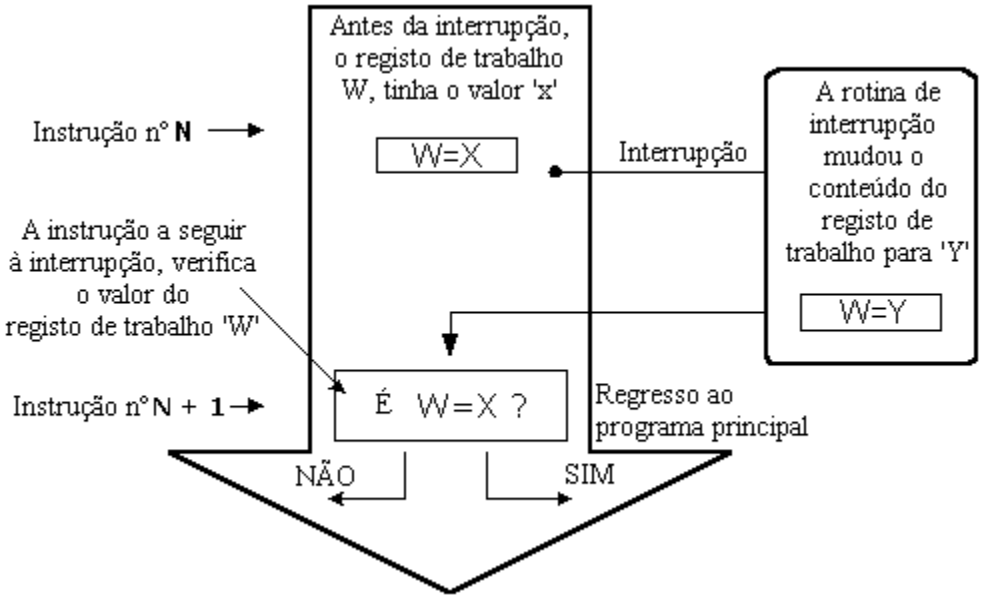
Depois de a interrupção ser processada, o bit que por ter sido posto a '1' permitiu a interrupção, deve agora ser reposto a '0', senão, a rotina de interrupção irá ser automaticamente processada novamente, mal se efectue o regresso ao programa principal.

7.6.1 - Guardando os conteúdos dos registos importantes

A única coisa que é guardada na pilha durante uma interrupção é o valor de retorno do contador de programa (por valor de retorno do contador de programa entende-se o endereço da instrução que estava para ser executada, mas que não foi, por causa de ter ocorrido a interrupção). Guardar apenas o valor do contador de programa não é, muitas vezes, suficiente. Alguns registos que já foram usados no programa principal, podem também vir a ser usados na rotina de interrupção. Se nós não salvaguardamos os seus valores, quando acontece o regresso da subrotina para o programa principal os conteúdos dos registos podem ser inteiramente diferentes, o que causaria um erro no programa. Um exemplo para este caso é o conteúdo do registo de trabalho W (work register). Se supormos

que o programa principal estava a usar o registo de trabalho W nalgumas das suas operações e se ele contiver algum valor que seja importante para a instrução seguinte, então a interrupção que ocorre antes desta instrução vai alterar o valor do registo de trabalho W, indo influenciar directamente o programa principal.

O procedimento para a gravação de registos importantes antes de ir para a subrotina de interrupção, designa-se por 'PUSH', enquanto que o procedimento que recupera esses valores, é chamado POP. PUSH e POP são instruções provenientes de outros microcontroladores (da Intel), agora esses nomes são aceites para designar estes dois processos de salvaguarda e recuperação de dados. Como o PIC16F84 não possui instruções comparáveis, elas têm que ser programadas.



Uma das possíveis causas de erros é não salvaguardar dados antes de executar um subprograma de interrupção

Devido à sua simplicidade e uso frequente, estas partes do programa podem ser implementadas com macros. O conceito de Macro é explicado em "Programação em linguagem Assembly". No exemplo que se segue, os conteúdos de W e do registo STATUS são guardados nas variáveis W_TEMP e STATUS_TEMP antes de correr a rotina de interrupção. No início da rotina PUSH, nós precisamos de verificar qual o banco que está a ser seleccionado porque W_TEMP e STATUS_TEMP estão situados no banco 0. Para troca

de dados entre estes dois registos, é usada a instrução SWAPF em vez de MOVF, pois a primeira não afecta os bits do registo STATUS.

Exemplo é um programa assembler com os seguintes passos:

1. Verificar em que banco nos encontramos

2. Guardar o registo W qualquer que seja o banco em que nos encontramos

3. Guardar o registo STATUS no banco 0.

4. Executar a rotina de serviço de interrupção ISR (Interrupt Service Routine)

5. Recuperação do registo STATUS

6. Restaurar o valor do registo W

Se existirem mais variáveis ou registos que necessitem de ser salvaguardados, então, precisamos de os guardar depois de guardar o registo STATUS (passo 3) e recuperá-los depois de restaurar o registo STATUS (passo 5).

```
Push
        BTFSS STATUS, RP0           ; Banco 0?
        GOTO RP0CLEAR               ; Sim
        BCF STATUS, RP0             ; Não, ir p/ Banco 0
        MOVWF W_TEMP                ; Guardar registo W
        SWAPF STATUS, W             ; W <- STATUS
        MOVWF STATUS_TEMP           ; STATUS_TEMP <- W
        BSF STATUS_TEMP, 1          ; RP0(STATUS_TEMP)=1
        GOTO ISR_Code               ; Push completado
RP0CLEAR
        MOVWF W_TEMP                ; Guardar registo W
        SWAPF STATUS, W             ; W <- STATUS
        MOVWF STATUS_TEMP           ; STATUS_TEMP <- W
;
ISR_Code
        :
        :Subprograma de Interrupção
        :
;
Pop
        SWAPF STATUS_TEMP, W        ; W <- STATUS_TEMP
        MOVWF STATUS                ; STATUS <-W
        BTFSS STATUS, RP0           ; Banco 1?
        GOTO Return_WREG            ; Não
        BCF STATUS, RP0             ; Sim, ir p/ o banco 0
        SWAPF W_TEMP, F             ; Recuperar o conteúdo de W
        SWAPF W_TEMP, W             ;
        BSF STATUS, RP0             ; Regressar ao banco 1
        RETFIE                      ; POP completo
Return_WREG
        SWAPF W_TEMP, F             ; Recuperar o conteúdo de W
        SWAPF W_TEMP, W             ;
        RETFIE                      ; POP completo
```

A mesma operação pode ser realizada usando macros, desta maneira obtemos um programa mais legível. Os macros que já estão definidos podem ser usados para escrever novos macros. Os macros BANK1 e BANK0 que são explicados no capítulo "Organização da memória" são usados nos macros 'push' e 'pop'.

```
push    macro
        movwf  W_Temp              ;W_Temp <- W
        swapf  W_Temp,F            ;trocar a ordem dos bits
        BANK1                      ;Macro p/ aceder ao banco 1
        swapf  OPTION_REG,W        ;W <- OPTION_REG
        movwf  Option_Temp         ;Option_Temp <- W
        BANK0                      ;Macro p/ aceder ao banco 0
        swapf  STATUS,W            ;W <- STATUS
        movwf  Stat_Temp           ;Stat_Temp <-W
        endm                       ;Fim do macro push

pop     macro
        swapf  Stat_Temp,W         ;W <- Stat_Temp
        movwf  STATUS              ;STATUS <- W
        BANK1                      ;Macro p/ aceder ao banco 1
        swapf  Option_Temp,W       ;W <- Option_Temp
        movwf  OPTION_REG          ;OPTION_REG <- W
        BANK0                      ;Macro p/ aceder ao banco 0
        swapf  W_Temp,W            ;W <- W_Temp
        endm                       ;Fim do macro pop
```

Interrupção externa no pino RB0/INT do microcontrolador

A interrupção externa no pino RB0/ INT é desencadeada por um impulso ascendente (se o bit INTEDG = 1 no registo OPTION<6>), ou por um impulso descendente (se INTEDG = 0). Quando o sinal correcto surge no pino INT, o bit INTF do registo INTCON é posto a '1'. O bit INTF (INTCON<1>) tem que ser reposto a '0' na rotina de interrupção, afim de que a interrupção não possa voltar a ocorrer de novo, aquando do regresso ao programa principal. Esta é uma parte importante do programa e que o programador não pode esquecer, caso contrário o programa irá constantemente saltar para a rotina de interrupção. A interrupção pode ser inibida, pondo a '0' o bit de controle INTE (INTCON<4>).

7.6.2 - Interrupção devido ao transbordar (overflow) do contador TMR0

O transbordar do contador TMR0 (passagem de FFh para 00h) vai pôr a '1' o bit TOIF (INTCON<2>), Esta é uma interrupção muito importante, uma vez que, muitos problemas da vida real podem ser resolvidos utilizando esta interrupção. Um exemplo é o da medição de tempo. Se soubermos de quanto tempo o contador precisa para completar um ciclo de 00h a FFh, então, o número de interrupções multiplicado por esse intervalo de tempo, dá-nos o tempo total decorrido. Na rotina de interrupção uma variável guardada na memória RAM vai sendo incrementada, o valor dessa variável multiplicado pelo tempo que o contador precisa para um ciclo completo de contagem, vai dar o tempo gasto. Esta interrupção pode ser habilitada ou inibida, pondo a '1' ou a '0' o bit TOIE (INTCON<5>).

7.6.3 - Interrupção por variação nos pinos 4, 5, 6 e 7 do porto B

Uma variação em 4 bits de entrada do Porto B (bits 4 a 7), põe a '1' o bit RBIF (INTCON<0>). A interrupção ocorre, portanto, quando os níveis lógicos em RB7, RB6, RB5 e RB4 do porto B, mudam do valor lógico '1' para o valor lógico '0' ou vice-versa. Para que estes pinos detectem as variações, eles devem ser definidos como entradas. Se qualquer deles for definido como saída, nenhuma interrupção será gerada quando surgir uma variação do nível lógico. Se estes pinos forem definidos como entradas, o seu valor actual é comparado com o valor anterior, que foi guardado quando se fez a leitura anterior do porto B. Esta interrupção pode ser habilitada/inibida pondo a '1' ou a '0', o bit RBIE do registo INTCON.

7.6.4 - Interrupção por fim de escrita na EEPROM

Esta interrupção é apenas de natureza prática. Como escrever num endereço da EEPROM leva cerca de 10ms (o que representa muito tempo quando se fala de um microcontrolador), não é recomendável que se deixe o microcontrolador um grande intervalo de tempo sem fazer nada, à espera do fim da operação da escrita. Assim, dispomos de um mecanismo de interrupção que permite ao microcontrolador continuar a executar o programa principal, enquanto, em simultâneo, procede à escrita na EEPROM. Quando esta operação de escrita

se completa, uma interrupção informa o microcontrolador deste facto. O bit EEIF, através do qual esta informação é dada, pertence ao registo EECON1. A ocorrência desta interrupção pode ser impedida, pondo a '0' o bit EEIE do registo INTCON.

7.6.5 - Iniciação da interrupção

Para que num microcontrolador se possa usar um mecanismo de interrupção, é preciso proceder a algumas tarefas preliminares. Estes procedimentos são designados resumidamente por "iniciação". Na iniciação, nós estabelecemos a que interrupções deve o microcontrolador responder e as que deve ignorar. Se não pusermos a '1' o bit que permite uma certa interrupção, o programa vai ignorar a correspondente subrotina de interrupção. Por este meio, nós podemos controlar a ocorrência das interrupções, o que é muito útil.

```
clrf  INTCON              ;  todas as interrupções impedidas
movlw  B'00010000'        ;  só autorizada a interrupção externa
movwf INTCON
bsf  INTCON, GIE          ;  permitida a ocorrência de interrupçõses
```

O exemplo de cima, mostra a iniciação da interrupção externa no pino RB0 de um microcontrolador. No sítio em que vemos '1', isso significa que essa interrupção está habilitada. A ocorrência de outras interrupções não é permitida, e todas as interrupções em conjunto estão mascaradas até que o bit GIE seja posto a '1'.

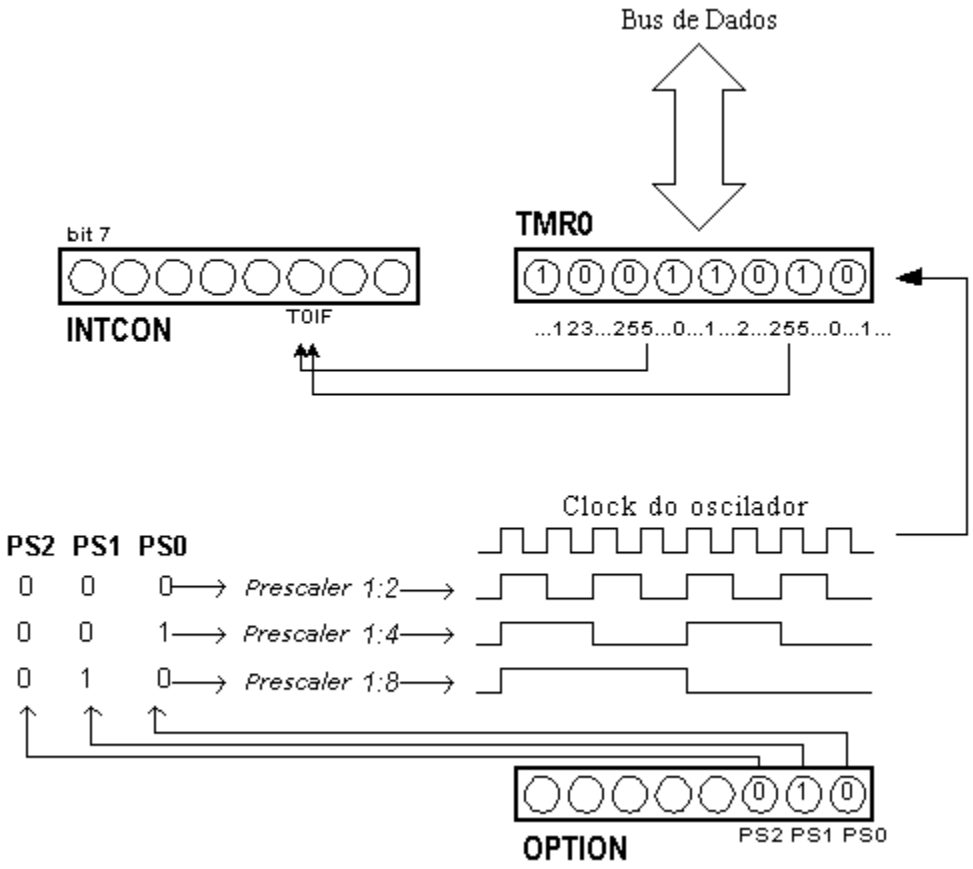
O exemplo que se segue, ilustra uma maneira típica de lidar com as interrupções. O PIC16F84 tem somente um endereço para a rotina de interrupção. Isto significa que, primeiro, é necessário identificar qual a origem da interrupção (se mais que uma fonte de interrupção estiver habilitada), e a seguir deve executar-se apenas a parte da subrotina que se refere à interrupção em causa.

```
org ISR_ADDR            ;ISR_ADDR é o endereço da rotina de interrupção
btfsc INTCON, GIE       ;bit GIE desligado ?
goto ISR_ADR            ;não, voltar ao princípio
PUSH                    ;guardar os conteúdos dos registos importantes
btfsc INTCON, RBIF      ;variação nos pinos 4, 5, 6 e 7 do porto B?
goto ISR_PORTB          ;saltar para a secção correspondente
btfsc INTCON,INTF       ;ocorreu uma interrupção externa em RB0 ?
goto ISR_RB0            ;saltar p/ esse local
btfsc INTCON, TOIF      ;o temporizador TMR0 transbordou ?
goto ISR_TMR0           ;saltar p/ essa secção
BANK1                   ;Banco 1 p/ aceder a EECON1
Btfsc EECON1, EEIF      ;escrita na EEPROM completa?
goto ISR_EEPROM         ;saltar para o endereço correspondente
BANK0                   ;Banco 0

ISR_PORTB
      :                 ;parte do código processado por uma
                        ;interrupção?
      :
      goto END_ISR      ;saltar para a saída da interrupção
ISR_RB0
      :                 ;parte de código processado pela interrupção?
      :
      goto END_ISR      ;saltar para a saída da interrupção
ISR_TMR0
      :                 ;parte de código processado pela interrupção?
      :
      goto END_ISR      ;saltar para a saída da interrupção
ISR_EEPROM
      :                 ;parte de código processado pela interrupção?
      :
      goto END_ISR      ;saltar para a saída da interrupção
END_ISR                 ;
      POP               ;recuperar os conteúdos dos
                        ;registos importantes
      RETFIE            ;regressar e pôr o bit GIE a '1'
```

O regresso de uma rotina de interrupção pode efectuar-se com as instruções RETURN, RETLW e RETFIE. Recomenda-se que seja usada a instrução RETFIE porque, essa instrução é a única que automaticamente põe a '1' o bit GIE, permitindo assim que novas interrupções possam ocorrer.

7.7 Temporizador TMR0

Os temporizadores são normalmente as partes mais complicadas de um microcontrolador, assim, é necessário gastar mais tempo a explicá-los. Servindo-nos deles, é possível relacionar uma dimensão real que é o tempo, com uma variável que representa o estado de um temporizador dentro de um microcontrolador. Físicamente, o temporizador é um registo cujo valor está continuamente a ser incrementado até 255, chegado a este número, ele começa outra vez de novo: 0, 1, 2, 3, 4, ...,255, 0,1, 2, 3,..., etc.

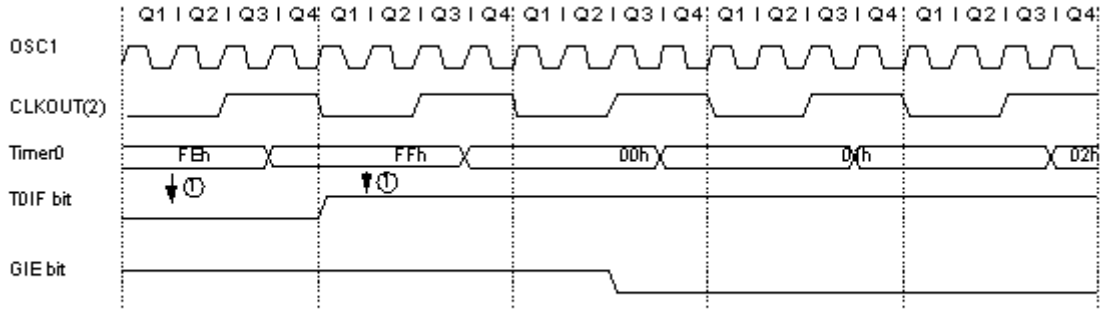


Relação entre o temporizador TMR0 e o prescaler

O incremento do temporizador é feito em simultâneo com tudo o que o microcontrolador faz. Compete ao programador arranjar maneira de tirar partido desta característica. Uma das maneiras é incrementar uma variável sempre que o microcontrolador transvaza (passa de 255

para 0). Se soubermos de quanto tempo um temporizador precisa para perfazer uma contagem completa (de 0 a 255), então, se multiplicarmos o valor da variável por esse tempo, nós obteremos o tempo total decorrido.

O PIC16F84, possui um temporizador de 8 bits. O número de bits determina a quantidade de valores diferentes que a contagem pode assumir, antes de voltar novamente para zero. No caso de um temporizador de 8 bits esse valor é 256. Um esquema simplificado da relação entre um temporizador e um prescaler está representado no diagrama anterior. Prescaler é a designação para a parte do microcontrolador que divide a frequência de oscilação do clock antes que os respectivos impulsos possam incrementar o temporizador. O número pelo qual a frequência de clock é dividida, está definido nos três primeiros bits do registo OPTION. O maior divisor possível é 256. Neste caso, significa que só após 256 impulsos de clock é que o conteúdo do temporizador é incrementado de uma unidade. Isto permite-nos medir grandes intervalos de tempo.

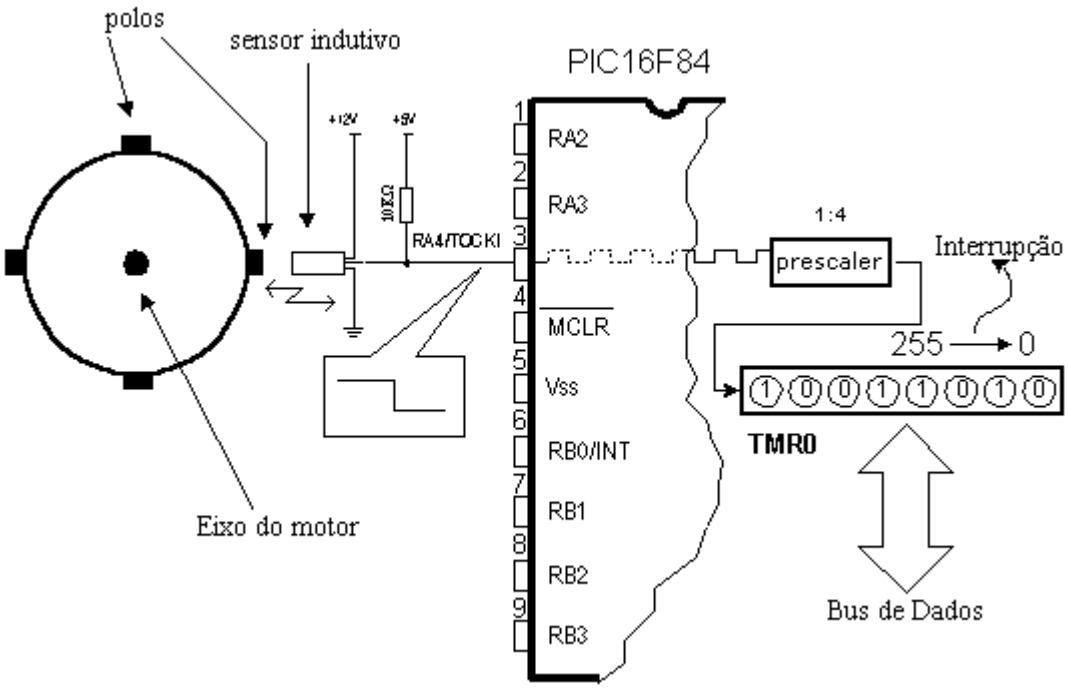


Notas:1. O estado da flag de interrupção TOIF é verificado em todos os Q1
2. CLKOUT existe só no modo de oscilador RC

Diagrama temporal de uma interrupção causada pelo temporizador TMR0

Quando a contagem ultrapassa 255, o temporizador volta de novo a zero e começa um novo ciclo de contagem até 255. Sempre que ocorre uma transição de 255 para 0, o bit TOIF do registo INTCON é posto a '1'. Se as interrupções estiverem habilitadas, é possível tirar partido das interrupções geradas e da rotina de serviço de interrupção. Cabe ao programador voltar a pôr a '0' o bit TOIF na rotina de interrupção, para que uma nova interrupção possa ser detectada. Além do oscilador de clock do microcontrolador, o conteúdo do temporizador pode também ser incrementado através de um clock externo ligado ao pino RA4/TOCKI. A escolha entre uma destas opções é feita no bit TOCS, pertencente ao registo OPTION. Se

for seleccionado o clock externo, é possível definir o bordo activo do sinal (ascendente ou descendente), que vai incrementar o valor do temporizador.



Utilização do temporizador TMR0 na determinação do número de rotações completas do eixo de um motor

Na prática, um exemplo típico que é resolvido através de um clock externo e um temporizador, é a contagem do número de rotações completas do eixo de uma máquina, como por exemplo um enrolador de espiras para transformadores. Vamos considerar que o 'rotor' do motor do enrolador, contém quatro polos ou saliências. Vamos colocar o sensor indutivo à distância de 5mm do topo da saliência. O sensor indutivo irá gerar um impulso descendente sempre que a saliência se encontre alinhada com a cabeça do sensor. Cada sinal vai representar um quarto de uma rotação completa e, a soma de todas as rotações completas, ficará registado no temporizador TMR0. O programa pode ler facilmente estes dados do temporizador através do bus de dados.

O exemplo seguinte mostra como iniciar o temporizador para contar os impulsos descendentes provenientes de uma fonte de clock externa com um prescaler 1:4.

```
      clrf TMR0          ;TMR0=0
      clrf INTCON        ;Interrupções inibidas e TOIF = 0
      bsf STATUS,RP0     ;Banco 1
      movlw B'00110001'  ;prescaler 1:4; interrupção externa no bordo descendente
                         ;fonte de clock externa e resistências de pull-up do
                         ;porto B, activadas.

      movwf OPTION_REG ;OPTION_REG <- W
TO_OVFL
      btfss INTCON, TOIF      ;testando a flag de transbordo
      goto TO_OVFL            ;a interrupção não ocorreu ainda, esperar
;
;(Parte do programa que processa os dados, consoante o número de voltas)
;
goto TO_OVFL                  ;esperar que torne a transbordar
```

O mesmo exemplo pode ser implementado através de uma interrupção do modo seguinte:

```
      org 0x00           ;endereço de reset
      goto Start         ;início do programa

      org 0x04           ;endereço de interrupção
      goto TO_OVFL       ;início da rotina de interrupção

Start clrf TMR0          ;TMR0=0
      clrf INTCON        ;Intrrupções inibidas e TOIF=0
      bsf STATUS,RP0     ;Banco 1
      movlw B'00110001'  ;prescaler 1:4; interrupção externa no bordo descendente
                         ;fonte de clock externa e resistências de pull-up do
                         ;porto B, activadas.
      movwf OPTION_REG   ;OPTION_REG <- W
      bsf INTCON,TOIE    ;interrupção ao transbordar habilitada
      bsf INTCON,GIE     ;interrupções permitidas
TO_OVFL

;(Parte do programa que processa os dados, consoante o número de voltas)
;

bcf INTCON,TOIF    ;a flag de interrupção é limpa, para que a próxima interrupção
                   ;possa ser detectada.
retfie             ;regresso da rotina de interrupção
```

O prescaler tanto pode ser atribuído ao temporizador TMR0, como ao watchdog. O watchdog é um mecanismo que o microcontrolador usa para se defender contra "estouros" do programa. Como qualquer circuito eléctrico, também os microcontroladores podem ter uma falha ou algum percalço no seu funcionamento. Infelizmente, o microcontrolador também pode ter problemas com o seu programa. Quando isto acontece, o microcontrolador pára de trabalhar e mantém-se nesse estado até que alguém faça o reset. Por causa disto, foi introduzido o mecanismo de watchdog (cão de guarda). Depois de um

certo período de tempo, o watchdog faz o reset do microcontrolador (o que realmente acontece, é que o microcontrolador executa o reset de si próprio). O watchdog trabalha na base de um princípio simples: se o seu temporizador transbordar, é feito o reset do microcontrolador e este começa a executar de novo o programa a partir do princípio. Deste modo, o reset poderá ocorrer tanto no caso de funcionamento correcto como no caso de funcionamento incorrecto. O próximo passo é evitar o reset no caso de funcionamento correcto, isso é feito escrevendo zero no registo WDT (instrução CLRWDT) sempre que este está próximo de transbordar. Assim, o programa irá evitar um reset enquanto está a funcionar correctamente. Se ocorrer o "estouro" do programa, este zero não será escrito, haverá transbordo do temporizador WDT e irá ocorrer um reset que vai fazer com que o microcontrolador comece de novo a trabalhar correctamente.

O prescaler pode ser atribuído ao temporizador TMR0, ou ao temporizador do watchdog, isso é feito através do bit PSA no registo OPTION. Fazendo o bit PSA igual a '0', o prescaler é atribuído ao temporizador TMR0. Quando o prescaler é atribuído ao temporizador TMR0, todas as instruções de escrita no registo TMR0 (CLRF TMR0, MOVWF TMR0, BSF TMR0,...) vão limpar o prescaler. Quando o prescaler é atribuído ao temporizador do watchdog, somente a instrução CLRWDT irá limpar o prescaler e o temporizador do watchdog ao mesmo tempo. A mudança do prescaler está completamente sob o controle do programador e pode ser executada enquanto o programa está a correr.

Existe apenas um prescaler com o seu temporizador. Dependendo das necessidades, pode ser atribuído ao temporizador TMR0 ou ao watchdog, mas nunca aos dois em simultâneo.

Registo de Controle OPTION

R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1
$\overline{\text{RBPU}}$ (1)	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0
bit 7							bit 0

Legenda:
R = bit p/ ler **W** = bit p/ escrever
U=Não implementado,ao ler-se dá '0' -n=valor de reset 'ao ligar'

bit 0:2 PS0, PS1, PS2 (bits de selecção do divisor prescaler)

O prescaler e como estes bits afectam o funcionamento do microcontrolador, são abordados na secção que trata de TMR0.

Bits	TRM0	WDT
000	1 : 2	1 : 1
001	1 : 4	1 : 2
010	1 : 8	1 : 4
011	1 : 16	1 : 8
100	1 : 32	1 : 16
101	1 : 64	1 : 32
110	1 : 128	1 : 64
111	1 : 256	1 : 128

bit 3 PSA (bit de Atribuição do Prescaler)

Bit que atribui o prescaler ou ao temporizador TMR0 ou ao temporizador do watchdog

1 = o prescaler está atribuído ao temporizador do watchdog.

0 = o prescaler está atribuído ao temporizador TMR0.

bit 4 T0SE (selecção de bordo activo em TMR0)

Se o temporizador estiver configurado para contar impulsos externos aplicados ao pino RA4/T0CKI, este bit vai determinar quando a contagem irá incidir sobre os impulsos ascendentes ou descendentes do sinal.

1 = bordo descendente

0 = bordo ascendente

bit 5 T0CS (bit de selecção de fonte de clock para TMR0)

Este pino habilita o contador/temporizador TMR0 a incrementar o seu valor ou com os impulsos do oscilador interno, isto é, a 1/4 das oscilações do clock do oscilador, ou através de impulsos externos aplicados ao pino

RA4/T0CKI.

1 = impulsos externos

0 = 1/4 do clock interno

Bit 6 INTEDG (bit de selecção do bordo activo da interrupção)

Se a ocorrência de interrupções estiver habilitada, este bit vai determinar qual o bordo em que a interrupção no pino RB0/INT vai ocorrer.

1 = bordo ascendente

0 = bordo descendente

Bit 7 RBPU (Bit de habilitação dos pull-up no porto B)

Este bit introduz ou retira as resistências de pull-up internas do porto B.

1 = resistências de 'pull-up' inseridas

0 = resistências de 'pull-up' retiradas

7.8 Memória de dados EEPROM

O PIC16F84 tem 64 bytes de localizações de memória EEPROM, correspondentes aos endereços de 00h a 63h e onde podemos ler e escrever. A característica mais importante desta memória é de não perder o seu conteúdo quando a alimentação é desligada. Na prática, isso significa que o que lá foi escrito permanece no microcontrolador, mesmo quando a alimentação é desligada. Sem alimentação, estes dados permanecem no microcontrolador durante mais de 40 anos (especificações do fabricante do microcontrolador PIC16F84), além disso, esta memória suporta até 10000 operações de escrita.

Na prática, a memória EEPROM é usada para guardar dados importantes ou alguns parâmetros de processamento.

Um parâmetro deste tipo, é uma dada temperatura, atribuída quando ajustamos um regulador de temperatura para um processo. Se esse valor se perder, seria necessário reintroduzi-lo sempre que houvesse uma falha na alimentação. Como isto é impraticável (e

mesmo perigoso), os fabricantes de microcontroladores começaram a instalar nestes uma pequena quantidade de memória EEPROM.

A memória EEPROM é colocada num espaço de memória especial e pode ser acedida através de registos especiais. Estes registos são:

EEDATA no endereço 08h, que contém o dado lido ou aquele que se quer escrever.

EEADR no endereço 09h, que contém o endereço do local da EEPROM que vai ser acedido

EECON1 no endereço 88h, que contém os bits de controlo.

EECON2 no endereço 89h. Este registo não existe fisicamente e serve para proteger a EEPROM de uma escrita acidental.

O registo EECON1 ocupa o endereço 88h e é um registo de controlo com cinco bits implementados.

Os bits 5, 6 e 7 não são usados e, se forem lidos, são sempre iguais a zero.

Os bits do registo EECON1, devem ser interpretados do modo que se segue.

Registo EECON1

U-0	U-0	U-0	R/W-1	R/W-1	R/W-x	R/S-0	R/S-x
—	—	—	EEIF (1)	WRERR	WREN	WR	RD
bit 7							bit 0

Legenda:
R = bit p/ ler **W** = bit p/ escrever
U = Não implementado, ao ler-se dá '0' -n=valor de reset 'ao ligar'

bit 0 RD (bit de controle de leitura)

Ao pôr este bit a '1', tem início a transferência do dado do endereço definido em EEADR para o registo EEDATA. Como o tempo não é essencial, tanto na leitura como na escrita, o dado de EEDATA pode já ser usado na instrução seguinte.

1 = inicia a leitura

0 = não inicia a leitura

Bit 1 WR (bit de controle de escrita)

Pôr este bit a '1' faz iniciar-se a escrita do dado a partir do registo EEDATA para o endereço especificado no registo EEADR.

1 = inicia a escrita

0 = não inicia a escrita

bit 2 WREN (bit de habilitação de escrita na EEPROM). Permite a escrita na EEPROM.

Se este bit não estiver a um, o microcontrolador não permite a escrita na EEPROM.

1 = a escrita é permitida

0 = não se pode escrever

bit 3 WRERR (Erro de escrita na EEPROM). Erro durante a escrita na EEPROM

Este bit é posto a '1' só em casos em que a escrita na EEPROM tenha sido interrompida por um sinal de reset ou por um transbordo no temporizador do watchdog (no caso de este estar activo).

1 = ocorreu um erro

0 = não houve erros

bit 4 EEIF (bit de interrupção por operação de escrita na EEPROM completa) Bit usado para informar que a escrita do dadoo na EEPROM, terminou.

Quando a escrita tiver terminado, este bit é automaticamente posto a '1'. O programador tem que repôr a '0' o bit EEIF no seu programa, para que possa detectar o fim de uma nova operação de escrita.

1 = escrita terminada

0 = a escrita ainda não terminou ou não começou.

Lendo a Memória EEPROM

Pondo a '1' o bit RD inicia-se a transferência do dado do endereço guardado no registo EEADR para o registo EEDATA. Como para ler os dados não é preciso tanto tempo como a escrevê-los, os dados extraídos do registo EEDATA podem já ser usados na instrução seguinte.

Uma porção de um programa que leia um dado da EEPROM, pode ser semelhante ao seguinte:

```
bcf   STATUS, RP0       ;banco 0, porque EEADR está em 09h
movlw 0x00              ;endereço do local a ler
movwf EEADR             ;endereço transferido para EEADR
bsf   STATUS, RP0       ;banco 1, porque EECON1 está em 88h
bsf   EECON1, RD        ;ler da EEPROM
bcf   STATUS, RP0       ;banco 0, porque EEDATA está em 08h
movf  EEDATA, W         ;W <-- EEDATA
```

Depois da última instrução do programa, o conteúdo do endereço 0 da EEPROM pode ser encontrado no registo de trabalho w.

Escrevendo na Memória EEPROM

Para escrever dados num local da EEPROM, o programador tem primeiro que endereçar o registo EEADR e introduzir a palavra de dados no registo EEDATA. A seguir, deve colocar-se o bit WR a '1', o que faz desencadear o processo. O bit WR deverá ser posto a '0' e o bit EEIF será posto a '1' a seguir à operação de escrita, o que pode ser usado no processamento de interrupções. Os valores 55h e AAh são as primeira e segunda chaves que tornam impossível que ocorra uma escrita acidental na EEPROM. Estes dois valores são escritos em EECON2 que serve apenas para isto, ou seja, para receber estes dois valores e assim prevenir contra uma escrita acidental na memória EEPROM. As linhas do programa marcadas como 1, 2, 3 e 4 têm que ser executadas por esta ordem em intervalos de tempo certos. Portanto, é muito importante desactivar as interrupções que possam interferir com a temporização necessária para executar estas instruções. Depois da operação de escrita, as interrupções podem, finalmente, ser de novo habilitadas.

Exemplo da porção de programa que escreve a palavra 0xEE no primeiro endereço da memória EEPROM:

```
        bcf   STATUS, RP0       ;banco 0, porque EEADR está em 09h
        movlw 0x00              ;endereço do local de memória
                                ;em que se quer escrever
        movwf EEADR             ;endereço transferido para
                                ;EEADR
        movlw 0xEE              ;escrever o valor 0xEE
        movwf EEDATA            ;dado no registo EEDATA
        bsf STATUS, RP0         ;banco 1
        bcf INTCON, GIE         ;todas as interrupções impedidas
        bsf EECON1, WREN        ;permissão de escrita
        movlw 55h
1)      movwf EECON2            ;1ª chave    55h --> EECON2
2)      movlw AAh
3)      movwf EECON2            ;2ª chave     AAh --> EECON2
4)      bsf EECON1,WR           ;iniciar a escrita
        bsf INTCON, GIE         ;interrupções habilitadas
```

Recomenda-se que WREN esteja sempre inactivo, excepto quando se está a escrever uma palavra de dados na EEPROM, deste modo, a possibilidade de uma escrita acidental é mínima.
Todas as operações de escrita na EEPROM 'limpam' automaticamente o local de memória, antes de escrever de novo nele !