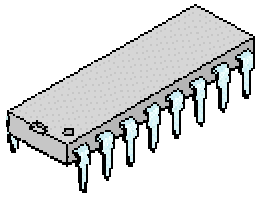


MICROCONTROLADORES

O curso todo é baseado no PIC 16F84, devido as suas facilidades de desenvolvimento, por não necessitar de apagadores de EPROM, mas outros modelos (16C711 com conversores A/D e 12C508 com apenas 8 pinos) também são apresentados de forma resumida, com exemplos.

Avisos Importantes:

1) Toda a documentação do software Mplab requer do usuário proficiência em inglês, e não faz parte deste curso, não responsabilizando o autor por eventuais problemas e dificuldades individuais devido à deficiência a língua inglesa.
2) O Autor não se responsabiliza por eventuais problemas de funcionamento e montagem dos circuitos aqui apresentados, sejam por erro do leitor ou qualquer outro, pois muitas vezes um circuito não funciona porque está além da capacidade técnica do montador naquele momento.

ÍNDICE

1-Microcontroladores e suas aplicações.

Praticamente estamos rodeados de aparelhos que possuem dentro de si um microcontrolador, e nem mesmo temes consciência disto.

Os vídeos cassetes, celulares, agendas eletrônicas, vários brinquedos, alarmes de automóvel, são apenas alguns exemplos mais comuns.

Basicamente o microcontrolador (anteriormente chamado de microcomputador de um só chip) é um componente que possui todos os periféricos dos microprocessadores comuns embutidos em uma só pastilha, facilitando assim o desenvolvimento de sistemas pequenos e baratos, embora complexo e sofisticados.

Costuma apresentas em uma única pastilha memórias de dados e programa, canal serias, temporizadores, interfaces para displays, memória EEPROM, PWM, e muito mais, dependendo do modelo.

O diagrama de blocos simplificado pode ser visto na figura 1.1

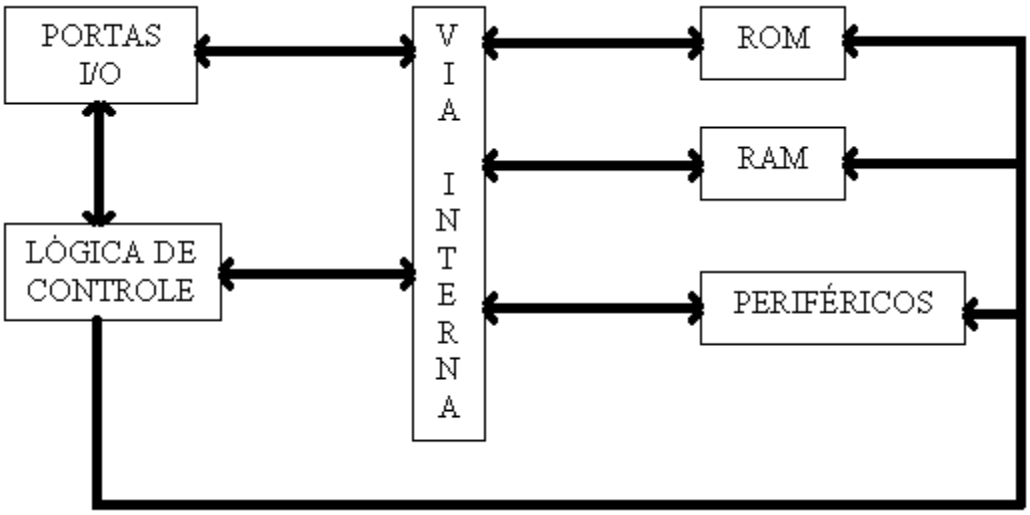


Figura 1.1-diagrama de blocos básico de um microcontrolador.

Devido à facilidade de estudo permitida pala fácil regravação da memória de programa do PIC16F84 da Microchip, e também pelo fato do mesmo possuir todos os periféricos dos demais membros da família PIC, o mesmo foi a escolhido como objeto de estudo desta apostila.

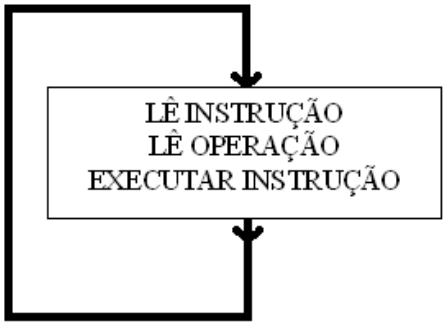
Além deste veremos alguns detalhes do PIC 17C711, que possui A/D interno e do 12C508, revolucionário microcontrolador com apenas 8 pinos.

Apresentamos ainda um resumo das famílias PIC 12C508, 16C5X e PIC 16CXXX.

2-Arquitetura Harvard X Von Neumann

A maioria dos microprocessadores comuns e vários microcontroladores existentes no mercador tem sua estrutura interna de memória de dados e programa baseados na conhecida arquitetura Von Neumann, que prevê um único bus (via) de comunicação entre memórias e CPU.

Basicamente podemos dizer que a seqüência de trabalho é:



Os PIC´s utilizam uma arquitetura diferente conhecida como Harvard, que prevê várias vias de comunicação entre CPU e periféricos, permitindo a realização de várias operações simultaneamente, que implica em aumento considerável na velocidade de execução e permiti ainda que a memória de dados e memória de programas tenham tamanhos diferentes.

No PIC 16F84, por exemplo os dados são de 8 bits e as instruções são de 14 bits.

Esta facilidade permite ainda que numa única palavra de 14 bits tenhamos o código da instrução, onde a mesma vai atuar e o eventual operando ou dado, se houver.

Criou-se então uma terminologia chamada RISC (reduced Instructions Set Computer-Computador com Set de Instruções Reduzido) que faz com que existam instruções (mais ou menos 35, dependendo do modelo) enquanto alguns microprocessadores tradicionais chegam a ter mais de 100 instruções.

Este set reduzido de instruções facilita muito o aprendizado.

Exemplo: somar 12 ao registro 24.

A)microprocessador comum:

3 bytes somar é a operação
12 é o operando
24 é o local

B)microprocessador RISC:

1 palavra somar + 12(operando) +24 (local)
de 14 bits (tudo em uma única linha)

Como as maiorias das instruções dos microprocessadores comuns usam 2 bytes (existem instruções de 1 a 3 bytes também), vemos que o código dos PIC´s já tem basicamente a metade do tamanho, isto é, 4 K no PIC são 4 K de instruções, enquanto 4K nos demais são aproximadamente 2K de instruções.

Outra vantagem da arquitetura Harvard esta no fato no fato de que enquanto uma instrução esta sendo executa e valores lidos ou escritos na memória ou I/O pela via apropriada, outra instrução já esta sendo carregada pela via da memória de programa.

Apenas como comparação , um 8051 padrão, rodando a 12 MHz (sua velocidade máxima típica), executa a maioria das instruções em 1 us, enquanto para executar em 1us o PIC precisa de apenas 4 MHz, ou seja, com 12 MHz o PIC será 3 vezes mais rápido.

Obs: nem todos os modelos de PIC´s chegam a 12 MHz.

3-Os Microcontroladores PIC

toda a família PIC de microcontroladores possui as características descritas anteriormente, como palavras de 14 bits (existem versões com 12 e de 16 bits também), tecnologia RISC, alta velocidade e facilidade de aprendizado.

As memórias de programa vão desde 512 palavras até dispositivos mais sofisticados co 16K repletos de periféricos com vários canais seriais, PWM, comparadores, vários timers, entro outros.

Mo capítulo 23 apresentaremos um resumo da família PIC para apreciação do leitor.

Nesta apostila apenas o 16F84 será estudado em profundidade, embora tenhamos citações e exemplos com 16C711 e o 12C508.

O 16F84 é um membro da família PIC de microcontroladores com 1K de programa em memória FLASH, possui 68 bytes de RAM para uso geral, 64 bytes de EEPROM para dados, 4 fontes de interrupção, 1 time de 8 bits, 1 WATCH DOG, 1 prescaler (divisor de freqüências) de 8 bits e 13 bits de I/O individualmente endereçáveis.

Devido à tecnologia FLASH, este chip é apagado e regravado automaticamente pelos gravadores, dispensando as lâmpadas ultravioletas comuns nos desenvolvimentos com EPROM comum.

Como todos os membros da família Pic de microcontroladores tem os mesmos periféricos básicos, o leitor não sentirá dificuldade na migração de um modelo para outro, devendo apenas estudar os detalhes de cada um conforme suas necessidades.

3.1-o clock e os ciclos de máquina.

Nos microcontroladores Pic o sinal de clock é dividido internamente por quatro gerando as fases conhecidas como Q1, Q2, Q3 e Q4.

Cada ciclo de instrução é composto das 4 fases, de forma que cada ciclo demanda então de um tempo.

$$t = \frac{1}{(Fosc/4)}$$

Se o clock é de 4MHz (T=250 ns), cada ciclo tem então a duração de 1 us.

Obs: Para facilitar, em todos os exemplos da apostila será usado este calor como referência do clock e tempo, exceto se indicado o contrário.

Como já vimos, existem modelos que chegam a rodar com o clock de até 20MHz, com tempo de ciclo de 200 ns.

A maioria das instruções é executada em apenas m ciclo, pois durante as fases Q1 e Q4 o hardware interno executa uma instrução, outra já foi lida na memória e aguarda o próximo ciclo.

As execuções são as instruções que alteram o PC (program counter), que levam dois ciclos, pois com a mudança do endereço a instrução já trazida da memória é descartada e outra , no novo PC, deverá ser lida.

4-O PIC 16F84

4.1-introdução

O pic 16F84 é um microcontrolador que pode operar de DC até 10MHz (ciclo de instrução de 400 ns) e devido as suas características de projeto funciona com um mínimo de componentes externos.

Possui ainda vários circuitos auxiliares internamente, que aumentam bastante sua flexibilidade e performance.

Suas principais características são:

- 1K(1024) palavras de 14 bits para programa
- 68 bytes de RAM para uso geral
- 64 bytes de EEPROM para dados
- Stack com 8 níveis
- Apenas 35 instruções
- 15 registros específicos em RAM para controle do chip e seus periféricos
- timer de 8 bits com opção de prescaler de 8 bits
- 13 pinos que podem ser configurados individualmente como entrada ou saída
- alta capacidade de corrente nos pinos de saída (podem acender um led diretamente)
- capacidade de gerenciar interrupções externas (até 5 entradas), do timer e da EEPROM
- WATCH DOG para recuperação e Reset em caso de travas no software
- Memória de programa protegida contra cópia
- Modo sleep para economia de energia
- Várias opções de osciladores

Outras características e maiores detalhes serão estudados nos capítulos correspondentes

4.2-Pinagem e características elétricas básicas

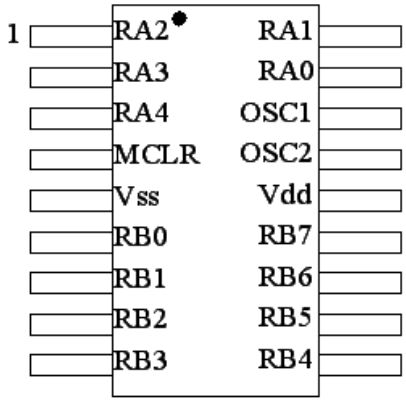


Figura 4.1-Pinos do 16F84 no encapsulamento DIP.

Faixa de tensão de alimentação: 2,0V a 6,0V (típica 5,0V)

Consumo de corrente:1)<2 mA a 5V, 4MHz
2)60 uA a 2V, 32KHz
3)26 uA a 2V em standby

Descrição dos pinos agrupados por blocos:

Pino 14	Vdd	Tensão de alimentação
Pino 5	Vss	Referência de Terra
Pino 17	RA0	Porta A, bit 0. Entrada ou saída digital
Pino 18	RA1	Porta A, bit 1. Entrada ou saída digital
Pino 1	RA2	Porta A, bit 2. Entrada ou saída digital
Pino 2	RA3	Porta A, bit 3. Entrada ou saída digital
Pino 3	RA4/TOCKI	Porta A, bit 4. Entrada ou saída digital,entrada TIMER 0
Pino 4	MCLR	Entrada de reset em nível 0
Pino 16	OSC1/CLKIN	Cristal de clock externo
Pino 15	OSC2/CLKOUT	Cristal ou saída Fosc/4 em modo RC
Pino 6	RB0/INT	Porta B, bit 0. Entrada ou saída digital, ou interrupção externa
Pino 7	RB1	Porta B, bit 1. Entrada ou saída digital
Pino 8	RB2	Porta B, bit 2. Entrada ou saída digital
Pino 9	RB3	Porta B, bit 3. Entrada ou saída digital
Pino 10	RB4	Porta B, bit 4. Entrada ou saída digital, interrupção nas mudanças de estados
Pino 11	RB5	Porta B, bit 5. Entrada ou saída digital, interrupção nas mudanças de estados
Pino 12	RB6	Porta B, bit 6. Entrada ou saída digital, interrupção nas mudanças de estados
Pino 13	RB7	Porta B, bit 7. Entrada ou saída digital, interrupção nas mudanças de estados

4.3-Nomenclatura dos pinos

Sempre que um pino for ativo em 0 ou sinalizar um evento sendo zerado, o mesmo será indicado por uma barra invertida " \ " após o mesmo.

Exemplo: **MCLR\.**

4.4-Funcionamento dos registros e memória

Os membros da família 16CXXX podem acessar tanto direta como indiretamente qualquer posição de memória RAM ou dos registros internos, pois estão todos mapeados no mesmo bloco de memória (ver item 5.2).

Qualquer operação pode ser feita com qualquer registro (de dados ou de controle).

As operações lógicas e aritméticas são realizadas por um bloco chamado de ULA (unidade lógica e aritmética) que possui um registro próprio chamado W(working register-o popular acumulador), que não esta presente na RAM e não é acessado por endereçamento (é um registro interno da ULA).

A ULA é de 8 bits que permite realizar somas, subtrações, deslocamentos (shifts) e operações lógicas.

Os bits de sinalização, ou flags, chamados Z (zero), C(carry) e DC(Digit Carry) refletem os resultados de várias operações realizadas na ULA.

5-Organização de memória no 16F84

A partir deste capítulo estudaremos em detalhes o PIC 16F84, iniciando pelas memórias internas ao mesmo.

Sempre que se fizer necessário, um pequeno programa assembler, devidamente comentado , será incluído. As instruções e detalhes de programação serão estudados no capítulo 18.

A partir deste ponto vamos ainda passar a usar as demonstrações originais das portas, a saber.

Porta A será chamada PORTA de port A.
Porta B será chamada PORTB de port B.

5.1-Memória de Programa

Como já vimos, a família PIC possui em sua arquitetura segmentos de memória separados para programas e dados, inclusive com palavras de tamanhos diferentes.

Como cada memória tem uma via separada no hardware interno, os dois blocos podem ser acessados simultaneamente pelo programa em um mesmo ciclo de máquina, que como já vimos permite instruções de uma só palavra e execução em um único ciclo.

O PC (contador de programa) da família 16CXXXtem 13 bits, permitindo memória de programa de 8K.

O PIC 16F84 possui apenas 1K implementado (de 00H até 3FFH). Quaisquer referências a outras posições de memória serão deslocadas para este bloco de 1K.

Exemplo: as posições 72H, 472H, C72H e outras somadas 400H referem-se sempre a posição original 72H.

No Reset o PC aponta para o endereço 000H e ao atender uma interrupção o PC é carregado com o endereço 004H.

5.1.1-Mapa da memória de programa

Na figura 5.1 temos um mapa da memória de programa . Como vemos entre o endereço de reset (000H) e o início da interrupção (004H) tem apenas 4 bytes. Se interrupções forem usadas o usuário deve providenciar nos endereços 000H, 001H, 002H ou 003H um desvio para outro local da memória.

Exemplo:

```
000H   goto    100H          ;ao reset desvia para o endereço 100H
;
;
;
004H   .....                 ;início da interrupção
;
;
;
```

```
100H   btfsc    RA0          ;programa principal
```

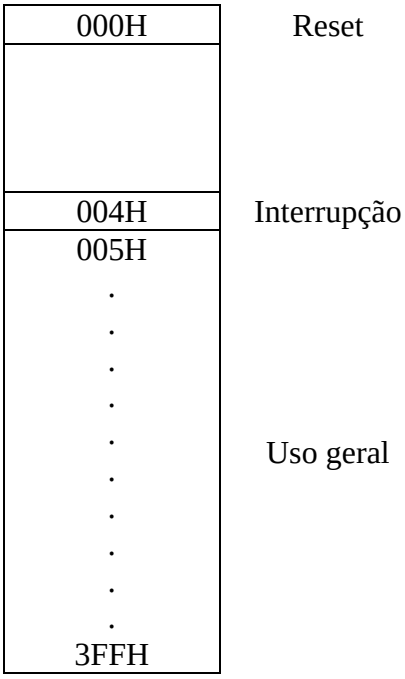


Figura 5.1-Mapa de memória de programa do PIC 16F84

5.1.2-STACK

O stack foi projetado apenas para salvar o Pc para o qual retornará sempre que uma interrupção ou sub-rotina for chamada. No stack o chip armazena apenas o PC de retorno, não salvando as condições da máquina, ficando estes por conta do usuário.

O stack do PIC 16F84 tem 8 níveis de profundidade, ou seja, apenas 8 sub-rotinas e interrupções podem estar simultaneamente em execução.

Cabe ao usuário gerenciar o mesmo.

5.2-Memória de dados e registro de controle

A memória de dados e memória de registros de controle nada mais são do que um grupo de memória RAM organizadas em dois bancos de registradores (banco 0 e banco 1).

Os flags RP1 e RP0 selecionam os bancos de registradores. Com 2 bits podemos selecionar até 4 bancos, mas apenas os dois primeiros estão disponíveis no PIC 16F84. Desta forma RP1 deve ser mantido sempre em 0.

Os registros de controle são conhecidos por SFR (special function registers) e serão assim denominados daqui a diante.

Os SFR e a memória RAM estão organizados no PIC 16F84 conforme a figura 5.2.

É importante lembrar que a memória RAM não retém seus valores em caso de falta de alimentação.

INDF	00H
TMR0	01H
PCL	02H
STATUS	03H
FSR	04H
PORTA	05H
PORTB	06H
-	07H
EEDATA	08H
EEADR	09H
PCLATH	0AH
INTCON	0BH
68 BYTES DE USO GERAL	OCH . . 4FH
Banco 0	Endereço

INDF	80H
OPTION	81H
PCL	82H
STATUS	83H
FSR	84H
TRISA	85H
TRISB	86H
-	87H
EECON1	88H
EECON2	89H
PCLATH	8AH
INTCON	8BH
MAPEADO NO BANCO 0	8CH . . CFH
Banco 1	Endereço

Figura 5.2-memória de dados com SFR e RAM de uso geral no PIC 16F84.

5.2.1-Memória RAM de uso geral

A área de memória RAM de uso geral, que deste ponto em diante será chamada apenas de RAM, vai do endereço 0CH até o endereço 4FH, totalizando 68 bytes disponíveis para o usuário.

Conforme a figura 5.2, os endereços 0CH até 4FH estão espelhados nos endereços 8CH até CFH, isto é, o acesso ao endereço 21H é o mesmo que acessar o endereço A1H.

Nesta área com 68 bytes o usuário deve alocar todas as variáveis de seu programa, bem como salvar informações úteis quando chamar sub-rotinas ou atender interrupções.

5.2.2-SFR

Os SFR são usados pela CPU e/ou periféricos para controlar o funcionamento do chip conforme o desejado.

Pode ser dividido em dois tipos, controle/uso da CPU e controle/uso dos periféricos.

No item 5.3 estudaremos os SFR dedicados a CPU. Os demais, utilizados pelos periféricos, serão estudados nos capítulos dedicados aos mesmos.

BANCO 0			
00H	INDF	Endereçamento indireto	Item 5.5
01H	TMR0	Registro de contagem do TIMER 0	Cap. 8
02H	PCL	Parte baixo do PC	
03H	STATUS	Registro STATUS	Item 5.3.1
04H	FSR	Ponteiro para endereçamento indireto	Item 5.5
05H	PORTA	Registro dos pinos do PORTA	Item 7.1
06H	PORTB	Registro dos pinos do PORTB	Item 7.2
07H	-	Não implementado	
08H	EEDATA	Dado lido/gravado na EEPROM	Item 6.2.2
09H	EEADR	Endereço para ler/gravar na EEPROM	Item 6.2.1
0AH	PCLATH	Parte alta do PC	
0BH	INTCON	Registro INTCON	Item 5.3.3

BANCO 1			
80H	INDF		
81H	OPTION	Registro OPTION	Item 5.3.2
82H	PCL		
83H	STATUS		
84H	FSR		
85H	TRISA	Direção dos pinos do PORTA	Item 7.1
86H	TRISB	Direção dos pinos do PORTB	Item 7.2
87H	-	Não implementado	
88H	EECON1	Controle da EEPROM	Item 6.2.3
89H	EECON2	Controle da EEPROM	Item 6.2.4
8AH	PCLATH		
8BH	INTCON		

5.3-Registros de controle da CPU.

São utilizados para controle da CPU, como interrupção, flags da ULA, timer e outros.

São os registros STATUS, OPTION, INTCON, e registros de controle das portas.

Bits indicados por R/W podem ser lidos e escritos.
Bits indicados por R só podem ser lidos.
Bits indicados por ‘u’ tem seus valores inalterados.
Bits indicados por ‘x’ tem seus valores indeterminados.

Estão detalhados na próxima tabela de visualização.

5.3.1-Registro STATUS

Configura banco de registradores, flags da ULA, e outros

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
IRP	**RP1**	**RP0**	**TO**	**PD**	**Z**	**DC**	**C**

Endereço: 03H, 83H valor no reset:00011XXX

Bit 7-RW	**IRP**	Seleciona bancos (para endereçamento indireto).
Bit 6-	**RP1**	(manter RP1 sempre zerado)
Bit 5-RW	**RP0**	Seleciona bancos (no endereçamento direto) 00=banco 0 (00H a 7FH) 01=banco 1 (80H a FFH) 10=banco 2 (100H a 17FH)-não implementado no pic 16F84 11=banco 3 (180H a 1FFH)-não implementado no pic 16F84
Bit 4-R	**TO**	Bit sinalizador de Time-out 1=após power-up, instrução CLRWDT ou SLEEP 0=ocorreu time-out do WATCH DOG
Bit 3-R	**PD**	Bit power down 1=após power-up ou pela instrução CLRWDT
Bit 2-RW	**Z**	Bit sinalizador de zero 1= o registro esta com valor 0 0= o registro não esta com valor de zero
Bit 1-RW	**DC**	Digit carry/Borrow 1=ocorreu um carry-out do 3º para o 4º bit 0=não ocorreu um carry out
Bit 0-RW		Carry/Borrow 1=ocorreu um carry-out do 7º bit do resultado 0=não ocorreu um carry-out

Situação no Reset:Banco 0, bits sinalizadores de time-out e power-down setados, bits da ULA indeterminados.

5.3.2-Registro OPTION

Configura prescaler, timers e outros.

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
RBPU	**INTEDG**	**TOCS**	**TOSE**	**PSA**	**PS2**	**PS1**	**PS0**

Endereço: 81H valor no reset:11111111

Bit 7	**RBPU**	Habilita pull-ups da PORTB 1=PORTB Pull-ups desabilitados 0=PORTB Pull-ups habilitados
Bit 6-RW	**INTEDG**	Como aceitará a interrupção externa int? 1=na subida do sinal no pino RBO/INT 0=na descida do sinal no pino RBO/INT
Bit 5	**TOCS**	Fonte de clock do TIMER 0 1=transição pino RA4/TOCKI 0=clock interno (CLKOUT=Fosc/4)
Bit 4	**TOSE**	Como o clock incrementará o TIMER 0? 1=na subida do sinal no pino RA4/TOCKI 0=na descida do sinal no pino RA4/TOCKI
Bit 3	**PSA**	Atribuição de prescaler 1=prescaler atribuído ao WATCH DOG 0=prescaler atribuído ao TMR0
Bit 2-RW	**PS2**	Ajustam a taxa de divisão do prescaler
Bit 1-RW	**PS1**	
Bit 0-RW	**PS0**	

Os bits PS2, PS1 e PS0 selecionam a taxa do prescaler, e estão detalhados na próxima tabela para facilitar a visualização.

Observe que um mesmo ajuste dá resultados diferentes co relação ao timer e ao WATCH DOG.

PS2	PS1	PS0	Divisão do Timer	Divisão do WATCH DOG
0	0	0	1:2	1:1
0	0	1	1:4	1:2
0	1	0	1:8	1:4
0	1	1	1:16	1:8
1	0	0	1:32	1:16
1	0	1	1:64	1:32
1	1	0	1:128	1:64
1	1	1	1:256	1:128

Observe que quando o prescaler está direcionado ao timer seu primeiro ajuste está em dividir por 2 (1:2).

Para assegurar uma taxa de 1:1 na freqüência de contagem do TIMER 0, direcionar o prescaler para o WATCH DOG.

Situação no Reset: Pull-ups do PORTB desabilitados, interrupção interna aceita na subida do pulso, TIMER 0com sinal externo, contando pela descida do sinal em RA4/TOCKI, prescaler ao WATCH DOG com divisão por 128.

5.3.3-Registro INTCON

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
GIE	**EEIE**	**TOIE**	**INTE**	**RBIE**	**TOIF**	**INTF**	**RBIF**

Endereço: 0BH, 8BH valor de reset:0000000X

Bit 7-RW	**GIE**	Bit global interrupt enable (habilitação global) 1=habilita as interrupções que estejam individualmente selecionadas 0=desabilita todas as interrupções
Bit 6-	**EEIE**	Interrupção de fim de escrita na EEPROM 1=habilita interrupção de fim de escrita da EEPROM 0=desabilita interrupção de fim de escrita da EEPROM
Bit 5-RW	**TOIE**	Interrupção gerada pelo overflow no TMR0 1=habilitada 0=desabilitada
Bit 4-R	**INTE**	Interrupção externa RB0/INT 1=habilitada 0=desabilitada
Bit 3-R	**RBIE**	Interrupção por mudanças na PORTB 1=habilitada 0=desabilitada
Bit 2-RW	**TOIF**	Sinaliza interrupção pelo overflow do TMR0(*) 1=ocorreu um overflow no TMR0 0=ainda não ocorreu overflow no TMR0
Bit 1-RW	**INTF**	Sinaliza interrupção externa no pino RB0/INT (*)
Bit 0-RW	**RBIF**	Sinaliza interrupção de mudanças na PORTB (*)

(*) devem ser zerados pelo software.

Situação no Reset: interrupções individuais e globais desabilitadas, requisições de interrupção do TMR0, interrupção RB0/INT desligadas. (o bit da interrupção em mudanças de estado não é afetado pelo reset).

5.4-Uso do Stack

diferente dos microprocessadores comuns, na família 16CXXX o stack não possui um registro de ponteiro (ou stack pointer, como é mais conhecido), acessível ao usuário.

O próprio stack, com seus 8 níveis, não é acessível ao programa (não permite leitura ou escrita).

Sempre que um CALL (chamada de sub-rotina) ou uma interrupção ocorrer, PC+1 será salvo no stack para permitir ao programa voltar ao ponto em que se encontrava antes.

Se mais de 8 CALL´s ou interrupções forem atendidas simultaneamente, o 1º endereço de retorno será perdido, sobrescrito pelo 9º, e assim por diante, de forma circular.

Como os membros da família 16CXXX não possuem flags sinalizadores de overflow ou underflow do stack, o usuário deve ter extremo cuidado para não permitir mais de 8 níveis de chamada ou retorno de simultâneos.

5.5-Endereçamento indireto

O SRF chamado INDF (endereço 00H da RAM) não é na verdade um registro fisicamente implementado.

Quando se acessa o registro INDF na verdade estamos acessando a posição indicada pelo FSR (file selection register- endereço 04H), que atua como um ponteiro para outras posições de memória.

Este é o chamado o endereçamento indireto de memória.

Vamos ver um exemplo bem simples:

```
CLRF    INDF    ;CLRF significa “clear register” onde “f” é o nome do
                endereço cujo conteúdo será zerado.
```

Se tivermos no registro FSR o valor 20H, a instrução acima não vai escrever o byte 00H na posição 00H da RAM, referente ao INDF , mas sim na RAM de endereço 20H, conforme apontado pelo SFR.

Importante: Nem o registro INDF e nem o FSR sofreram qualquer tipo de alteração.

6-EEPROM de dados.

O PIC 16F84 possui 64 bytes de memória EEPROM que podem ser utilizados como memória de dados. Sua principal vantagem em relação a RAM esta no fato de não perder as informações co a falta de alimentação. E. contrapartida, o tempo de escrita pode passar dos 10 ms (lembre-se que a 10 MHz, o PIC 16F84 escreve na RAM comum em 400 ns).

Sua principal utilização está nos sistemas que devem memorizar dados como o último número discado em telefones, ajustes de som, cor, brilho, em televisores e outros como memória das estações nos rádios, códigos personalizados de acesso. Entre várias outras possibilidades.

Estes 64 bytes não são diretamente endereçados como a RAM normal, mas sim através de 4 registros especialmente dedicados a este fim, que são:

-EEADR endereço desejado para leitura ou escrita na EEPROM
-EEDATA dado a escrever ou dado lido da EEPROM
-EECON1 registro de controle 1
-EECON2 registro de controle 2

6.1-Principio de funcionamiento

A EEPROM funciona como uma memória EPROM comum, isto é, tem os dados gravados eletricamente e não são perdidos com a falta de alimentação.

A principal diferença reside em que a mesma pode ser apagada eletricamente, sem a necessidade das luzes ultravioletas das EPROM´s comuns.

Possui ainda dispositivos internos para impedir o apagamento ou gravação acidental por erro do programa ou ruídos.

No caso da EEPROM do PIC 16F84 o usuário não precisa entender o funcionamento da mesma, bastando manipular corretamente os registros conforme a função desejada.

6.2-Registros de controle da EEPROM do PIC 16F84

Neste item estudaremos como cada registro de controle da EEPROM funciona e seu uso.

Sempre que necessário, pequenos trechos de código serão apresentados, lembrando ao leitor que maiores detalhes sobre cada instrução podem ser vistos no capítulo 18.

6.2.1-EEADR- Endereço da EEPROM

Este registro de 8 bits, presente na RAM no endereço 09H, permite escolher até 256 posições diferentes.

Lembre-se que apenas as 64 primeiras (00H a 3FH) estão implementadas no16F84.

Neste registro escrevemos o endereço onde queremos gravar ou o endereço que desejamos LER da EEPROM.

6.2.2-EEDATA- Dado da EEPROM

Neste registro de 8 bits presente na RAM no endereço 08H, escrevemos o dado a ser gravado no endereço EEADR, assim como teremos o dado lido da EEPROM quando efetuamos leitura.

6.2.3-EECON1- Registro de controle nº 1.

Registro que controla as operações co a EEPROM. Apenas 5 bits estão implementados.

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
-	-	-	**EEIF**	**WRERR**	**WREN**	**WR**	**RD**

Endereço: 88H valor no reset:0000X000

Bits 7, bit 6 e bit 5 não implementados, sempre lidos como 0.

Bit 4-R/W	**EEIF**	Flag de interrupção de fim de escrita 1=já escreveu na EEPROM.(zerar pelo software) 0=ainda não acabou de escrever
Bit 3-R/W	**WRERR**	Flag de erro na escrita da EEPROM 1=parou por reset do WATCH DOG 0=escrita completada com sucesso
Bit 2-R/W	**WREN**	Bit de habilitação de escrita na EEPROM 1=Escrita na EEPROM permitida 0=Não permitido escrever na EEPROM
Bit 1-(*)	**WR**	Bit de inicio da escrita 1=inicia um ciclo de escrita. (é zerado pelo hardware ao fim da escrita) 0=operação de escrita finalizada
Bit 0-(*)	**RD**	Bit de inicio de leitura 1=inicia a leitura da EEPROM (é zerado pelo hardware ao fim da escrita) 0=não inicia a operação de leitura

(*)-estes bits podem ser lidos, mas o software somente pode setar estes bits. Somente o hardware pode zerar estes bits.

Observações Importantes:

A) a operação de escrita na EEPROM leva em média 10 ms.
B) A leitura é realizada em apenas 1 ciclo de máquina.

Para iniciar uma operação de leitura ou escrita, basta acertar os valore de EEADR e EEDATA (na escrita) e setar os bits RD ou WR conforme a operação desejada.

Como a EEPROM não tem dados gravados imediatamente estes bits não podem ser zerados pelo software para evitar um truncamento do processo interno de leitura e principalmente de gravação.

Para proteção contra escritas indesejáveis, WREN deve ser mantido zerado pelo software, sendo setado somente quando for gravar um dado, e sendo novamente zerado ao final da gravação.

As operações de leitura não são afetadas pelo estado do bit WREN.

Se um reset ocorrer (MCLR\ ou WATCH DOG) durante uma operação de escrita, a mesma não se completará e haverá um erro, setando o bit WRERR. No reset sem queda de alimentação , os registros EEADR e EEDATA não são alterados, e pela verificação do bit WRERR o programa pode reinicias a gravação que foi interrompida.

No reset por power-up (ligação da alimentação ao circuito) o estado do bit WRERR é desconhecido e o programa deve proceder à outra lógica para diferenciar os dois tipos de reset e zerar o mesmo.

Sempre que uma operação de escrita for concluída com sucesso, o flag EEIF será setado e a interrupção da EEPROM será requisitada.

O usuário pode decidir entre habilitar esta interrupção ou efetuar a varredura no bit. Este bit deve ser zerado pelo software antes de terminar a rotina de interrupção.

6.2.4-EECON2

Este registro não necessita de ajustes por parte do usuário. É usado durante o processo de escrita na EEPROM conforme veremos no item 6.5.

6.3-Leitura da EEPROM

A leitura da EEPROM é feita apenas endereçando a posição desejada da mesma e setando o bit RD em EECON1.Antes devemos escrever em EEADR o endereço desejado.

Exemplo: Lendo o endereço 30H da EEPROM e salvando na RAM de endereço 25H(RP1=0 sempre).

```
bcf     STATUS,RP0      ;bcf →bit clear em 'f'. Zera o bit indicado no registro f
                        ;Seleciona o banco 0, onde está o registro EEADR
movlw   30H             ;registro W=constante 30H
movwf   EEADR           ;move o valor de w para EEADR, EEADR=30H
bsf     STATUS,RP0      ;bsf →bit set em 'f'. Seta o bit indicado no registro f
                        ;seleciona o banco 1, onde está o registro EECON1
bsf     EECON1          ;faz RD=1 iniciando a leitura.
                        ;A leitura leva apenas um ciclo
bcf     STATUS,RP0      ;volta ao banco 0, onde está EEDATA
movf    EEDATA,0        ;W=EEDATA, que é o dado que estava na EEPROM
movwf   25H             ;escreve W na RAM com endereço 25H.
```

6.5-Escrita na EEPROM

Para demonstrar o processo de escrita na EEPROM vamos nos utilizar mais um exemplo de programa bem simples e detalhado, escrevendo no endereço 20H da EEPROM 0 dado 12H.

```
bcf         STATUS,RP0      ;Seleciona banco 0
movlw       20H             ;W=20H
movwf       EEADR           ;EEADR=W=20H
movlw       12H             ;W=12H
movwf       EEDATA          ;EEDATA=W=12H
bsf         STATUS,RP0      ;seleciona banco 1 para EECON1
bcf         INTCON,GIE      ;e já desabilita interrupções
                            ; pois INTCON está no banco 0 e 1
bsf         EECON1,WREN     ;permite escrita na EEPROM
movlw(*)    55H             ;W=55H
movwf (*)   EECON2          ;EECON2=55H
movlw (*)   0AAH            ;W=0AAH
movwf (*)   EECON2          ;EECON2=0AAH
bsf(*)      EECON1,WR       ;bit WR=1. inicia o processo de escrita.
```

As 5 interrupções marcadas com (*) são obrigatórias e nesta ordem, e sem as mesmas o processo de escrita não será executado.

As interrupções devem estar globalmente desabilitadas antes destas 5 instruções.(GIE=0).

Este mecanismo de escrever 55H e depois AAH no registro EECON2 antes de fazer WR=1 evita que eventuais erros ou desvios inesperados de programa.possam setar o bit WR e escrever em posições não desejadas. Aliado ao bit WREN permite grande margem de segurança para dados da EEPROM.

Após estas linhas de código o bit WREN pode ser imediatamente zerado que não atrapalhará o ciclo de escrita em andamento e teremos garantido a proteção a EEPROM.

6.5.1-Como configurar o fim da escrita na EEPROM

Após este trecho podemos optar por duas linhas de raciocínio:

1) esperar a escrita acabar efetuando varredura no bit EEIF sem gerar interrupção, criando uma sub-rotina de espera.

```
                            ;considere que já estamos no banco 1
espera:
btfss       EECON1,EEIF     ;se EEIF for 1 pula a próxima instrução.
goto        espera          ;EEIF=0, volta para a linha 'espera'
bcf         EECON1,EEIF     ;EEIF=1,pulei a linha do goto
                            ;e já zero o bit sinalizador
bsf         INTCON,GIE      ;volta aceitar interrupções.
```

2)reabilitar as interrupções, inclusive a da EEPROM e ao fim da escrita atender a mesma, resetando o bit EEIF.

É o processo mais indicado, pois a escrita na EEPROM pode levar 10 ms ou mais. Com clock de 10MHz, executamos 1 instrução a cada 400 ns, e nos 10 ms a CPU já poderá ter executado 25000 instruções.

7-Portas de entrada

As portas de entrada/saída são mais conhecidas por portas I/O, e serão assim chamadas deste ponto em diante.

O PIC 16F84 possui 2 portas I/O, chamadas como vimos de PORTA e PORTB.

Alguns pinos das portas são multiplexados com outras funções, como interrupção externa e contador externo.

7.1-PORTA, seu registro de controle de direção e sua função alternativa.

O PORTA é composto de um latch de 5 bits e cada um de seus bits pode ser individualmente ser ajustado com entrada ou saída, sendo também referenciados como RA (RA0 à RA4).

Para ajuste do bit da porta como entrada ou saida, um registro especial chamado TRISA existe na RAM e quando um de seus bits está em 1 o bit correspondente do PORTA é entrada, e se for 0 será saída.

No reset todos os bits nos registros TRISA e TRISB são setados, ajustando todos os bits das portas como entrada.

7.1.1-Condições elétricas

BIT	ENTRADA	SAÍDA
PORTA bit 0(RA0)	TTL	TTL
PORTA bit 1(RA1)	TTL	TTL
PORTA bit 2(RA2)	TTL	TTL
PORTA bit 3(RA3)	TTL	TTL
PORTA bit 4(RA4)	Schmidt Trigger	Open Drain

7.1.2-Leitura e escrita

Ler a porta significa ler o estado presente nos pinos, enquanto escrever significa escrever latchs de saída.

Todas as operações de escrita são realizadas numa seqüência .

LÊ → MODOFICA → ESCREVE

Como funciona ?

Quando se escreve em qualquer bit da porta, a CPU lê o estado atual dos latches de saída da porta toda, modifica os bits desejados e reescreve toda a porta nos latches de saída.

7.1.3-Função alternativa do pino RA4

O pino RA4 também pode ser usado como entrada de sinal externo pata o TIMER 0 (estudaremos no capítulo 8).

7.1.4-Inicialização

Sempre após o reset as portas devem ser inicializadas escrevendo-se nas mesmas , antes de ajustar o TRIS equivalente.

Exemplo: inicializando o PORTA com bits RA0 e RA1 saída e os demais entrada.

Para tal TRISA deve ser escrito XXX11100, então escreveremos 00011100 = 0CH

```
clrf    PORTA       ;inicializa PORTA
bsf     STATUS,RP0  ; seleciona o banco 1 para acessar TRISA
movlw   1CH         ;W=1CH=00011100
movwf   TRISA       ;ajusta RA0 e RA1 saídas, RA2, RA3 e RA4
entrada
```

7.1.5-Sobre o TRIS

Os registros TRISA e TRISB estão alocados no banco 1 de memória e podem se alterados a qualquer momento pelo programa. Isto significa que um bit ajustado como saída pode ser ajustado para ser entrada no decorrer do programa.

Isto permite que o projeto do hardware permita grande flexibilidade, usando um só pino com as duas funções.

7.2-PORTB, seu registro de controle de direção e suas funções especiais

O PORTB é uma porta de 8 bits e assim como o PORTA, cada um de seus bits pode ser configurado como entrada ou saída, pelo registro TRISB.

Cada bit do PORTB é chamado de RB0, RB1,....,RB7.

Como no PORTA, para o PORTB:

Bit TRIS = 1→Entrada (padrão do reset)
Bit TRIS = 0→Saída

7.2.1-Condições elétricas.

Quando I/O, todos os bits do PORTB são TTL. RB0 é SCHIMIDT TRIGGER quando usada para interrupção externa

7.2.2-Leitura e escrita

Funcionam de forma semelhante ao PORTA. Na leitura o estado presente nos pinos é efetivamente carregado, e na escrita é realizada a operação Lê→Modifica→Escreve já vista.

7.2.3-Pull-ups

Todos os bits do PORTB tem um pull-up interno. Para habilitar o funcionamento dos mesmos de forma global, o bit RBPU\ (OPTION, bit 7) deve ser zerado. Este bit é setado no reset.

Individualmente, apenas os bits do TRISB ajustados como entrada terão os pull-ups habilitados, mesmo que RBPU\ = 0.

7.2.4-Interrupção de mudança de estado no PORTB.

Os pinos RB4 e RB7 tem como função alternativa gerar um pedido de interrupção sempre que o estado dos mesmos sofrer alteração (novamente, somente aqueles configurados como entrada).

O estado dos pinos é comparado ao estado anterior armazenado em um latch, e se o estado foi alterado (de 1 para 0 ou de 0 para 1), o bit RBIF será setado gerando a interrupção (se estiver habilitada).

Esta interrupção pode fazer com que o chip saia do modo SLEEP (capítulo 16) e há duas maneiras de ressetar o pedido de interrupção: zerar o bit RBIF ou ler o PORTB.

7.2.5-Utilidade dessas funções especiais.

A característica de possuir pull-ups internos e sair do modo SLEEP pela mudança de estado em alguns bits permite a fácil implementação de teclados e sobretudo de dispositivos que operam somente com o pressionar de determinadas teclas, como controles remotos, por exemplo, economizando energia durante os períodos de inatividade.

Não se recomenda a varredura desses pinos do PORTB se a interrupção em mudança de estado estiver habilitada.

7.2.6-Função alternativa do pino RB0/INT (interrupção externa)

O pino RB0/INT tem como função alternativa entrada para um sinal externo de interrupção (item 14.7).

7.2.7-Inicislização do TRIS

Funciona de modo semelhante ao PORTA.

7.3-Escrita e leitura seqüenciais na mesma porta.

Devido a problemas inerentes a carga externa e ao funcionamento interno do hardware de escrita e leitura, deve~se evitar escrever e ler na mesma porta seqüencialmente, para não correr o risco de que na leitura seja lido o bit que foi alterado na instrução anterior, antes da mudança real ter ocorrido.

Não fazer:

```
bcf     PORTB,1         ;faz RB1=0
btfsz   PORTB,1         ;se RB1=0, pula a próxima instrução.
goto    RB1_1           ;RB1=1, vai para o label RB1_1
```

Corremos o risco de não ter dado tempo para que RB1 seja efetivamente zerado e o goto seja executado.

Maneira segura:

```
bcf     PORTB,1         ;faz RB1=0
nop                     ;perde 1 ciclo
btfsz   PORTB,1         ;se RB1=0, pula a próxima instrução.
goto    RB1_1           ;RB1=1, vai para o label RB1_1
```

Aqui demos um tempo entre alterar e ler o mesmo bit, assegurando que deu tempo para a instrução ser executada externamente ao chip.

8-Módulo TIMER 0

O PIC 16F84 possui um contador/temporizador (TIMER 0) de 8 bits que incrementa seu valor a cada pulso na entrada RA4/TOCKI (modo contador) ou a cada ciclo de instrução (modo timer) que ocorre a freqüência Fosc/4.

Quando ocorre o overflow de 255 para 0, o hardware seta o bit TOIF gerando o pedido de interrupção do TIMER 0.

Suas principais características:

- 8 bits de contagem
- o registro TMR0 pode ser lido ou escrito a qualquer momento
- sinal de contagem interno ou externo
- pode ter o sinal de contagem dividido por um prescaler
- geração de interrupção se estiver habilitada

8.1-Modo Timer (sinal interno Fosc/4)

Ao ajustarmos TOCS=0 (registro OPTION, bit 5), fazemos com que o TIMER 0 seja incrementado a cada ciclo de instrução (a 4 MHz, incrementa a cada 1 us).

Nestas condições o TIMER 0 funciona livremente. Não existe um controle para que o mesmo não seja incrementado. Se o timer o não for utilizados manter sua interrupção sempre desabilitada com TOIE = 0 (INTCON, bit 5).

IMPORTANTE: sempre que se escreve um novo valor no registro TMR0, a contagem é suspensa por dois ciclos de máquina. Este atraso deve ser levado em conta pelo software.

8.2-Modo contador (sinal esterno no pino RA4/TOCKI)

Com o bit TOCS=1 o TIMER 0 incrementará sua contagem a cada pulso presente no pino RA4/TOCKI, que deverá ser ajustado como entrada (TRISA, bit 4=1).

Para que o contador incremente na subida do sinal externo, o bit TOSE (OPTION, bit 4) deve ser zerado, e para que o contador incremente na descida do sinal externo, o bit TOSE deve ser setado.

8.2.1-Sem prescaler

Se não usarmos o prescaler (divisor de freqüência) os tempos em nível alto e baixo do sinal externo devem ser de pelo menos 2 Tosc (ou mais que ½ Tciclo), devido às limitações na amostragem do sinal interno e ao não sincronismo entre sinal externo e fases internas da CPU, que realizam amostragem no pino RA4/TOCKI duas vezes por ciclo.

8.2.2-Com prescaler

Se o prescaler for utilizado, os pulsos do sinal externo serão divididos pelo valor do prescaler.

Para permitir a amostragem é necessários que o período do sinal seja pelo menos 4X Tosc (1 ciclo) “dividido” pelo valor do prescaler, e o tempo alto ou baixo nunca devem ser menores que 10 ns.

Como a saída do prescaler é sincronizada com o clock interno, um pequeno atraso pode ocorrer entre os pulsos do sinal e o efetivo incremento do registro TMR0.

8.3-Interrupção do TIMER 0

Quando o registro TMR0 muda de FFH (255) para 0, ocorre o chamado overflow do timer, e o bit TOIF será setado sinalizando o overflow e gerando o pedido de interrupção.

Se a interrupção estiver habilitada (TOIE=1 e GIE=1), o endereço PC+1 será salvo no stack e então o programa será desviado para o endereço 004H onde tem inicio a rotina de tratamento de interrupções.

Antes de finalizar a interrupção (instrução RETFIE), o bit TOIF deve ser zerado, senão a interrupção será novamente chamada.

Como o TIMER 0 fica parado no modo SLEEP, não há a possibilidade de sair do modo SLEEP por esta interrupção.

8.4-Prescaler

Um outro registro de 8 bits está disponível como divisor de freqüência, permitindo divisões de até 256 vezes na freqüência do sinal de entrada.

Este prescaler pode ser utilizado como extensor do tempo do WATCH DOG, de modo mutuamente exclusivo, Istoé se for utilizado pelo timer não estará disponível ao WATCH DOG, e vice versa. (maiores detalhes no capítulo 15).

No item 5.3.2 temos os ajustes do prescaler (OPTION, bits 3, 2, 1 e 0).

8.4.1-Prescaler para o TIMER 0

Quando designado para o TIMER 0, o prescaler divide a freqüência de entrada (externa ou interna) por 2, 4, 8,....., 256, conforme ajustes no OPTION.

Se desejarmos que não haja qualquer divisão do sinal para o TIMER 0, basta designar o prescaler para o WATCH DOG, e teremos 1:1.

Embora o valor da divisão seja ajustado no OPTION, o registro onde é incrementada a contagem não é acessível ao usuário.

Este registro é zerado sempre que se efetua qualquer escrita no registro TMR0.

9-Fusíveis de configuração interna

O PIC 16F84 possui no endereço 2007H da memória de programa 14 bits implementados que permitem ao usuário escolher a configuração de trabalho do chip.

Estes bits são acessíveis somente na gravação ou verificação, e seu estado quando o chip está apagado é ‘1’.

Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
CP	CP	CP	CP	CP	CP	CP	CP	CP	CP	PWRTE	WDTE	FOSC1	FOSC0

Endereço 2007H valor no reset: uuuuuuuuuuuuuu

Bit 13 a 4	CP	Bit de proteção do código 1= código desprotegido 0= código protegido
Bit 3	PWRTE	Habilitação do power-up timer 1= power-up timer desabilitado 0= power-up timer habilitado
Bit 2	WDTE	Bit de habilitação do WATCH DOG 1= WDT habilitado 0= WDT desabilitado
Bit 1	FOSC1	Seleciona tipo de oscilador 11= modo RC externo 10= modo cristal ou ressonador de alta velocidade HS 01= cristal ressonador de baixa velocidade XT 00= cristal de baixa potência LP
Bit 0	FOSC0	

IMPORTANTE:

O leitor não precisa se preocupar com esta posição de memória, pois os gravadores permitem ao usuário apenas escolher as opções desejadas e então efetua a gravação automaticamente, sem que o usuário precise lembrar qual o nome de cada bit e nem seu endereço.

10-Configurações do oscilador

O PIC 16F84 pode funcionar com 4 modos de oscilador.

Ao gravar o chip devemos escolher qual o oscilador utilizado sob pena de não funcionar ou até mesmo danificar o chip.

Os modos de funcionamento são:

1-RC, modo RC externo
2-HS, cristal ou ressonador de alta velocidade (> 4MHz)
3-XT, cristal ou ressonador de baixa velocidade (de 100 KHz a 4 MHz)
4-LP, cristal de baixa potência (< 200 KHZ)

Estes valores são mais ilustrativos do que regras rígidas. É possível , por exemplo, rodar a 10 MHz no modo XT, embora não seja aconselhável.

Como os mais utilizados são o modo XT e o RC, vamos estudar apenas estes dois. Caso o leitor desejar informações mais detalhadas sobre estes e os outros modos, deverá recorrer aos manuais do fabricante.

10.1-Modo RC

Neste modo usamos um circuito RC externo para gerar oscilação. Neste modo não temos precisão dos valores de clock, sendo mais usado nas aplicações que não requerem rotinas de tempo baseadas no clock interno, sendo também a alternativa mais econômica.

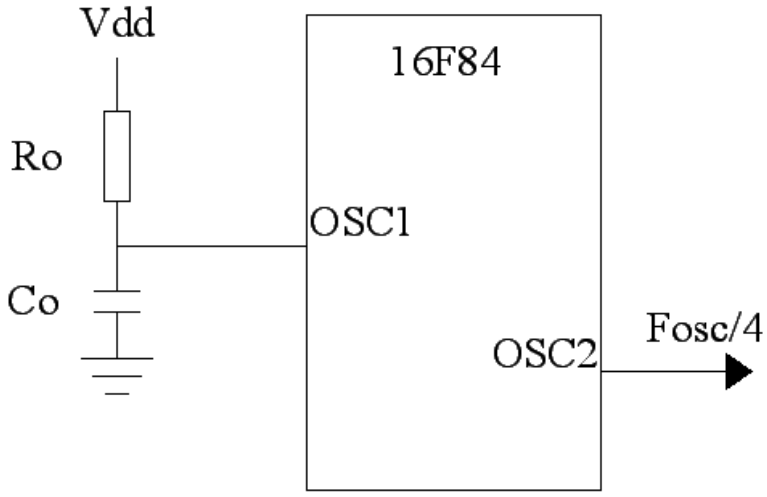


Figura 10.1-Modo RC do PIC 16F84

No pino OSC1 ligamos a junção do resistor e o capacitor, e no pino OSC2 temos a freqüência de clock dividida por 4 (Fosc/4).

Recomenda-se que Ro esteja entre 3 KΩ e 100KΩ, e Co pode nem existir, embora recomenda-se valores acima de 20 pF para evitar ruídos e melhoras a estabilidade.

Com /co = 33pF e um trimpot de 10 KΩ em Ro, podemos facilmente ajustar a freqüência de clock em 4 MHz, gerando ciclo de instruções de 1 us.

Para tal coloque um osciloscópio ou frequencímetro no pino OSC2 e ajuste Ro até que a freqüência indicada seja de 1 MHz (Fosc/4).

Importante: Mesmo ajustando a freqüência em 4 MHz como acima, o clock poderá cariar devido a variações de temperatura e flutuações na alimentação.

10.2-Modo cristal ou ressonador XT

Neste modo temos o circuito na figura 10.2

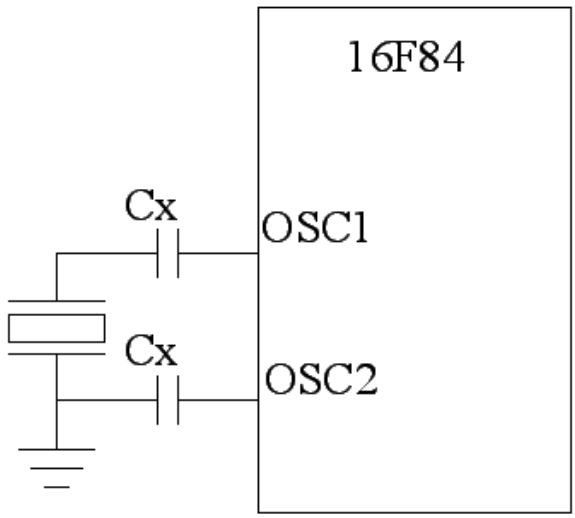


Figura 10.2-Modo XT

Para cristal de 4 MHz os capacitores podem ter valores entre 15 e 33 pF. Valores maiores aumentam o tempo de estabilização do oscilador, podendo até mesmo não oscilar.

11-Reset

Vários eventos podem resetar o chip:
- ao ligar, com power-on reset;
- pino MCLR\ ir a ‘0’ durante operação normal ou SLEEP;
- Overflow do WATCH DOG durante operação normal ou durante SLEEP;

11.1-Power-on reset (POR)

O circuito mínimo para gerar o POR consiste apenas em ligar o pino MCLR\ ao Vdd, eliminando os circuitos RC comumente utilizados.

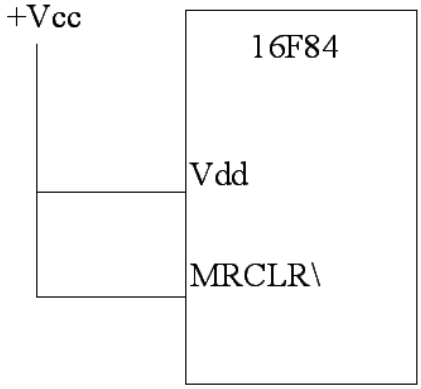


Figura 11.1-Circuito mínimo para POR do PIC 16F84

11.1.1-Circuito de POR mais elaborado

Há casos que se deseja ter a possibilidade de resetar o chip manualmente, ou devido a problemas na alimentação precisamos de um circuito mais confiável. Nestas condições, o circuito da figura 11.2 atende plenamente esta função.

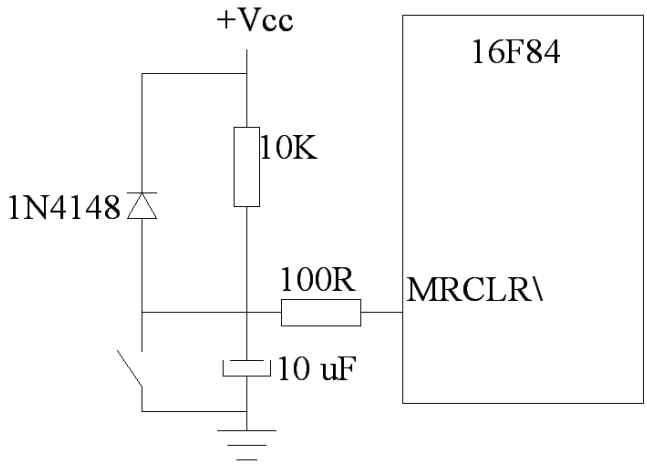


Figura 11.2-Circuito de Reset Melhorado do PIC 16F84 com reset manual

11.2-Situação dos principais registros após o reset

Como ainda vamos estudar o WATCH DOG e alguns registros comportam-se de maneira diferente dependendo do reset, veremos no item 16.6 o estado dos registros após os vários tipos de reset.

12-POWER-UP (PWRT)

O PIC 16F84 possui ainda um sistema interno que gera um delay de 72 ms fixo, a partir do final do POR (Power-on reset ou reset ao ligar a alimentação).

O chip já permanece em modo de reset durante o tempo normal de reset, seja pelo RC externo ou pelo tempo de subida da fonte, quando o pino MCLR\ está direto em Vdd.

Com este timer habilitado, mais 72 ms são adicionados neste tempo durante o qual o estado do reset é mantido.

Este recurso permite evitar que o reset seja não seja corretamente executado principalmente por problemas de ruído ou demora na estabilização da fonte.

O tempo de 72 ms é normal, podendo variar pela variação do Vdd, temperatura, ...,etc.

13 –Timer de partida do oscilador (Oscillator Start-up Timer - OST)

quando utilizamos o modo XT, LP ou HS, este timer e automaticamente acionado logo apões o tempo do PWRT, fazendo com que a CPU ainda espere mais 1024 períodos do oscilados para sair do reset.

É útil para garantir que o cristal ou ressonador já estejam estabilizados.

13.1-Vantagens

O uso combinado dos recursos de PWRT e OST, geralmente garante que a operação de reset será bem sucedida, permitindo o uso circuito mínimo de reset visto na figura 11.1.

Se porém, a fonte estiver sujeita a flutuações ou ruído excessivo, além destes recursos, o circuito da figura 11.2 é o mais indicado.

14-Interrupções no PIC 16F84

O PIC 16F84 possui 4 sinalizadores internos de interrupção:

1- Interrupção externa pelo pino RB0/INT
2- Overflow do TIMER 0
3- Fim da escrita na EEPROM
4- Mudança nos pinos RB4 à RB7

Com exceção da interrupção de fim de escrita na EEPROM, cujo bit sinalizador está no registro EECON1 (item 6.2.3), todos os demais bits de controle de habilitação e sinalização estão no registro INTCON (item 5.3.3).

Basicamente podemos colocar assim:

Se GIE=0 nenhuma interrupção será atendida, independente do estado dos bits individuais de habilitação estarem ou não habilitando uma interrupção em particular.

Se GIE=1 As interrupções que estiverem com seus bits de habilitação em 1 serão atendidas, e as demais estarão desabilitadas.

Quando qualquer interrupção que estiver sendo atendida, GIE será zerado pelo hardware e setado novamente ao fim desta.

14.1-Princípio de atendimento de interrupções

Ao aceitar a interrupção, a CPU faz GIE = 0, para inibir que outras interrupções possam interromper a que já esta sendo executada, salva no STACK o endereço de retorno (a próxima após o PC atual, isto é, PC+1), e depois desvia para o endereço 004 da memória de programa.

Ao encontrar a instrução RTFIE (return from interrupt-volte da interrupção) a CPU finaliza a interrupção em andamento, recuperando a PC do STACK e fazendo novamente GIE = 1.

A instrução RTFIE nunca deve ser utilizada como retorno de sub-rotinas (ver capítulo 18, instruções RETLW e RETURN).

Para as interrupções externas, o atraso no inicio da rotina de interrupção é de 3 a 4 ciclos de instrução.

14.2-Determinando a fonte da interrupção.

Como a interrupção sempre desvia o PC para o endereço 004H, o software deve verificar nos bits de requisição de interrupções qual setado, determinando a fonte da interrupção.

Estes bits, como já vimos, são: RBIF, INTF, TOIF, EEIF.

É importante sempre frisar dois pontos:

1º- Mesmo com GIE = 0, os bits são setados pela respectiva fonte interrupção. Por exemplo, no overflow do TIMER 0, o bit TOIF será setado, mesmo que TOIE = 0.

2º- Com base nisto, antes de habilitar uma interrupção em particular o leitor deve verificar se a mesma já não esta sendo requisitada (já está pendente), pois seu atendimento imediato ao habilitá-la pode não ser desejado.

14.3-Cuidados ao desabilitar globalmente as interrupções (GIE= 0)

Existem situações nas quais o usuário deverá inibir temporariamente as interrupções, fazendo GIE = 0, como por exemplo na escrita da EEPROM (item 6.4).

Se porém, um pedido de interrupção for aceito no mesmo momento (lembra-se que vários eventos podem ocorrer simultaneamente à execução de uma instrução, como amostragem dos sinais dos sinais de interrupção, por exemplo) corremos o risco de ter a interrupção atendida, e ao voltar desta a instrução RTFIE fará GIE = 1 novamente, voltando o programa na posição imediatamente após aquela onde fizemos GIE = 0. Assim, pensamos que as interrupções estão desabilitadas, mas na verdade foram reabilitadas pelo fim da interrupção que foi atendida.

Para assegurar que GIE = 0 devemos executar a seguinte seqüência :

```
garante:
(1)     bcf     INTCON, GIE     ; fazemos GIE = 0
   (*)                          ;
(2)     btfsc   INTCON, GIE     ;se GIE = 0 pula a próxima instrução pelo motivo
                                ;acima
(3)     goto    garante         ;exposto, GIE pode ser 1 ainda. Neste caso volta
para
                                ;zerar.
(4)                             ;continua com GIE = 0
```

(*) Devemos observar que uma interrupção pode ter sido atendida entre as instruções 1 e 2 acima.

14.4- A interrupção da EEPROM

Sempre que uma operação na EEPROM for finalizada, o bit EEIF(EECON1, bit 4), será setado, gerando o pedido de interrupção da EEPROM, que será atendido se os bits EEIE = 1 e GIE = 1.

Para maiores detalhes ver o capitulo 6.

14.5-interrupção do TIMER 0

Ao ocorrer um overflow no registro TMR0, o bit TOIF será setado e a interrupção do TIMER 0 será requisitada, se TOIE e GIE = 1.

Lembre-se de que o TIMER 0, quando com clock interno (Fosc/4), terá seu registro TMR0 incrementado sempre, independente dos bits de controle. Não há controle tipo contar/parar no TIMER 0.

Ver capítulo 8 para maiores detalhes sobre o TIMER 0.

14.6-Interrupção de mudança no PORTB

Uma mudança de estado nos bits RB4 à RB7 ajustados como entrada vai pedir a CPU uma interrupção , setando o bit RBIF. Esta interrupção será atendida se RBIE = 1 e GIE = 1.

Ver item 7.2.4 para maiores detalhes.

14.7-Interrupção externa RB0/INT

O pino RB0/INT, ao invés de I/O digital, pode ser usado como entrada para sinal externo de interrupção.

Sempre que INTE = 1 e GIE = 1, uma variação de 1 para 0 (INTEDG = 1) ou de 0 para 1 (INTEDG = 0) neste pino fará INTF = 1, sinalizando o pedido de interrupção externa.

14.8-Reset dos bits de requisição de interrupção

IMPORTANTE:

O hardware não reseta os bits de pedido de interrupção INTF, RBIF, EEIF e TOIF. Cabe ao usuário zerar os mesmos antes de executar a instrução RTFIE.

14.9-Salvando a situação atual durante o atendimento das interrupções

Como já vimos, somente o PC+1 é salvo no Stack ao desviar para o endereço 004H, atendendo a interrupção.

Como praticamente todas as instruções usam o registro W e o STATUS, sobretudo devido às manipulações dos bancos de registradores e sinalizações da ULA (zero, carry, ...) convém ao usuário salvar os mesmos em posições reservadas da RAM para poder recupera-los (vale para qualquer outra posição de memória que porventura possa ser alterada na rotina de interrupção).

O trecho de programa a seguir armazena a seguir o registro W em uma posição de RAM chamada de W2 e o STATUS em uma outra posição da RAM chamada de STATUS2. Estas posições são determinadas pelo usuário ao escrever o programa.

```
004:
movwf       W2          ;primeira instrução da interrupção
                        ;W2=W
movf        STATUS,0    ;faz W=STATUS
movwf       STATUS2     ;STATUS2=STATUS original
…                       ;
…                       ;
…                       ;rotina da interrupção
…                       ;
…                       ;
movf        STATUS2,W   ;finalizou a rotina de interrupção
                        ;copia o STATUS antigo em W
movwf       STATUS      ;e volta ao registro original
movf        W2,0        ;recupera o W original
retfie                  ;retorna
```

15-WATCH DOG

o WATCH DOG é um timer especial que roda livremente sem qualquer ligação com o clock ou na dependência de sinais externos. O WATCH DOG funciona mesmo que o clock seja retirado do sistema.

Se ocorrer um overflow do registro do WATCH DOG (este registro não é acessível ao usuário), mais conhecido por time-out do WATCH DOG, a CPU será resetada, e se isto ocorrer durante o modo SLEEP a CPU retornará a operação normal, no endereço logo após a instrução SLEEP.

O WATCH DOG pode estar permanentemente ligado ou desligado, sendo esta opção ajustada com o fusível de programação WDTE (cap. 9), e este ajuste não pode ser alterado durante a execução do programa.

15.1-Finalidade e princípio de funcionamento.

A principal finalidade do WATCH DOG está em recuperar a CPU de eventuais travamentos no programa, que podem ocorrer por interferências externas ou até mesmos erros no software.

Basicamente, se o WATCH DOG estiver habilitado, devemos de tempos em tempos resetá-lo, de forma a nunca ocorrer o overflow do mesmo.

Se em qualquer momento o programa travar, o WATCH DOG não será resetado e o seu registro sofrerá um overflow, ocasionando o reset da CPU.

15.2-Período do WATCH DOG

O período do WATCH DOG é de 18 ms, sem o prescaler, valor este ajustado na fabricação do chip. Este tempo médio pode ser alterado por variações na temperatura, na alimentação e nos processos de fabricação do chip.

Com prescaler ajustado para funcionar com o WATCH DOG, este tempo pode chegar a até 2 segundos ou mais.

Observe no item 5.3.2 que as taxas de divisão para o WATCH DOG vão de 1:1 à 1:128 (128 X 18ms = 2,304 s).

São duas as instruções que zeram o registro do WATCH DOG: SLEEP (entra em modo SLEEP) e CLRWDT (zera o WATCH DOG). Além do registro do WATCH DOG, estas instruções zeram também o registro do prescaler (o registro onde o prescaler efetua a contagem, e não seu ajuste, feito no OPTION).

Sempre que ocorrer um overflow do WATCH DOG (time-out) o bit TO\ será zerado, devendo a rotina de inicialização verificar este bit para saber se o reset foi normal ou gerado pelo WATCH DOG.

15.3-Uso com interrupção

Embora o reset do WATCH DOG pela instrução CLRWDT possa ser executado a qualquer momento, recomenda-se não faze-lo nas interrupções, pois o programa pode estar travado e ainda assim estar atendendo interrupções normalmente.

Como exemplo vamos citar a interrupção externa RB0/INT baseada nas passagens do zero da rede. Com freqüência de 60 Hz temos 120 interrupções (duas por ciclo) por segundo que ocorrem a aproximadamente 8,333 ms. Se o WATCH DOG estiver com tempo mínimo (18 ms) e efetuamos o CLRWDT nas rotinas de zero da

rede, o time-out nunca ocorrerá, e ainda assim um determinado trecho de programa fora desta interrupção poderá estar travado.

16-Modo SLEEP (Power Down)

O modo power-down, mais conhecido por modo SLEEP, faz com que o chip entre em estado praticamente inerte, onde o chip em si passa a consumir aproximadamente 60 uA (contra 7 mA em operação normal).

Se o hardware for projetado visando este modo, deverá set tal que economize energia nestas condições.

Esta característica aliada aos pull-ups do PORTB e a interrupção nas mudanças de estado permitem o projeto de sistemas para funcionar a bateria com longa duração, como controles remotos em geral.

Para entrar no modo power-down, basta executar a instrução SLEEP.

Neste modo, se o WATCH DOG estiver habilitado,o mesmo sra zerado e continuará rodando, o bit PD\ que sinaliza o modo power-down será zerado e o bit TO\ (Time-out) será setado, e o oscilador interno desligado, não gerando mais fases Q1 a Q4 do clock e parando tanto a CPU quanto o TIMER 0.

Se uma escrita estiver ocorrendo na EEPRM neste momento a mesma será finalizada normalmente.

O estado dos pinos I/O permanece inalterado.

16.1-Saindo do modo power-down (SLEEP)

Existem três maneiras de sair do modo SLEEP:

- Reset externo pelo pino MCLR\
- Time-out do WATCH DOG, caso esteja habilitado
- Interrupção externa RB0/INT, interrupção por mudanças no PORTB e interrupção de fim de escrita na EEPROM.

O TIMER 0 não finaliza o modo SLEEP porque não funciona sem o clock.

Se sair pelo reset no pino MCLR\ a CPU iniciará pelo endereço 000H.

As interrupções citadas, para tirarem a CPU do SLEEP precisam estar com os bits de requisição individuais habilitados, embora GIE não precise estar em ‘1’.

Temos então dois modos de funcionamento:

A)GIE = 0 A CPU volta a funcionar pela execução da instrução imediatamente após a instrução SLEEP. É como se o SLEEP fosse uma sub-rotina, que parou todo o chip.

B)GIE = 1 A instrução imediatamente após o SLEEP será executada, e então a interrupção será executada como uma interrupção qualquer.

Se por ventura desejarmos que a interrupção seja atendida sem que a instrução após o SLEEP seja executada antes, basta colocar NOP como a instrução após o SLEEP.

16.2-Interrupção simultânea à execução da instrução SLEEP

A execução de qualquer instrução ocorre em 1 ciclo, que possui 4 fases pó pulsos de clock, como já vimos.

Se durante umas das fases Q1 a Q4 da instrução SLEEP ocorrer uma requisição de interrupção, a CPU poderá se comportar de 2 maneiras distintas:

1)A interrupção ocorreu antes da execução total do SLEEP

O SLEEP não será executado, sendo tratado como um NOP pela CPU. Desta forma, o WATCH DOG não será zerado, TO\ não será setado e PD\ não será zerado.

Em outras palavras, é como se o SLEEP nunca tivesse existido.

2) A interrupção ocorreu imediatamente após ou durante o reconhecimento do SLEEP

Embora entre em SLEEP a CPU voltará imediatamente ao normal (o SLEEP terá se comportado como uma simples instrução), embora o WATCH DOG tenha sido zerado e os bits de sinalização ajustados, isto é, TO\ = 1 e PD\ = 0.

Para que o programa saiba se realmente o SLEEP ocorreu basta ver se o bit PD\ =0. se não estiver, o SLEEP foi executado como um NOP.

Embora o WATCH DOG seja zerado pelo SLEEP, devemos por segurança incluir a instrução CLRWDT imediatamente antes da instrução SLEEP, pois por coincidência o WATCH DOG pode estar sofrendo time-out durante execução do SLEEP (como no caso 1 acima).

16.3-Estado dos SFR após os vários tipos de reset

Temos na tabela a seguir o estado dos principais SFR após os vários tipos de reset possíveis.

As convenções para esta tabela são:

U = inalterado
- = não implementado
X = desconhecido
Q = depende de outras condições

SFR	POR	MCLR\ normal ou após SLEEP, WATCH DOG em uso normal	Sai do SLEEP por interrupção ou WATCH DOG
PC	000H	000H	PC+1
STATUS	00011XXX	000QQUUU	UUUQQUUU
W	XXXXXXXX	UUUUUUUU	UUUUUUUU
TMR0	XXXXXXXX	UUUUUUUU	UUUUUUUU
FSR	XXXXXXXX	UUUUUUUU	UUUUUUUU
PORTA	- - - XXXXX	- - - UUUUU	- - - UUUUU
TRISA	- - - 11111	- - - 11111	- - - UUUUU
PORTB	XXXXXXXX	UUUUUUUU	UUUUUUUU
TRISB	11111111	11111111	UUUUUUUU
EEDATA	XXXXXXXX	UUUUUUUU	UUUUUUUU
EEADR	XXXXXXXX	UUUUUUUU	UUUUUUUU
EECON1	- - - 0X000	- - - 0Q000	- - - 0UUUU
OPTION	11111111	11111111	UUUUUUUU
INTCON	0000000X	0000000U	UUUUUUUU

Figura16.1-Estado dos SFR após vários tipos de reset.

17-Posições de identificação

Existem 4 endereços na memória de programa, de 2000H à 2003H, que servem para armazenar até 4 nibles (números ou palavras de 4 bits) como identificação (ID), check-sum ou código outro código desejado pelo usuário.

Estes valores não podem ser lidos durante a execução normal do programa, apenas durante a gravação/verificação do chip.

Apenas os 4 bits menos significativos são utilizados.

Bit 13			Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
-	-	-	-	a	b	c	d
-	-	-	-	e	f	g	h
-	-	-	-	i	j	k	l
-	-	-	-	m	n	o	p

Exemplo: ID 3F18H = 0011 1111 0001 1000

Teremos :

2000H	xxxxxxxxxxxx0011
2001H	xxxxxxxxxxxx1111
2002H	xxxxxxxxxxxx0001
2003H	xxxxxxxxxxxx1000

17.1-Gravação dos valores desejados nas posições de ID.

O usuário não precisa se preocupar como estes bits são gravados na CPU, pois os gravadores já providenciam um campo para escolha do ID, gravando-os automaticamente ou não conforme desejo do usuário.

18-O SOFTWARE do PIC 16F84

O conjunto de instruções do PIC 16F84 é formado por 35 instruções, todas formadas por apenas uma palavra de 14 bits.

Nesta palavra esta o código da função a ser executada, além dos parâmetros necessários, como constantes, registros, bits.....

Neste capítulo vamos apresentar o conjunto de instruções de forma resumida, no seguinte formato:

INSTRUÇÃO significado Explicação: Flags (bits) afetados: Exemplo:

Podemos dividir o conjunto de instruções em três tipos básicos:

A) Operações de byte com registros
B) Operações de bit com registros
C) Operações com constantes e de controle

As seguintes convenções serão utilizadas, pois são reconhecidas pelos compiladores MPASM e MPASMWIN.

Campo	Descrição
f	registro entre 0 e 127 (0 à 7FH)
w ou W	registro W
b	bit utilizado pela operação (de 0 a 7)
k	constante ou label
d	destino do resultado se d = 0, o resultado é armazenado em W se d = 1, o resultado é armazenado no próprio registro indicado na operação d = 1 é o padrão quando não indicado

O formato do programa segue o seguinte padrão:

label: operação (opcode) operando(s) ; comentários

O campo label é facultativo e indica posições particulares do programa.
O campo opcode sempre existe e indica qual a operação a ser realizada.
O campo operando existirá se for necessário à instrução.
Após o ponto e vírgula tudo será ignorado pelo compilador.

18.1-Resumo do conjunto de instruções

Temos na figura 18.1 uma tabela resumida das instruções, e no item 18.3 iniciamos o estudo de cada instrução em particular.

INSTRUÇÃO	OPERANDO	DESCRIÇÃO	CICLOS	BITS AFETADOS
OPERAÇÕES DE BYTE COM REGISTROS				
ADDWF	f, d	Soma w e f	1	C,DC, Z
ANDWF	f, d	AND entre w e f	1	Z
CLRF	f	Zera f	1	Z
CLRW		Zera w	1	Z
COMF	f, d	Complementa f	1	Z
DECF	f, d	Decrementa f	1	Z
DECFSZ	f, d	Decrementa f, pula se f = 0	1(2)	-
INCF	f, d	Incrementa f	1	Z
INCFSZ	f, d	Incrementa f, pula se f = 0	1(2)	-
IORWF	f, d	OR entre w e f	1	Z
MOVF	f, d	Move f	1	Z
MOVWF	f	Move w para f	1	-
NOP		Nenhuma operação	1	-
RLF	f, d	Roda a esquerda pelo carry	1	C
RRF	f, d	Roda a direita pelo carry	1	C
SUBWF	f, d	Subtrai w de f	1	C, DC Z
SWAPF	f, d	Troca nibles em f	1	-
XORWF	f, d	XOR entre w e f	1	Z
OPERAÇÕES DE BIT COM REGISTROS				
BCF	f, b	Zera bit b em f	1	-
BSF	f, b	Seta bit b em f	1	-
BTFSC	f, b	Se bit b em f = 0, pula	1(2)	-
BTFSS	f, b	Se bit b em f = 1, pula	1(2)	-
OPERANDO COM CONSTANTES DE CONROLE				
ADDLW	K	Soma w e k	1	C, DC, Z
ANDLW	K	AND entre w e k	1	Z
CALL	K	Chama sub-rotina	2	-
CLRWDT		Zera o timer do WATCH DOG	1	TO\, PD\
GOTO	K	Desvia para o label k	1	-
IORLW	K	OR entre w e k	1	Z
MOVLW	K	W = k	1	-
RETFIE		Retorna da interrupção	2	-
RETLW	K	Retorna con w = k	2	-
RETURN		Retorna da sub-rotina	2	-
SLEEP		Entra no modo SLEEP	1	TO\, PD\
SUBLW	K	Subtrai k de w	1	C, DC, Z
XORLW	k	XOR entre w e k	1	Z

Figura 18.1-Conjunto de instruções.

Sempre vale a pena recordar que cada instrução será executada em apenas 1 ciclo de máquina (1 us a 4 MHz), exceto aquelas que alteram o PC, indicadas como 1(2) ou 2 ciclos.

As instruções indicadas 1(2) levam 2 ciclos se o teste resultar verdadeiro. Maiores detalhes serão apresentados nas instruções em particular.

18.2-Considerações sobre as constantes

Para o compilador, as constantes têm o seguinte formato:

Constante decimal:	
d 'valor' ou D 'valor'	exemplo: 20 decimal = d'20' ou D'20'
Constante binária:	
B'xxxxxxxx'	exemplo:01010101 = B'01010101'
Constante hexadecimal:	
0x'valor ou valorH	exemplo: 12 hexa = 0x12 ou 12H, A0 hexa = 0A0H
Obs: a constante hexadecimal iniciada por letras (A-F) deve ser precedida de 0.	

O compilador tem como padrão valores hexadecimais, logo se não indicarmos o tipo de constante o compilador assumirá hexadecimal.

Exemplo:

```
movlw 20H
movlw 20          ;para o compilador este 20 é hexadecimal.
```

Em nossos exemplos redefiniremos as constantes para sistema decimal.

18.3-Conjunto de instruções detalhado e comentado

A partir de agora, vamos estudar cada instrução em particular, mostrando seu funcionamento e fornecendo um exemplo.

Diferente da figura 18.1, onde colocamos as instruções por tipo, aqui as mesmas serão classificadas em ordem alfabética, para facilitar ao leitor consultar futuras.

Quando formos nos referir a um registro 'f' nos exemplos, seu endereço será assim indicado:

Registro f no endereço 12H → f (12H)

Existem instruções que permitem escolher se o resultado será salvo em w ou no próprio registro indicado na mesma (veja instrução COMF).

Outro ponto importante: instruções que movem valores na realidade apenas movem uma copia do valor, permanecendo inalterado o registro original (veja a instrução MOVWF).

ADDLW k	Somar a constante k a w(w=w+k)
Explicação	A constante k será somada ao registro w e o valor resultante será escrito em w.(0 <= k <= 255)
Flags (bits) afetados	C, DC, Z
exemplo	Temos w = 20H, a instrução ADDLW 12H Resultará em w = 32H

ADDWF f, d	Somar o valor em w e o valor em f
Explicação	O valor presente em w será somado ao valor presente no registro f Se d = 1, o resultado será salvo em w Se d = 0, o resultado será salvo em f
Flags (bits) afetados	C, DC, Z
exemplo	Temos w = 10H e f = 12H, a instrução ADDWF 12H, 0 ;Resultará em w = 22H a instrução ADDWF 12H ;Resultará em f (12H)= 22H

ANDLW k	AND entre o valor em w e a constante k (w = w AND k)
Explicação	Com o valor presente em w será efetuada uma operação lógica AND, bit a bit, com o valor em k. o resultado será salvo em W.
Flags (bits) afetados	Z
exemplo	Temos w = 55H, a instrução ANDLW 33H,0 resultará em w = 11H w = 55H = 01010101 & 33H = 00110011 00010001 = 11H

ANDWF f, d	AND entre o valor em w e o valor em f (d = w AND f)
Explicação	Com o valor presente em w será efetuada uma operação lógica AND, bit a bit com o valor em f Se d = 0 o resultado será salvo em w Se d = 1 o resultado será salvo em f
Flags (bits) afetados	Z
exemplo	Temos w = F5H e f (20H) = 12H a instrução ANDWF 20H,0 resultará em w = 10H a instrução ANDWF 20H resultará em f(20H) = 10H w = F5H = 01010101 & f(20H) = 12H = 00110011 00010000 = 10H

BCF f, b	Zera o bit ‘b’ no registro f (bit clear f) 0 <= f <= 7
Explicação	O bit indicado por ‘b’ será zerado no registro ‘f ’ indicado
Flags (bits) afetados	-
exemplo	Temos f(25H) = C9H = 1100**1**001 A instrução BCF 25H, 3 resultará em F(25H) = C1H = 1100**0**001

BSF f, b	Seta o bit 'b' no registro f (bit clear f) 0 <= f <= 7
Explicação	O bit indicado por 'b' será zerado no registro 'f ' indicado
Flags (bits) afetados	-
exemplo	Temos f(25H) = C9H = 110**0**1001 A instrução BCF 25H, 4 resultará em F(25H) = C1H = 110**1**0001

BTFSC f, b	Pula a instrução seguinte se o bit 'b' no registro f for 0. (bit test, skip if clear)
Explicação	O bit indicado por 'b' no registro 'f' será verificado. Se for '0' a próxima instrução será ignorada Se for '1' a próxima instrução será executada normalmente
Flags (bits) afetados	-
IMPORTANTE: Esta configuração levará 2 ciclos de máquina se o bit for 0, pois a instrução após esta será executada como NOP	
exemplo	Temos f(10H) = 22H = 0010**0**010 BTFSC 10H, 3 ;testa o bit 3 (esta em 0) GOTO SETADO ;se estivesse em 1 executaria este goto BSF 10H, 3 ;como esta em 0, executa esta instrução e ;escreve 1 em 10H, 3

BTFSS f, b	Pula a instrução seguinte se o bit 'b' no registro f for 1. (bit test, skip if set)
Explicação	O bit indicado por 'b' no registro 'f' será verificado. Se for '1' a próxima instrução será ignorada Se for '0' a próxima instrução será executada normalmente
Flags (bits) afetados	-
IMPORTANTE: Esta configuração levará 2 ciclos de máquina se o bit for 1, pois a instrução após esta será executada como NOP	
exemplo	Temos f(10H) = 55H = 010**1**0101 BTFSC 22H, 4 ;testa o bit 4 (esta em 1) GOTO ZERADO ;se estivesse em 0 executaria este goto BSF 10H, 3 ;como esta em 1, executa esta instrução e ;escreve 0 em 22H, 4

CALL k	Chamada de sub-rotina (sub rotina k)
Explicação	O endereço PC+1 (próxima instrução) é salvo no STACK. Então o programa desvia para o endereço k.
Flags (bits) afetados	-
IMPORTANTE: Esta é uma instrução de 2 ciclos de máquina	
exemplo	No endereço 20H o programa chama a sub-rotina em 50H. 20H: CALL 50H ;o endereço 21H é salvo no stack e o PC ; é carregado com o endereço 50H 50H RETURN ;volta ao endereço 21H 21H MOVLW 10H ;Ao voltar continua a execução normalmente

CLRF f	Zera o registro f
Explicação	O registro f passará a ter o valor 0 e o flag de zero (Z) será setado
Flags (bits) afetados	Z
exemplo	Temos f(25H) = 09H A instrução CLRF 25H resultará em F(25H) = 0

CLRW	Zera o registro W
Explicação	O registro W passará a ter o valor 0 e o flag de zero (Z) será setado
Flags (bits) afetados	Z
exemplo	Temos W = 21H A instrução CLRW resultará em W = 0

CLRWDT	Reseta o WATCH DOG
Explicação	O registro WATCH DOG será zerado, assim como o registro do prescaler (se este estiver direcionado para o WATCH DOG).
Flags (bits) afetados	TO\, PD\
IMPORTANTE: Se o WATCH DOG estiver habilitado, esta instrução deverá ser executada de tempos em tempos pelo programa, para evitar o time-out do registro do WATCH DOG e conseqüentemente o reset do sistema.	
exemplo	O registro do WATCH DOG esta contando. Após a instrução CLRWDT o registro do WATCH DOG voltará a 0, e se estiver usando prescaler, seu registro também será zerado

COMF f, d	Os bits do registro f são complementados (invertidos)
Explicação	Cada bit do registro f será invertido. Se for 0 ficará 1 e vice-versa Se d = 0, o resultado será salvo em w Se d = 1, o resultado será salvo em f
Flags (bits) afetados	Z
exemplo	Temos f(21H) = 5AH = 01011010 A instrução COMF 21H resultará em f(21H) = A5H = 10100101 A instrução COMF 21H,0 resultará em w = A5H = 10100101 Neste ultimo caso , o resultado foi armazenado em W, mas o registro f permaneceu inalterado.

DECF f, d	Diminui em 1 o valor armazenado em f (decrement f)
Explicação	O valor em f será diminuído em 1 unidade. Se d = 0, o resultado será salvo em w Se d = 1, o resultado será salvo em f
Flags (bits) afetados	Z
exemplo	Temos f (1CH) = 24H A instrução DECF 1CH resultará em f (1CH) = 23H A instrução DECF 1CH,0 resultará em w = 23H

DECFSZ f, d	Diminui em 1 o valor armazenado em f e se f = 0, pula a próxima instrução. (decrement f, skip if 0)
Explicação	O valor em f será diminuído em 1 unidade. Se f resultar = 0, a próxima instrução será ignorada. Se f > 0, executa a próxima instrução normalmente. Se d = 0, o resultado será salvo em w Se d = 1, o resultado será salvo em f
Flags (bits) afetados	-
IMPORTANTE: Esta configuração levará 2 ciclos de máquina se f resultar = 0, pois a instrução após esta será executada como NOP	
exemplo	Temos f(0DH) = 01H DCFSZ 0DH ;f(0DH) = F(0DH) – 1 GOTO maior ;se f(0DH) > 0 executa o GOTO GOTO zerado ;resultou em 0 Se usarmos DECFSZ 0DH,0 a operação será w = f(0DH) – 1, mas a lógica de desvio será a mesma.

GOTO k	Desvia para o endereço k
Explicação	O programa é desviado para o endereço indicado por k. É um desvio incondicional, isto é, não depende de nenhum teste
Flags (bits) afetados	-
exemplo	A seqüência executada pelo programa abaixo é a indicada pelos números à esquerda (10H, 11H, 35H, ...) repetindo-se sempre 10H: MOVLW 10H ; w = 10H 11H: GOTO 35H ; desvia para 35H ... ... 35H: CLRW ; W = 0 36H: GOTO 10H ; desvia para 10H

INCF f, d	Aumenta em 1 o valor armazenado em f (increment f)
Explicação	f = f +1 se d = 0, o resultado será salvo em w se d = 1, o resultado será salvo em f
Flags (bits) afetados	Z
exemplo	Temos f(1CH) = F0H A instrução INCF 1CH resultará em f(1CH)=F1H A instrução INCF 1CH,0 resultará em w = F1H

INCFSZ f, d	Diminui em 1 o valor armazenado em f e se f = 0, pula a próxima instrução. (increment f, skip if 0)
Explicação	O valor em f será aumentado em 1 unidade. Se f resultar = 0, a próxima instrução será ignorada. Se f > 0, executa a próxima instrução normalmente. Se d = 0, o resultado será salvo em w Se d = 1, o resultado será salvo em f
Flags (bits) afetados	-
IMPORTANTE: Esta configuração levará 2 ciclos de máquina se f resultar = 0, pois a instrução após esta será executada como NOP	
exemplo	Temos f(0DH) = FFH INCFSZ 0DH ;f(0DH) = F(0DH) + 1 GOTO maior ;se f(0DH) > 0 executa o GOTO GOTO zerado ;resultou em 0 Se usarmos INCFSZ 0DH,0 a operação será w = f(0DH) + 1, mas a lógica de desvio será a mesma.

IORLW k	OR entre o valor de w e a constante k (inclusive OR literal with W)
Explicação	Com o valor presente em w será executada uma operação lógica OR, bit a bit com a constante k O resultado será salvo em W
Flags (bits) afetados	Z
exemplo	Temos W = 96H A instrução IORLW 17H resultará em w = 97H W = 96H = 10010110 OR 17H = 00010111 10010111

IORWF f, d	OR entre o valor de w e o valor em f (inclusive OR W with f)
Explicação	Com o valor presente em w será executada uma operação lógica OR, bit a bit com o valor em f Se d = 0, o resultado será salvo em W Se d = 1, o resultado será salvo em f
Flags (bits) afetados	Z
exemplo	Temos w = 5CH e f(20H) = C5H A instrução IORWF 20H,0 resultará em w = DDH A instrução IORWF 20H resultará em f(20H) = DDH W = 5CH = 01011100 OR f(20H) = C5H = 11000101 11011101

MOVLW k	W = constante k (move literal to W)
Explicação	O registro w será carregado com a constante k
Flags (bits) afetados	-
exemplo	Temos w = 11H A instrução MOVLW 33H resultará em w = 33H

MOVF f, d	Move f para d
Explicação	O dado no registro f é copiado para o destino d Se d = 0, o resultado será salvo em w Se d = 1, o resultado será salvo no próprio f
Flags (bits) afetados	Z
IMPOTANTE: reescrever no próprio registro (d = 1) serve para verificar o flag de zero sem alterar o w existente	
exemplo	Temos f(2FH) = 0H e flag Z = 0 A instrução MOVF 2FH resultará em f(2FH) = 0H, e o flag de zero Z = 1. (apenas testamos o zero). A instrução MOVF 2FH,0 resultará em w = 0H , e flag Z = 1

MOVWF	Move w para f
Explicação	O dado carregado em w é copiado para o registro f
Flags (bits) afetados	-
exemplo	Temos w = 78H e f(29H) = 10H A instrução MOVWF 29H resultará em f(29H) = 78H W permanece inalterado

NOP	Nenhuma operação
Explicação	Esta instrução não executa nenhuma operação na CPU, apenas perde 1 ciclo de máquina.
Flags (bits) afetados	-
exemplo	10H MOVLW 34H ;W = 34H 11H NOP ;perde 1 ciclo 12H MOVF 12H,0 ;W = f(12H)

OPTION	Carrega o registro OPTION
Explicação	Esta instrução faz OPTION = W
Flags (bits) afetados	-
NÃO UTILIZAR ESTA INSTRUÇÃO PARA MANTER COMPATIBILIDADE COM VERSÕES FUTURAS Esta implementada para permitir compatibilidade co a família 16C5X OPTION é um registro endereçável	

RETFIE	Retorne da interrupção
Explicação	Esta instrução promove o fim da interrupção que está em execução. O endereço de retorno é recuperado do stack e o bit GIE é setado
Flags (bits) afetados	-
IMPORTANTE: Esta é uma instrução de 2 ciclos de máquina	
exemplo	Stack (topo de um total de 8 níveis) = 37H 120H MOVLW 34H ; W = 34H 121H NOP ;perde 1 us a 4 MHz 122H RETFIE ;retorna da interrupção 37H GOTO TESTE ;retorna ao endereço que estava no ;topo do Stack com GIE = 1

RETLW k	Retorne da sub-rotina com w = k
Explicação	Esta instrução promove o fim da sub-rotina que esta em execução, além de fazer w = constante k O endereço de retorno é recuperado do stack
Flags (bits) afetados	-
IMPORTANTE: Esta é uma instrução de 2 ciclos de máquina	
exemplo	Stack (topo de um total de 8 níveis) = 40H 100H MOVLW 10H ; W = 10H 101H RETLW 1 ;retorna mas com w = 1 40H ADDLW 1 ;faz w = 2

RETURN	Retorne da sub-rotina
Explicação	Esta instrução promove o fim da sub-rotina que esta em execução O endereço de retorno é recuperado do stack
Flags (bits) afetados	-
IMPORTANTE: Esta é uma instrução de 2 ciclos de máquina	
exemplo	Stack (topo de um total de 8 níveis) = 40H 100H MOVLW 10H ; W = 10H 101H RETURN ;retorna 40H ADDLW 1 ;faz w = 11H

RLF f, d	Rotaciona à esquerda pelo carry (rotate left f through carry)
Explicação	O valor no registro f é rotacionado em um bit para a esquerda, passando pelo carry. Se d = 0, o resultado será salvo em w Se d = 1, o resultado será salvo no próprio f
Flags (bits) afetados	C
exemplo	f(25H) = 00110011 e C =1 após 10H RLF 25H,1 teremos f(25H) = 01100111 e C = 0

RRF f, d	Rotaciona à direita pelo carry (rotate right f through carry)
Explicação	O valor no registro f é rotacionado em um bit para a esquerda, passando pelo carry. Se d = 0, o resultado será salvo em w Se d = 1, o resultado será salvo no próprio f
Flags (bits) afetados	C
exemplo	f(25H) = 00110011 e C =1 após 10H RRF 25H,1 teremos f(25H) = 10011001 e C = 0

SLEEP	SLEEP
Explicação	A CPU entra no modo SLEEP e o oscilador para , WATCH DOG e prescaler são zerados. TO\ = 1 e PD\ = 0
Flags (bits) afetados	TO\ e PD\
exemplo	100H SLEEP

SUBLW k	Subtrai w da constante k (subtract w from literal)
Explicação	O registro w é subtraído da constante k (complemento de 2). O resultado é salvo em w w = k – w (0<= k <= 255)
Flags (bits) afetados	C, DC, Z
exemplo	Temos w = 1H A instrução SUBLW 2 resultará em w = 2 – 1 = 1 e C = 1 (positivo) Temos w = 3H A instrução SUBLW 2 resultará em w = 2 – 3 = -1 = FFH e C = 0 (negativo)

SUBWF f, d	Subtrai w do registro f (subtract w from f)
Explicação	O registro w é subtraído do registro f (complemento de 2). d = f – w (0<= k <= 255) Se d = 0, o resultado será salvo em w Se d = 1, o resultado será salvo no próprio f
Flags (bits) afetados	C, DC, Z
exemplo	Temos f(20H) = 0 e w = 1 A instrução SUBWF 20H,1 resultará em f(20H) =0 – 1= -1= FFH e C = 0 (negativo). Temos f(20)H=FFH e w = 0 A instrução SUBWF 20H,1 resultará em f(20H) =FFH – 0 = FFH E C = 1 (positivo). Observe que d = 1, logo w não é alterado.

SWAPF f, d	Troca nibles de f
Explicação	O registro f tem seus bits 7 a 4 trocados pelos bits de 3 a 0 Se d = 0, o resultado será salvo em w Se d = 1, o resultado será salvo no próprio f
Flags (bits) afetados	-
exemplo	Temos f(11H) = 25H A instrução SWAPF 11H, 0 resultará em f(11H)= 25H e w = 52H 25H = 0100101 52H = 1010010

TRIS	Carrega o registro TRIS
Explicação	Esta instrução faz TRIS = W
Flags (bits) afetados	-
NÃO UTILIZAR ESTA INSTRUÇÃO PARA MANTER COMPATIBILIDADE COM VERSÕES FUTURAS Esta implementada para permitir compatibilidade co a família 16C5X TRIS é um registro endereçável pelo programa	

XORWF f, d	XOR entre w e f (exclusive OR W with f)
Explicação	Com o valor presente em w será efetuada uma operação lógica XOR, bit a bit, com o valor f. (bits iguais XOR = 0, bits diferentes XOR = 1) se d = 0, o resultado será salvo em w se d = 1, o resultado será salvo em f
Flags (bits) afetados	Z
exemplo	Temos w = 52H e f(20H) = 52H A instrução XORWF 20H,0 resultará em w = 0 A instrução XORWF 20H resultará em f(20H) = 0 w = 52H = 01010010 XOR f(20H) = 52H = 01010010 00000000

XORLW k	XOR entre w e constante k (exclusive OR literal with W)
Explicação	Com o valor presente em w será efetuada uma operação lógica XOR, bit a bit, com a constante k. (bits iguais XOR = 0, bits diferentes XOR = 1) o resultado será salvo em w
Flags (bits) afetados	Z
exemplo	Temos w = 10101101 A instrução XORLW 3AH,0 resultará em w = 97H w = ADH = 10101101 XOR 3AH = 00111010 10010111

19-Exemplos de hardware e software

Neste capítulo veremos exemplos que buscam iniciar o leitor na programação e no uso do PIC 16F84.

Vamos partir de exemplos bem simples, praticamente sem função operacional especifica, até chegarmos a exemplos mais complexos.

Para todos os programas apresentados vamos utilizar o circuito apresentado na figura 19.1, que poderá ser montada pelo leitor.

Todos os programas aqui apresentados fora compilados no MPASMWIN e vale ressaltar que somente este programa é citado neste curso.

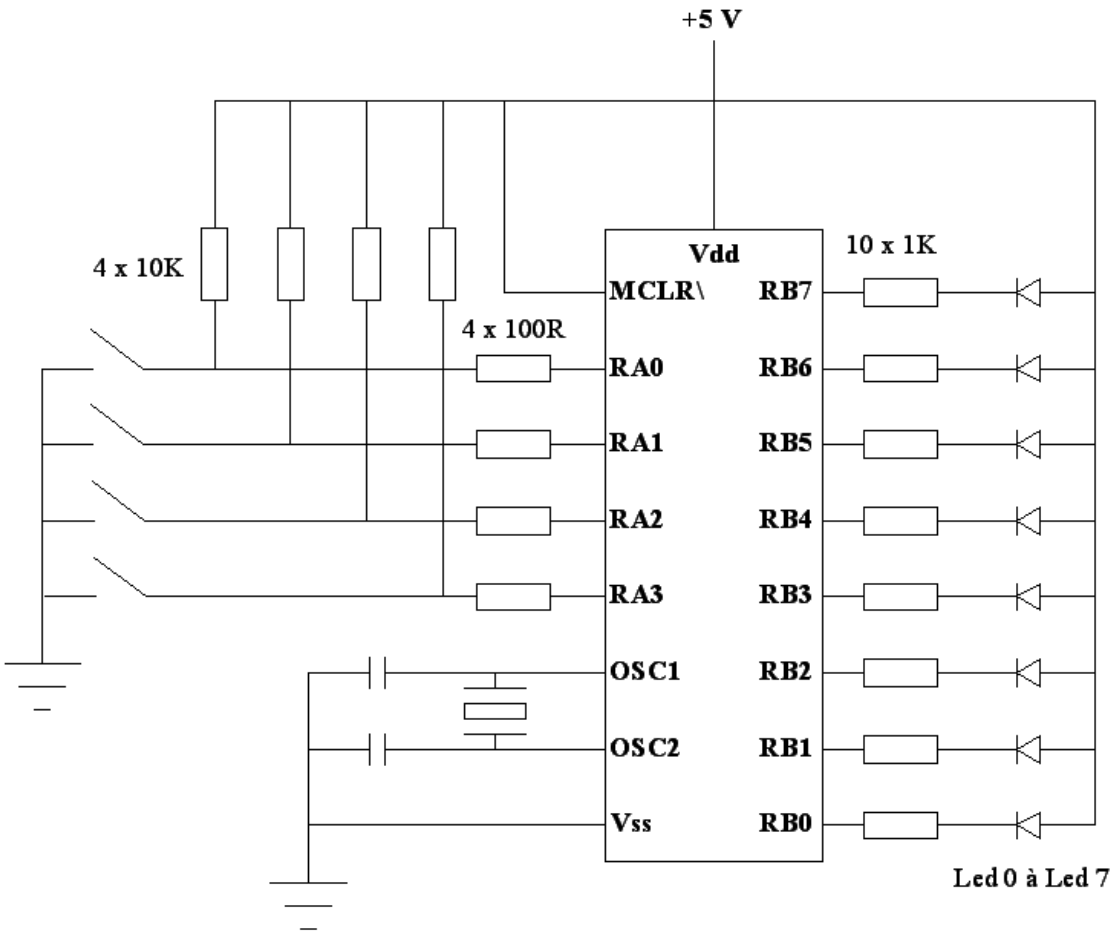


Figura 19.1-circuito dos exemplos

19.1-Características do circuito básico

- PORTB ligado a 8 leds (acendem quando o nível for 0).
- 4 teclas NA (normalmente aberto) ligadas aos pinos RA0 à RA3 (vão a nível 0 ao fechar).
- Pino RA4 disponível para I\O conforme desejo do leitor.
- Clock de 4 MHz com cristal (se o leitor desejar pode montar o circuito da figura 10.1).
- Reset por POR (ao ligar).

19.2-Programa básico para testes

Em todos os programas usaremos o corpo básico apresentado na figura 19.2, que contém p mínimo de instruções e permite ao leitor ter uma visão geral de como escrever seus programas.

```
;=====================================================================
;     Programa padrao para testes
;     Registros  da CPU com letras maiusculas
;=====================================================================
      list p=16F84      ;para qual processador o codigo sera gerado
      radix dec         ;padrao DECIMAL para valores sem identificacao
      include    <P16F84.INC>;anexa arquivo com definicoes do 16F84

;=====================================================================
;Tabela de definições de RAM e constantes

X     equ   0CH         ;define variavel auxiliar X na RAM 0CH
                        ;(primeira posicao)
Y     equ   0DH         ;
W     equ   0           ;facilita a referencia a w quando necessario
;=====================================================================
;memória de programa

      org   0           ;define o trecho a seguir em 000

      goto  inicio      ;desvia para o label "inicio" deixando o
                        ; endereço 004 para a rotina de interrupcao
;=====================================================================
;inicio da rotina de interrupção

      org   4           ;interrupção sempre inicia em 004
                        ;código da interrupção
      retfie                  ;retorna da interrupção
;=====================================================================
;inicio do programa logo após o reset

inicio:
      movlw B'00000000' ;W = ajuste para os bits do INTCON
      movwf INTCON      ;INTCON = W

      bsf   STATUS,RP0  ;seleciona banco 1 para OPTION e TRIS

      movlw B'11011111' ;W = ajuste para os bits do OPTION
      movwf OPTION_REG  ;OPTION = W.Definiu-se o nome OPTION_REG
                        ;para evitar conflito com a instrucao OPTION

      movlw B'11111111' ;W = ajuste para os bits do TRISA
      movwf TRISA       ;TRISA = W; bits= 1 entrada;bits=0 saida

      movwf B'00000000' ;W = ajuste para os bits do TRISB
      movwf TRISB       ;TRISB = W; bits= 1 entrada;bits=0 saída

      bcf   STATUS,RP0  ;retorna ao banco 0 (padrao do Reset)
;=====================================================================
principal:

                        ;codigo do programa

      END               ;fim do programa fonte
```

Figura 19.2-Programa padrão para exemplos do curso.

19.2.1-Considerações sobre o programa padrão

Devemos observar a obrigatoriedade de seguir o padrão indicado no capítulo 18.

Label:	operação	Operando	;comentários
0	1	2	

⬆Margem do texto

Onde: label está na tabulação 0(coluna 1)
Operação está na tabulação 1
Operando está na tabulação 2
Comentários após o "**;**" em qualquer ponto

Os principais itens do programa são:

A) Comentários

Sempre que possível devemos incluir comentários em nossos programas para fácil estudo posterior ou a compreensão por parte de outros leitores.

No programa assembler o comentário fica logo após o "**;**". Tudo que esta após "**;**" é ignorado pelo compilador.

B) list p=

Após o sinal de "**=**" colocamos o nome do processador que será utilizado, pois o compilador é genérico e necessita saber qual o código binário que vai gerar para cada instrução. Por exemplo, para o 16F84 (família 16CXXX) o CLRWDT tem código 0064H e para o 17C42 (17CXXX) o CLRWDT tem código 0004H.

C) radix dec

Informamos para o compilador que números sem indicação de base (item 18.2) deverão ser considerados DECIMAIS, e não hexadecimais conforme o padrão.

D) include<XYZ>

Informamos ao compilador para anexar ao nosso fonte, durante o processo de compilação, o arquivo XYZ (no nosso caso, P16F84.INC).

Como já vimos, o compilador é genérico e não sabe o endereço e outras definições de cada processador. No list informamos o processador e no arquivo XYZ estão as definições dos registros internos, como por exemplo, informa que o registro EEADR está no endereço 09H (no 16C65A o endereço 09H pertence ao PORTE).

E) abe EQU 123

Os EQU´s servem para que possamos atribuir substituições.

No programa padrão definimos duas variáveis em RAM com os nomes X e Y. É muito mais fácil lembrar, durante a elaboração do programa, que o contador de interrupções está em X do que ficar memorizando seu endereço, principalmente num programa de muitas variáveis.

Todos os registros da CPU definido no include nada mais são do que EQU´s para cada um deles.

F) org

O "org" permite ao usuário iniciar um trecho do programa em um endereço especifico na memória de programa.

No exemplo temos org 0 para indicar o endereço do reset e o org 4 para indicar o inicio da interrupção.

Se desejarmos que a instrução indicada pelo label inicio seja alocada na posição 100H, devemos usar

```
        org     100H
inicio: CLRW                ;esta instrução estará no endereço 100H
```

No caso da interrupção, ao invés de org 4 poderíamos usar:

```
        org     0       ;reset
        goto    inicio  ;endereço 000. Todas as instruções têm apenas uma palavra
        nop             ;endereço 001
        nop             ;endereço 002
        nop             ;endereço 003
        goto    interrupção     ;endereço 004. Primeira interrupção da instrução
```

G) END

Sinaliza para o compilador o fim do arquivo fonte.

19.2.2-Desligar o WATCH DOG

IMPORTANTE:

Nos exemplos 1 a 6 o WATCH DOG não está sendo utilizado.

Ao gravar o chip o usuário deve lembrar-se de ajustar o WATCH DOG para ‘off’, caso contrário o chip resetará de tempos em tempos (com o ajuste do OPTION em 11011111 o reset será a cada 2,304 s).

19.3-Exemplo 1- Rotina de tempo baseado nos ciclos de clock.

```
;===============================================================================
;       Programa exemplo 1
;       Registros  da CPU com letras maiúsculas
;       Objetivo:               estudar rotina de tempo baseada em software
;       Funcionamento           pisca Led7 a cada 250 ms
;===============================================================================
        list p=16F84            ;para qual processador o codigo sera gerado
        radix  dec              ;padrão DECIMAL para valores sem identificação
        include<P16F84.INC>     ;anexa arquivo com definições do 16F84

;===============================================================================
;Tabela de definições de RAM e constantes

Led7    equ     7               ;led 7 está em RB7
tempo   equ     0CH             ;variável tempo na RAM 0CH
X       equ     0DH             ;define variavel auxiliar X na RAM 0CH
                                ;(primeira posição)
Y       equ     0EH             ;
W       equ     0               ;facilita a referencia a w quando necessário
;===============================================================================
;memória de programa

        org     0               ;define o trecho a seguir em 000
        goto    inicio          ;desvia para o label "inicio" deixando o
                                ;endereço 004 para a rotina de interrupção
;===============================================================================
;inicio da rotina de interrupção

        org     4               ;interrupção sempre inicia em 004

        nop                     ;código da interrupção

        retfie                  ;retorna da interrupção
;===============================================================================
;inicio do programa logo após o reset

inicio:
        movlw   B'00000000'     ;W = ajuste para os bits do intcon
        movwf   INTCON          ;INTCON = W

        bsf     STATUS,RP0      ;seleciona banco 1 para OPTION e TRIS

        movlw   B'11011111'     ;W = ajuste para os bits do OPTION
        movwf   OPTION_REG      ;OPTION = W.Definiu-se o nome OPTION_REG
                                ;para evitar conflito com a istrução OPTION

        movlw   B'11111111'     ;W=ajuste para od bits do TRISA
        movwf   TRISA           ;TRISA = W; bits= 1 entrada;bits=0 saida

        movwf   B'00000000'     ;W = ajuste para os bits do TRISB
        movwf   TRISB           ;TRISB = W; bits= 1 entrada;bits=0 saída

        bcf     STATUS,RP0      ;retorna ao banco 0 (padrao do Reset)
;===============================================================================
principal:

        bcf     PORTB,Led7      ;fazendo PORTB pino 7=0 acende o led 7
        call    ms250           ;espera 250 ms
        bsf     PORTB,Led7      ;fazendo PORTB pino 7=1 apaga o led 7
        call    ms250           ;espera 250 ms

        goto    principal
;===============================================================================
;rotina de tempo de 250 ms

ms250:                          ;o call para a rotina leva 2 us
        movlw   250             ;+1
        movwf   tempo           ;+1     total1= 4 us

ms250a:
        movlw   248             ;+1
        movwf   X               ;+1     total2=2 us

ms250b:
        nop                     ;1
        decfsz  X               ;1      248 x4us +(1us nop+2us quando decfsz da 0

        goto    ms250b          ;2      da um tempo total3 = 995 us
        decfsz  tempo           ;1      1us pq tempo>0+2us goto da um total4=3 us

        goto    ms250a          ;2      250 x(total2+total3+total4)+2 us
                                ;       quando tempo=0=total5=250.002 us
        return                  ;delay =total1+total5+2us do return = 250008 us
```

```
        END                     ;fim do programa
```

Sobre a rotina de tempo:

$$4 + 250 * (2 + 248*4 + 1 + 2 + 1 + 2) + 2 + 2 = 250.008 \text{ us}$$

O leitor deve estudar atentamente como cada instrução funciona e o tempo decorrido de cada uma para entender a rotina de tempo.

Nem sempre tempos exatos podem ser conseguidos.

19.4-Exemplo 2- Instruções de deslocamento

```
;*****************************************************************************
;       Programa exemplo 2
;       Objetivo:               estudar instruções de deslocamento
;      Funcionamento           pisca os leds 7 a 0 nesta ordem e repete
;                              OS LEDS ACENDEM EM 0. ver figura 19.1
;
;*****************************************************************************
        list p=16F84            ;para qual processador o codigo sera gerado
        radix  dec              ;padrão DECIMAL para valores sem identificação
        include<P16F84.INC>     ;anexa arquivo com definições do 16F84

;*****************************************************************************
;Tabela de definições de RAM e constantes

tempo   equ     0CH             ;variável tempo na RAM 0CH
dt1     equ     0DH             ;define variavel auxiliar dt1 na RAM 0CH
                                ;(primeira posição)
X       equ     0EH             ;define variavel auxiliar X na RAM 0DH
Y       equ     0FH             ;define variavel auxiliar Y na RAM 0EH
W       equ     0               ;facilita a referencia a w quando necessário
;*****************************************************************************
;memória de programa

        org     0               ;define o trecho a seguir em 000
        goto    inicio          ;desvia para o label "inicio" deixando o endereço
                                ;004 para a rotina de interrupção
;*****************************************************************************
;inicio da rotina de interrupção

        org     4               ;interrupção sempre inicia em 004

        nop                     ;código da interrupção

        retfie                  ;retorna da interrupção
;*****************************************************************************
;inicio do programa logo após o reset

inicio:
        movlw   B'00000000'     ;W = ajuste para os bits do intcon
        movwf   INTCON          ;INTCON = W
        clrf    PORTA           ;inicializa PORTA e PORTB,ver itens 7.1.4 e7.2.7
        clrf    PORTB

        bsf     STATUS,RP0      ;seleciona banco 1 para OPTION e TRIS

        movlw   B'11011111'     ;W = ajuste para os bits do OPTION
        movwf   OPTION_REG      ;OPTION = W.Definiu-se o nome OPTION_REG
                                ;para evitar conflito com a istrução OPTION

        movlw   B'11111111'     ;W=ajuste para od bits do TRISA
        movwf   TRISA           ;TRISA = W; bits= 1 entrada;bits=0 saida

        movwf   B'00000000'     ;W = ajuste para os bits do TRISB
        movwf   TRISB           ;TRISB = W; bits= 1 entrada;bits=0 saída

        bcf     STATUS,RP0      ;retorna ao banco 0 (padrao do Reset)
;*****************************************************************************
principal:
        movlw   255             ;W=11111111
        movwf   X               ;x=w=11111111
        bcf     STATUS,C        ;bit carry=0

repete:
        rrf     X               ;desloca o valor em X para a direita
                                ;o bit 0 vai ao carry e o carry vai ao bit 7
        btfss   STATUS,C        ;se C=1 pula a próxima instrução
        goto    principal       ;quando carry=0 acabou de deslocar 8 vezes

        movf    X,W             ;W=valor em X que está escrito no PORTB
                                ;ver instrução MOVF f,d
escreve:
        movwf   PORTB           ;PORTB=W=X
        call    ms250           ;espera 250 ms
        goto    repete          ;se carry=1    ainda não deslocou 8 vezes
                                ;efetua novo deslocamento
;*****************************************************************************
;rotina de tempo de 250 ms

ms250:                          ;o call para a rotina leva 2 us
        movlw   250             ;+1
        movwf   tempo           ;+1    total1= 4 us

ms250a:
```

```
        movlw   248             ;+1
        movwf   dt1             ;+1     total2=2 us

ms250b:
        nop                     ;1
        decfsz  dt1             ;1      248 x4 us + (1us nop + 2 us quando decfsz da 0

        goto    ms250b          ;2      da um tempo total3 = 995 us
        decfsz  tempo           ;1      1 us pq tempo > 0+2 us goto da um total4=3 us

        goto    ms250a          ;2      250 x(total2+total3+total4)+2 us
                                ;       quando tempo=0=total5=250.002 us
        return                  ;delay=total 1+total5+2 us do return = 250008 us

        END                     ;fim do programa
```

19.5-Exemplo 3-Leitura de teclas

```
;*****************************************************************************
;       Programa exemplo 3
;       Objetivo:              estudar varredura de teclas
;      Funcionamento          tecla S0 solta: pisca leds 7 a 0 nesta ordem
;                             tecla S0 pressionada: pisca de 0 a 7
;*****************************************************************************
        list p=16F84           ;para qual processador o codigo sera gerado
        radix  dec             ;padrão DECIMAL para valores sem identificação
        include<P16F84.INC>    ;anexa arquivo com definições do 16F84

;*****************************************************************************
;Tabela de definições de RAM e constantes

tempo   equ     0CH            ;variável tempo na RAM 0CH
dt1     equ     0DH            ;define variavel auxiliar dt1 na RAM 0CH
                               ;(primeira posição)
X       equ     0EH            ;define variavel auxiliar X na RAM 0DH
Y       equ     0FH            ;define variavel auxiliar Y na RAM 0EH
W       equ     0              ;facilita a referencia a w quando necessário
S0      equ     0              ;S0 está ligada ao RA0(PORTA bit 0)
;*****************************************************************************
;memória de programa

        org     0              ;define o trecho a seguir em 000
        goto    inicio         ;desvia para o label "inicio" deixando o endereço
                               ;004 para a rotina de interrupção
;*****************************************************************************
;inicio da rotina de interrupção

        org     4              ;interrupção sempre inicia em 004

        nop                    ;código da interrupção

        retfie                 ;retorna da interrupção
;*****************************************************************************
;inicio do programa logo após o reset

inicio:
        movlw   B'00000000'    ;W = ajuste para os bits do intcon
        movwf   INTCON         ;INTCON = W
        clrf    PORTA          ;inicializa PORTA e PORTB,ver itens 7.1.4 e7.2.7
        clrf    PORTB

        bsf     STATUS,RP0     ;seleciona banco 1 para OPTION e TRIS

        movlw   B'11011111'    ;W = ajuste para os bits do OPTION
        movwf   OPTION_REG     ;OPTION = W.Definiu-se o nome OPTION_REG
                               ;para evitar conflito com a istrução OPTION

        movlw   B'11111111'    ;W=ajuste para od bits do TRISA
        movwf   TRISA          ;TRISA = W; bits= 1 entrada;bits=0 saida

        movwf   B'00000000'    ;W = ajuste para os bits do TRISB
        movwf   TRISB          ;TRISB = W; bits= 1 entrada;bits=0 saída

        bcf     STATUS,RP0     ;retorna ao banco 0 (padrao do Reset)
;*****************************************************************************
principal:
        movlw   255            ;W=11111111
        movwf   X              ;x=w=11111111
        bcf     STATUS,C       ;bit carry=0

repete:
        btfss   PORTA,S0       ;Tecla S0 está em RA0. Se for 1 pula próxima instrução
        rlf     X              ;desloca o valor em X para a direita
                               ;S0=0 desloca o valor em X para a esquerda
                               ;o bit 0 vai ao carry e o carry vai ao bit 7
        btfsc   PORTA,S0       ;se S0=1 este teste serve como um nop
        rrf     X              ;se está em 0, já fez o rlf então pula
        btfss   STATUS,C       ;se C=1 pula a próxima instrução
        goto    principal      ;quando carry=0 acabou de deslocar 8 vezes

        movf    X,W            ;W=valor em X que está escrito no PORTB
                               ;ver instrução MOVF f,d
escreve:
        movwf   PORTB          ;PORTB=W=X
        call    ms250          ;espera 250 ms
        goto    repete         ;se carry=1    ainda não deslocou 8 vezes
                               ;efetua novo deslocamento
;*****************************************************************************
;rotina de tempo de 250 ms

ms250:                         ;o call para a rotina leva 2 us
```

```
        movlw   250             ;+1
        movwf   tempo           ;+1     total1= 4 us

ms250a:
        movlw   248             ;+1
        movwf   dt1             ;+1     total2=2 us

ms250b:
        nop                     ;1
        decfsz  dt1             ;1      248 x4 us + (1us nop + 2 us quando decfsz da 0

        goto    ms250b          ;2      da um tempo total3 = 995 us
        decfsz  tempo           ;1      1 us pq tempo > 0+2 us goto da um total4=3 us

        goto    ms250a          ;2      250 x(total2+total3+total4)+2 us
                                ;       quando tempo=0=total5=250.002 us
        return                  ;       delay=total 1+total5+2 us do return = 250008 us

        END                     ;       fim do programa
```

19.6-Exemplo 4-incrementa/decrementa valores em RAM.

```
;*******************************************************************************
;       Programa exemplo 4
;       Objetivo:             incrementar/decrementar posução de memória pelas teclas
;                             S0 e S1
;      Funcionamento         posição valor varia entre 0 e 63
;                             inicia o valor em 32=00100000
;                             S0→valor=valor –1,se valor >0
;                             S1→valor=valor +1, se valor <63
;*******************************************************************************
       list p=16F84           ;para qual processador o codigo sera gerado
       radix  dec             ;padrão DECIMAL para valores sem identificação
       include<P16F84.INC>    ;anexa arquivo com definições do 16F84

;*******************************************************************************
;Tabela de definições de RAM e constantes

tempo  equ    0CH             ;variável tempo na RAM 0CH
dt1    equ    0DH             ;define variavel auxiliar dt1 na RAM 0CH
                              ;(primeira posição)
X      equ    0EH             ;define variavel auxiliar X na RAM 0DH
Y      equ    0FH             ;define variavel auxiliar Y na RAM 0EH
Valor  equ    10H
W      equ    0               ;facilita a referencia a w quando necessário
S0     equ    0               ;S0 está ligada ao RA0(PORTA bit 0)
S1     que    1               ;S1 esta ligada a0 RA1(PORTA bit 1)
;*******************************************************************************
;memória de programa

       org    0               ;define o trecho a seguir em 000
       goto   inicio          ;desvia para o label "inicio" deixando o endereço
                              ;004 para a rotina de interrupção
;*******************************************************************************
;inicio da rotina de interrupção

       org    4               ;interrupção sempre inicia em 004

       nop                    ;código da interrupção

       retfie                 ;retorna da interrupção
;*******************************************************************************
;inicio do programa logo após o reset

inicio:
       movlw  B'00000000'     ;W = ajuste para os bits do intcon
       movwf  INTCON          ;INTCON = W
       clrf   PORTA           ;inicializa PORTA e PORTB,ver itens 7.1.4 e7.2.7
       clrf   PORTB

       bsf    STATUS,RP0      ;seleciona banco 1 para OPTION e TRIS

       movlw  B'11011111'     ;W = ajuste para os bits do OPTION
       movwf  OPTION_REG      ;OPTION = W.Definiu-se o nome OPTION_REG
                              ;para evitar conflito com a istrução OPTION

       movlw  B'11111111'     ;W=ajuste para os bits do TRISA
       movwf  TRISA           ;TRISA = W; bits= 1 entrada;bits=0 saida

       movwf  B'00000000'     ;W = ajuste para os bits do TRISB
       movwf  TRISB           ;TRISB = W; bits= 1 entrada;bits=0 saída

       bcf    STATUS,RP0      ;retorna ao banco 0 (padrao do Reset)
;*******************************************************************************
principal:                    ;inicio do programa principal
       movlw  32              ;W=00100000
       movwf  valor           ;valor=32
       comf   valor,W         ;complementa os bits em valor e salva em W
       movwf  PORTB           ;escreve W nos leds

vetec:
       btfss  PORTA,S0        ;se PORTA,0=1(S0 aberta)pula
       goto   S0fech          ;se PORTA,0=0(S0 fechada)desvia

S0Ab:
       btfsc  PORTA,S1        ;se aberta, testa S1. Se PORTA,1=0(S1 fechado) pula
       goto   vetec           ;S1 também aberta, repeta a espera de teclas

S1fech:                       ;chegou aqui porque S1 está fechada
       Incf   valor           ;valor=valor+1
       btfss  valor,6         ;se valor,bit 6=1 passou de 63
       goto   S1ok            ;aceitou S1
       decf   valor           ;não aceitou, volta

S1ok:
```

```
        Comf    valor,W         ;complementa os bits em valor mas salva em W
        movwf   PORTB           ;escreve nos leds
        call    ms250           ;250 ms entre repetição de teclas
        goto    vetec           ;volta para ver teclas novamente

S0fech:                         ;chegou aqui porque S0 está fechada
        decf    valor           ;valor=valor-1
        btfss   valor,7         ;se valor,bit7=1,de 0 foi para 255
        goto    S0ok            ;valo >=0
        incf    valor           ;volta ao normal

S0ok:
        comf    valor,W         ;complementa os bits em valor mas salva em W
        movwf   PORTB           ;escreve nos leds
        call    ms250           ;250 ms entre repetição de teclas
        goto    vetec           ;volta para ver teclas
;================================================================================
;rotina de tempo de 250 ms

ms250:                          ;o call para a rotina leva 2 us
        movlw   250             ;+1
        movwf   tempo           ;+1     total1= 4 us

ms250a:
        movlw   248             ;+1
        movwf   dt1             ;+1     total2=2 us

ms250b:
        nop                     ;1
        decfsz  dt1             ;1      248 x4 us + (1us nop + 2 us quando decfsz da 0

        goto    ms250b          ;2      da um tempo total3 = 995 us
        decfsz  tempo           ;1      1 us pq tempo > 0+2 us goto da um total4=3 us

        goto    ms250a          ;2      250 x(total2+total3+total4)+2 us
                                ;       quando tempo=0=total5=250.002 us
        return                  ;       delay=total 1+total5+2 us do return = 250008 us

        END                     ;       fim do programa
```

19.7-Exemplo 5-Leitura e escrita na EEPROM

```
;================================================================================
;       Programa exemplo 5
;       Objetivo:              testar leitura e escrita na EEPROM
;      Funcionamento          posição valor varia entre 0 e 63
;                              inicia o valor em 32=00100000
;                              S0→valor=valor –1,se valor >0
;                              S1→valor=valor +1, se valor <63
;                              S2→se pressionado, salva “valor+128” no endereço “valor”
;                              S3→enquanto estiver pressionado, lê o dado do endereço
;                              “valor” e escreve nos leds.
;================================================================================
       list p=16F84           ;para qual processador o codigo sera gerado
       radix  dec             ;padrão DECIMAL para valores sem identificação
       include<P16F84.INC>    ;anexa arquivo com definições do 16F84

;================================================================================
;Tabela de definições de RAM e constantes

tempo  equ    0CH             ;variável tempo na RAM 0CH
dt1    equ    0DH             ;define variavel auxiliar dt1 na RAM 0CH
                              ;(primeira posição)
X      equ    0EH             ;define variavel auxiliar X na RAM 0DH
Y      equ    0FH             ;define variavel auxiliar Y na RAM 0EH
Valor  equ    10H
lido   equ    11H             ;endereço do dado lido da EEPROM
gravado equ   12H             ;endereço do dado gravado na EEPROM
W      equ    0               ;facilita a referencia a w quando necessário
S0     equ    0               ;S0 está ligada ao RA0(PORTA bit 0)
S1     equ    1               ;S1 esta ligada a0 RA1(PORTA bit 1)
S2     equ    2               ;S2 esta ligada a0 RA2(PORTA bit 2)
S3     equ    3               ;S3 esta ligada a0 RA3(PORTA bit 3)
;================================================================================
;inicio do programa
       org    0               ;define  o inicio do trecho a seguir em 000
inicio:
       movlw  B'00000000'     ;W = ajuste para os bits do intcon
       movwf  INTCON          ;INTCON = W
       clrf   PORTA           ;inicializa PORTA e PORTB,ver itens 7.1.4 e7.2.7
       clrf   PORTB

       bsf    STATUS,RP0      ;seleciona banco 1 para OPTION e TRIS

       movlw  B'11011111'     ;W = ajuste para os bits do OPTION
       movwf  OPTION_REG      ;OPTION = W.Definiu-se o nome OPTION_REG
                              ;para evitar conflito com a istrução OPTION

       movlw  B'11111111'     ;W=ajuste para os bits do TRISA
       movwf  TRISA           ;TRISA = W; bits= 1 entrada;bits=0 saida

       movwf  B'00000000'     ;W = ajuste para os bits do TRISB
       movwf  TRISB           ;TRISB = W; bits= 1 entrada;bits=0 saída

       bcf    STATUS,RP0      ;retorna ao banco 0 (padrao do Reset)
;================================================================================
principal:                    ;inicio do programa principal
       movlw  32              ;W=00100000
       movwf  valor           ;valor=32
       comf   valor,W         ;complementa os bits em valor e salva em W
       movwf  PORTB           ;escreve W nos leds

vetec:
       btfss  PORTA,S0        ;se PORTA,0=1(S0 aberta)pula
       goto   S0fech          ;se PORTA,0=0(S0 fechada)desvia

S0Ab:
       btfsc  PORTA,S1        ;se aberta, testa S1. Se PORTA,1=0(S1 fechado) pula
       goto   veS2S3          ;S1 também aberta, repeta a espera de teclas

S1fech:                       ;chegou aqui porque S1 está fechada
       Incf   valor           ;valor=valor+1
       btfss  valor,6         ;se valor,bit 6=1 passou de 63
       goto   S1ok            ;aceitou S1
       decf   valor           ;não aceitou, volta

S1ok:
       Comf   valor,W         ;complementa os bits em valor mas salva em W
       movwf  PORTB           ;escreve nos leds
       call   ms250           ;250 ms entre repetição de teclas
       goto   vetec           ;volta para ver teclas novamente

S0fech:                       ;chegou aqui porque S0 está fechada
       decf   valor           ;valor=valor-1
       btfss  valor,7         ;se valor,bit7=1,de 0 foi para 255
```

```
        goto    S0ok            ;valo >=0
        incf    valor           ;volta ao normal

S0ok:
        comf    valor,W         ;complementa os bits em valor mas salva em W
        movwf   PORTB           ;escreve nos leds
        call    ms250           ;250 ms entre repetição de teclas
        goto    vetec           ;volta para ver teclas

veS2S3:                         ;vê se salva ou lê na EEPROM
        btfss   PORTA,S2        ;vê se S2(salvar) está solta(s2=1)
        goto    salvar          ;não está, vai salvar
        btfss   PORTA,S3        ;vê se S3(ler) está solta(S3=1)
        goto    ler             ;não está, vai ler o dado

salvar:
        bcf     STATUS,RP0      ;
        movf    valor,W         ;W=valor
        movwf   EEADR           ;endereço=W=Valor
        iorl    80H             ;W=W OR 80H=(seta bit 7)
        movwf   EEDATA          ;dado=valor com bit 7=1
        bsf     STATUS,RP0      ;vai para o banco 1
        bsf     EECON1,WREN     ;habilita a escrita,interrupção já desabilitada
        movlw   55H             ;W=55H=1010101
        movwf   EECON2          ;move o valor de W para registro EECON2
        movlw   0AAH            ;W=0AAH=10101010.Obs- valores  hexadecimais começados com
                                ;letras devem ter um “0” na frente
        movwf   EECON2          ;EECON2 = W
        bsf     EECON1,WR       ;inicia o processo de escrita

espera2:
        btfss   EECON1,EEIF     ;vê se finalizou a escrita
        goto    espera2         ;se EEIF = 0, então ainda não terminou a escrita
        bcf     EECON1,WREN     ;desabilita a escrita na EEPROM
        bcf     EECON1,EEIF     ;zera EECON1,EEIF
        bcf     STATUS,RP0      ;volta para o banco 0
        goto    vetec           ;

ler:
        bcf     STATUS,RP0      ;vai para banco 0
        movf    valor,W         ;valor=W
        movwf   EEADR           ;EEADR= W = endereço
        bsf     STATUS,RP0      ;volta ao banco 1
        bsf     EECON1,RD       ;realiza leitura(salva em EEDATA)
        bcf     STATUS,RP0      ;volta ao banco 0
        movf    EEDATA,W        ;move valor de EEDATA para W
        movwf   lido            ;move o valor de W para a variável “lido”(11H)
        comf    lido,W          ;complementa(inverte) o valor armazenado em “lido”(11H)
        movwf   PORTB           ;escreve nos leds o dado lido da EEPROM

espera1:
        call    ms250           ;aguarda 250 ms
        btfss   PORTA,3         ;vê se S3 esta fechada
        goto    espera1         ;está então repete leitura
        comf    valor           ;W=valor complementado
        movwf   PORTB           ;escreve “valor” nos leds
;================================================================================
;rotina de tempo de 250 ms

ms250:                          ;o call para a rotina leva 2 us
        movlw   250             ;+1
        movwf   tempo           ;+1     total1= 4 us

ms250a:
        movlw   248             ;+1
        movwf   dt1             ;+1     total2=2 us

ms250b:
        nop                     ;1
        decfsz  dt1             ;1      248 x4 us + (1us nop + 2 us quando decfsz da 0

        goto    ms250b          ;2      da um tempo total3 = 995 us
        decfsz  tempo           ;1      1 us pq tempo > 0+2 us goto da um total4=3 us

        goto    ms250a          ;2      250 x(total2+total3+total4)+2 us
                                ;       quando tempo=0=total5=250.002 us
        return                  ;       delay=total 1+total5+2 us do return = 250008 us

        END                     ;       fim do programa
```

19.7.1-Comentários sobre o programa

Ao resetar, os leds indicam o endereço 32(00100000).

Pressionando S0, diminuímos o endereço até chegar a 0.

Pressionando S1, aumentamos o endereço até o limite (a EEPROM tem apenas 64 bytes, de 0 a 63).

Ao pressionar S2, gravamos no endereço indicado binariamente nos leds o dado "endereço" OR 128 (setamos o bit 7).

Exemplo: se o endereço for 10H(16 decimal=00010000) o dado gravado será 90H (144 decimal=10010000).

Com os leds indicando o endereço 10H(00010000) se pressionarmos S4 os mesmos mostrarão o valor 90H(10010000) que estava na EEPROM.

Faça um teste: após gravar alguns endereços, desligue e ligue a alimentação e então leia os endereços anteriormente gravados e verifique se estão corretos.

IMPORTANTE: quando gravamos um programa novo no PIC 16F84 os dados já gravados na EEPROM de dados não são alterados.

19.8-Exemplo 6-Interrupção do TIMER 0 gerando onda quadrada com T=200 us (interrupção a cada 100 us).

```
;*****************************************************************************
;       Programa exemplo 6
;       Objetivo:             estudar interrupções
;      Funcionamento          complementa o PORTB a cada 100 us
;                             PORTB começa = 01010101. a cada interrupção temos
                              PORTB=01010101 e depois 10101010 repetidamente.
;*****************************************************************************
       list p=16F84           ;para qual processador o codigo sera gerado
       radix  dec             ;padrão DECIMAL para valores sem identificação
       include<P16F84.INC>    ;anexa arquivo com definições do 16F84

;*****************************************************************************
;Tabela de definições de RAM e constantes

W      equ    0              ;
saida  equ    7              ;saída de onda quadrada em RB7(led 7)
tempo  equ    0CH            ;
X      equ    0DH            ;
Y      equ    0EH            ;
W2     que    0FH            ;espelho do W original
;*****************************************************************************
;memória do programa

       org    0              ;define  o inicio do trecho a seguir em 000

       goto   inicio         ;

;      como o próprio programa é uma interrupção, detenha-se na compreensão da mesma.
;*****************************************************************************
;rotina de interrupção

       org    4              ;
       movwf  W2             ;salvo o W atual pois usarei o mesmo
       comf   X              ;inverte X
       movf   X,W            ;W=X

altera:
       movwf  PORTB          ;PORTB recebe os bits complementados
       movlw  165            ;W = valor a carregar no TMR0
       movwf  TMR0           ;TIMER 0 conta do valor ajustado até o overflow
                             ;como quero um overflow de 100 us, deveria ajustar inicio
                             ;em 256-100=156, mas devido ao atraso de 2 ciclos ao
                             ;recarregar  o  TMR0,  o  atraso  do  atendimento  da
                             interrupção e o tempo das instruções, usamos o valor 165.
       bcf    INTCON,TOIF    ;reabilito interrupção TIMER 0 antes de voltar
       movf   W2,W           ;recupero o valor de W
       retfie                ;retorna da interrupção

;*****************************************************************************
inicio:
       movlw  B'00000000'    ;W = ajuste para os bits do intcon
       movwf  INTCON         ;INTCON = W

       bsf    STATUS,RP0     ;seleciona banco 1 para OPTION e TRIS

       movlw  B'11011111'    ;W = ajuste para os bits do OPTION
       movwf  OPTION_REG     ;OPTION = W.Definiu-se o nome OPTION_REG
                             ;para evitar conflito com a istrução OPTION

       movlw  B'11111111'    ;W=ajuste para os bits do TRISA
       movwf  TRISA          ;TRISA = W; bits= 1 entrada;bits=0 saida

       movwf  B'00000000'    ;W = ajuste para os bits do TRISB
       movwf  TRISB          ;TRISB = W; bits= 1 entrada;bits=0 saída

       bcf    STATUS,RP0     ;retorna ao banco 0 (padrao do Reset)
;*****************************************************************************
principal:
       movlw 55H             ;W=01010101
       movwf  X              ;
       movwf  PORTB          ;PORTB inicializa com 01010101
       movlw  165            ;primeira carga do registro TMR0
       movwf  TMR0           ;
       bcf    INTCON,TOIF    ;retiro eventual pedido pendente
       bsf    INTCON,TOIE    ;habilita a interrupção do TMR0
;*****************************************************************************
;loop infinito, nada faz até que a próxima interrupção seja atendida

loop:
       goto   loop           ;fica aqui indefinidamente

       END                   ;fim do programa
```

19.9-Exemplo 7-Funcionamento do WATCH DOG

```
;===============================================================================
;       Programa exemplo 7
;       Objetivo:             estudar o watch dog
;      Funcionamento          se ficar mais de 2 segundos sem pressionar S0 o chip será
;                             resetado
;===============================================================================
        list p=16F84          ;para qual processador o codigo sera gerado
        radix  dec            ;padrão DECIMAL para valores sem identificação
        include<P16F84.INC>   ;anexa arquivo com definições do 16F84

;===============================================================================
;Tabela de definições de RAM e constantes

W       equ     0             ;
Led7    equ     7             ;
S0      equ     0             ;
tempo   equ     0CH           ;
X       equ     0DH           ;
Y       equ     0EH           ;
;===============================================================================
;memória do programa

        org     0             ;define  o inicio do trecho a seguir em 000

inicio:
        movlw   B'00000000'   ;W = ajuste para os bits do intcon
        movwf   INTCON        ;INTCON = W

        bsf     STATUS,RP0    ;seleciona banco 1 para OPTION e TRIS

        movlw   B'11011111'   ;W = ajuste para os bits do OPTION
        movwf   OPTION_REG    ;OPTION = W.Definiu-se o nome OPTION_REG
                              ;para evitar conflito com a istrução OPTION

        movlw   B'11111111'   ;W=ajuste para os bits do TRISA
        movwf   TRISA         ;TRISA = W; bits= 1 entrada;bits=0 saida

        movwf   B'00000000'   ;W = ajuste para os bits do TRISB
        movwf   TRISB         ;TRISB = W; bits= 1 entrada;bits=0 saída

        bcf     STATUS,RP0    ;retorna ao banco 0 (padrao do Reset)
;===============================================================================
principal:
        clrw                  ;
        movwf   PORTB         ;
        movlw   10            ;
        movwf   Y             ;

seg1:
        call    ms100         ;
        decfsz  Y             ;
        goto    seg1          ;
        movlw   255           ;
        movwf   PORTB         ;
repete:
        bcf     PORTB,Led7    ;
        call    ms100         ;
        bsf     PORTB.Led7    ;
        call    ms100         ;

        btfss   PORTA,S0      ;
        clrwdt                ;
        goto    repete        ;
;===============================================================================
;rotina de tempo de 100 ms

ms100:                        ;apenas troquei 250 por 100
        movlw   100           ;
        movwf   tempo         ;

ms100a:
        movlw   248           ;
        movwf   X             ;

ms100b:
        nop                   ;
        decfsz  X             ;

        goto    ms100b        ;
        decfsz  tempo         ;

        goto    ms100a        ;
                              ;
        return                ;
        END                   ;      fim do programa
```

20-A placa de gravação

A Microchip é uma empresa que apostou em uma linha diversificada de chips microcontroladores, de 8 a 40 pinos, de 6 a 33 entradas e saídas digitais e analógicas. As capacidades de memória ROM vão de 512 bytes a 8 Kbytes e RAM de 56 a 376 bytes, além da memória Flash. Este capítulo pretende abordar um programador universal de PICS de 8, 18, 28 e 40 pinos, de baixo custo, ligado ao PC pela porta paralela. O software de controle é padrão Windows. O protocolo de gravação é o da Microchip- **ICSP (In Circuit Serial Programing)** - aberto , e foi extraído das notas de aplicação DS30189 e AN589. Na internet existem dezenas de circuitos compatíveis com este, além de softwares de programação. Não grava os chips 16C5X e 17C4X. de protocolo diferente. Estes artigos técnicos podem ser baixados no site do fabricante: **www.microchip.com**

20.1-Modo programação

Para colocar o PIC em modo programação , é necessário manter em baixo nível os pinos RB6 e RB7(clock e dados) enquanto se produz um flanco ascendente de baixo a Vpp (0V a 13,4V- tensão mínima para gravação) do pino MCLR (S1). Uma vez neste estado, pode-se acessar a memória de programa, RB6 é utilizado como entrada de clock (sincronismo, relógio), RB7 é utilizado para entrada de bits de comando e para entrada e saída de bits de dados durante a operação série.

20.2-Arquivo Hexadecimal

O arquivo com extensão ".hex" contém o programa compilado pelo software MPASM do fabricante Microchip (ou qualquer outro software compilador). O arquivo é lido pelo programador e transferido ao PIC. O formato do arquivo é Intel INHX8M, padrão mais utilizado. Uma outra alternativa interessante é empregar compiladores na linguagem C, gerando com o compilador C um arquivo ".hex" a ser gravado no PIC. No curso não será abordado este tipo de compilação.

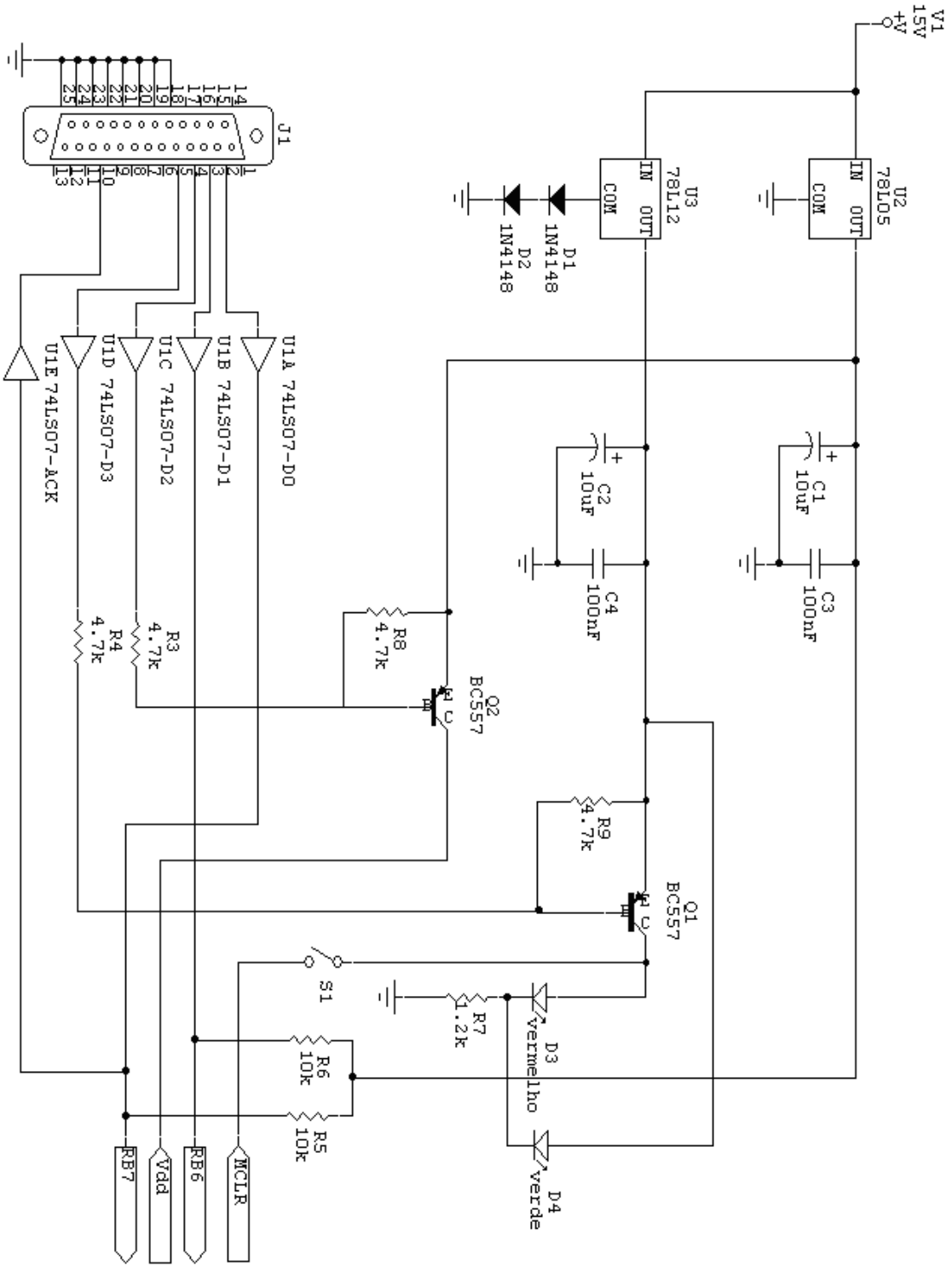
20.3- Hardware do gravador universal de PIC´s pela porta paralela

O hardware é composto por três módulos: fonte, lógica e soquetes.

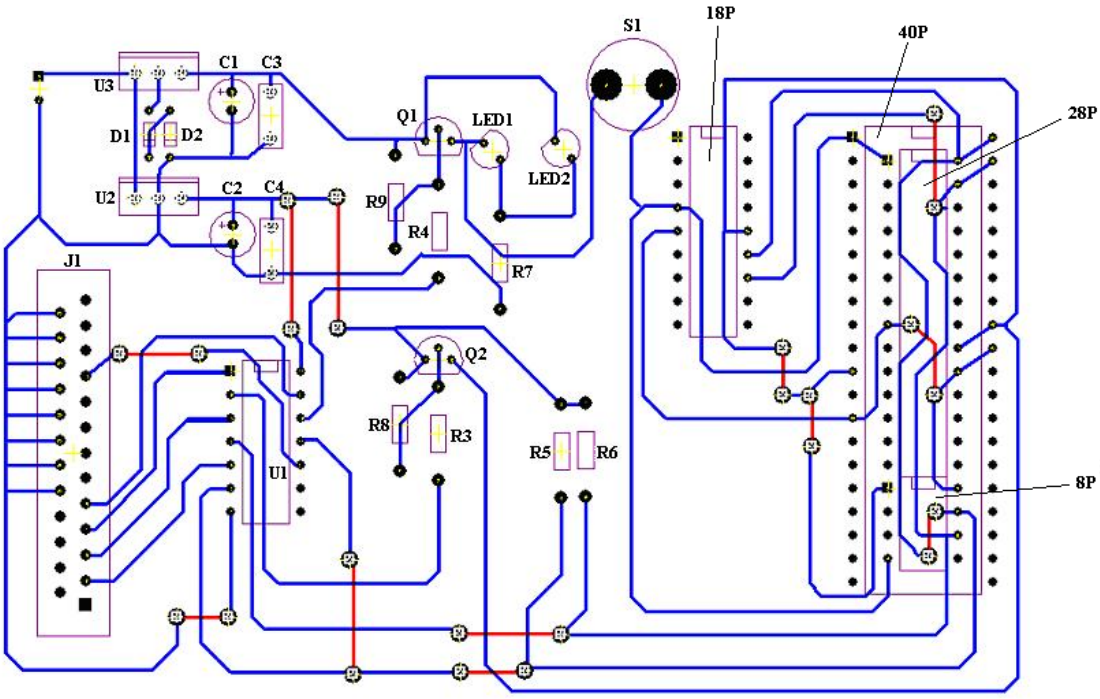
Fonte: A alimentação do PIC é padrão de 5 volts. Para a gravação dos chips o fabricante define a tensão superior a 13,4 volts para Vpp (tensão de programação). A tensão de entrada não regulada é de no mínimo 15 volts. Os diodos em série aumentam a tensão de saída.

Lógica e Sinalização: através de portas lógicas buffers TTL, os sinais que saem da porta paralela do PC são isolados e adequados para controle dos pinos data, clock e Vpp. A sinalização é feita pelos leds que indicam os estados do circuito.

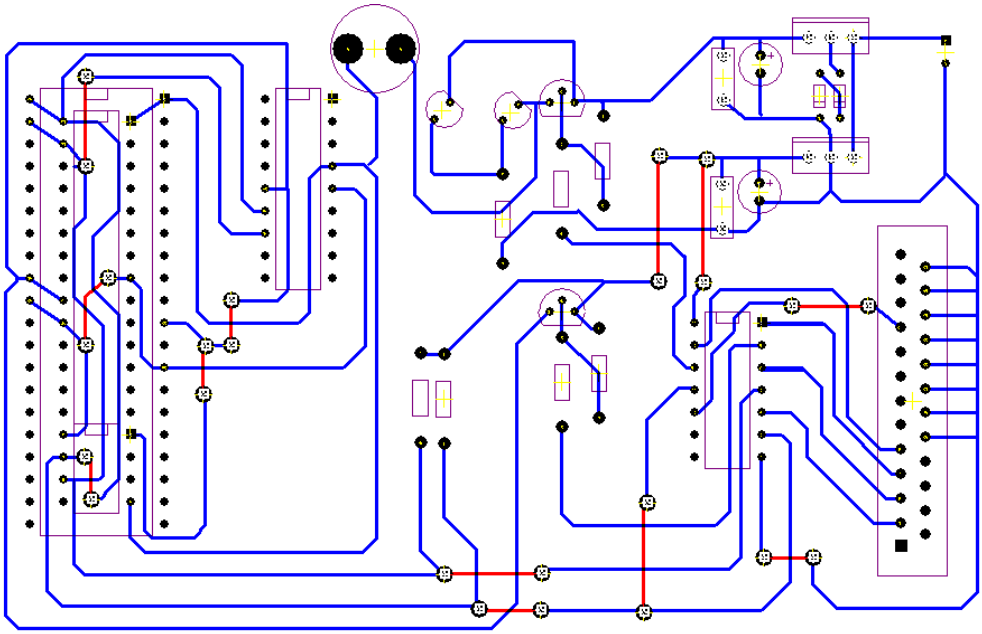
Soquetes: Como o gravador é universal e permite leitura e gravação de dezenas de PIC´s de 8, 18, 28 e 40 pinos, o circuito possui quatro soquetes na placa.


V1
15V
+V
U2
78L05
IN OUT
COM
U3
78L12
IN OUT
COM
D1
1N4148
D2
1N4148
J1
C1
10uF
C2
10uF
C3
100nF
C4
100nF
U1A 74LS07-D0
U1B 74LS07-D1
U1C 74LS07-D2
U1D 74LS07-D3
U1E 74LS07-ACK
R8
4.7k
R3
4.7k
R4
4.7k
Q2
BC557
R9
4.7k
Q1
BC557
S1
R7
1.2k
D3
vermelho
R6
10k
R5
10k
D4
verde
RB7
Vdd
RB6
MCLR

LADO COMPONENTES



LADO COBREADO



20.4- Uso da placa de gravação com os exemplos da apostila

Todos os exemplos fornecidos, bem como exemplos que serão dados para o 16C711 e o 12C508, funcionam na placa de gravação.

Basta gravar os chips e coloca-los no circuito teste, testando as funções programadas.

21-O PIC 16C711 (com conversores A/D)

21.1-Introdução

O PIC 16C711 é um microcontrolador que pode operar de DC até 20 MHz (ciclo de instrução de 200 ns) e assim como o PIC 16F84 funciona com um mínimo de componentes externos.

Diferente do PIC 16F84 que possui EEPROM de dados, o PIC 16C711 tem como periférico especial um conversor A/D de 8 bits com 4 canais multiplexados.

IMPORTANTE:

O PIC 17C711 NÃO POSSUI MEMÓRIA DE PROGRAMA EEPROM, SEDO DISPONÍVEL PARA DESENVOLVIMENTO UMA VERSÃO COM JANELA, APAGÁVEL COM LUZ ULTRAVIOLETA. A MICROCHIP NÃO RECOMENDA A PROTEÇÃO DE CÓDIGO DE DISPOSITIVOS COM JANELA.

Suas principais características são:

- 1K (1024) palavras de 14 bits.
- 68 bytes de RAM para uso geral
- Stack com 8 níveis
- Apenas 35 instruções
- 16 registros específicos em RAM para controle do chip.
- Timer 0 de 8 bits com opção de prescaler de 8 bits
- 13 pinos que podem ser configurados individualmente como entrada ou saída
- alta capacidade de corrente nos pinos (podem acender um led diretamente)
- capacidade de gerenciar interrupções externas(até 5 entradas), do timer 0 e do A/D
- Watch dog para recuperação e reset em caso de travas o software
- Memória de programas protegida contra cópias.
- Modo SLEEP para economia de energia
- Várias opções de osciladores
- Brown-out reset

Neste capítulo vamos estudar apenas as diferenças entre o 16F84 e o 16C711, sendo que os demais periféricos funcionam de modo semelhante.

21.2-Pinagem e características elétricas básicas

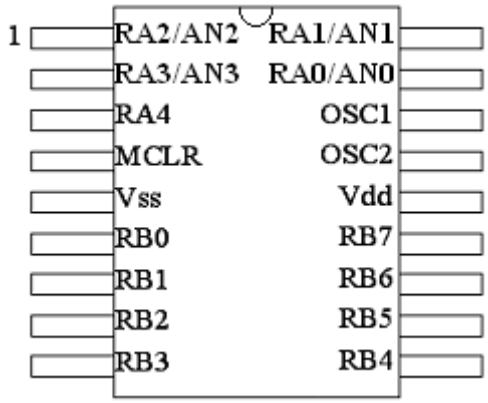


Figura 21.1-Pinos do PIC 16C711 no encapsulamento DIP

Descrição dos pinos agrupados por blocos

Pino 14	Vdd	Tensão de alimentação
Pino 5	Vss	Referência de Terra
Pino 17	RA0	Porta A, bit 0. Entrada ou saída digital Canal analógico 0
Pino 18	RA1	Porta A, bit 1. Entrada ou saída digital Canal analógico 1
Pino 1	RA2	Porta A, bit 2. Entrada ou saída digital Canal analógico 2
Pino 2	RA3	Porta A, bit 3. Entrada ou saída digital Canal analógico 3
Pino 3	RA4/TOCKI	Porta A, bit 4. Entrada ou saída digital,entrada TIMER 0
Pino 4	MCLR	Entrada de reset em nível 0
Pino 16	OSC1/CLKIN	Cristal de clock externo
Pino 15	OSC2/CLKOUT	Cristal ou saída Fosc/4 em modo RC
Pino 6	RB0/INT	Porta B, bit 0. Entrada ou saída digital, ou interrupção externa
Pino 7	RB1	Porta B, bit 1. Entrada ou saída digital
Pino 8	RB2	Porta B, bit 2. Entrada ou saída digital
Pino 9	RB3	Porta B, bit 3. Entrada ou saída digital
Pino 10	RB4	Porta B, bit 4. Entrada ou saída digital, interrupção nas mudanças de estados
Pino 11	RB5	Porta B, bit 5. Entrada ou saída digital, interrupção nas mudanças de estados
Pino 12	RB6	Porta B, bit 6. Entrada ou saída digital, interrupção nas mudanças de estados
Pino 13	RB7	Porta B, bit 7. Entrada ou saída digital, interrupção nas mudanças de estados

Faixa de tensão de alimentação: 2,0V a 6,0V (típica 5,0V)
Consumo de corrente:1)<2 mA a 5V, 4MHz
2)15 uA a 3V, 32KHz
3)1 uA em standby

21.3-Principais diferenças nos registros

Endereço	No PIC 16F84	No PIC16C711
08H	EEDATA	ADCON0
09H	EEADR	ADRES
88H	EECON1	ADCON1
89H	EECON2	ADRES
87H	Não usado	PCON

21.4-Registro INTCON no PIC 16C711

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
GIE	**ADIE**	**TOIE**	**INTE**	**RBIE**	**TOIF**	**INTF**	**RBIF**

Endereço: 0BH, 8BH valor de reset:0000000X

Bit 7-RW	**GIE**	Bit global interrupt enable (habilitação global) 1=habilita as interrupções que estejam individualmente selecionadas 0=desabilita todas as interrupções
Bit 6-	**ADIE**	Interrupção do conversor A/D 1=habilita interrupção do A/D 0=desabilita interrupção do A/D
Bit 5-RW	**TOIE**	Interrupção gerada pelo overflow no TMR0 1=habilitada 0=desabilitada
Bit 4-R	**INTE**	Interrupção externa RB0/INT 1=habilitada 0=desabilitada
Bit 3-R	**RBIE**	Interrupção por mudanças na PORTB 1=habilitada 0=desabilitada
Bit 2-RW	**TOIF**	Sinaliza interrupção pelo overflow do TMR0(*) 1=ocorreu um overflow no TMR0 0=ainda não ocorreu overflow no TMR0
Bit 1-RW	**INTF**	Sinaliza interrupção externa no pino RB0/INT (*)
Bit 0-RW	**RBIF**	Sinaliza interrupção de mudanças na PORTB (*)

(*) devem ser zerados pelo software.

21.5-Registro PCON no PIC 16C711

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
-	-	-	-	-	-	**POR**	**BOR**

Endereço: 0BH, 8BH valor de reset:0000000X

Bit 7 a 2		Não implementados. Lidos como 0
Bit 1-RW	**POR**	Bit de status do Power on reset 1=não ocorreu power on reset 0=ocorreu power on reset (deve ser setado pelo software logo após o power on reset)
Bit 0-RW	**BOR**	Bit de status do Brown out 1=não ocorreu Brown out reset 0=ocorreu Brown out reset (deve ser setado pelo software logo após o Brown out reset)

21.6-Registros de controle do A/D

Os registros de controle do A/D são o ADCON1 e ADCON2. No registro ADRES teremos o resultado da última conversão realizada.

21.6.1-ADCON0

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
ADCS1	**ADCS0**	-	**CHS1**	**CHS0**	**GO/DONE**	**ADIF**	**ADON**

Endereço: 08H valor de reset:00000000

Bit 7 a 2		Não implementados. Lidos como 0
Bit 7-RW Bit 6-RW	**ADCS1** **ADCS0**	Seleção de clock para o A/D 00=Fosc/2 01=Fosc/8 10=Fosc/32 11=Frc (o clock deriva de um oscilador RC)
Bit 5	Não implementado: sempre lido como 0	
Bit 4-RW Bit 3-RW	**CHS1** **CHS0**	Seleciona em qual canal será efetuada a amostragem e conversão 00=canal 0(RA0/AN0) 01=canal 1(RA1AN/1) 10=canal 2(RA2/AN2) 11=canal 3(RA3/AN3)
Bit 2-RW	**GO/DONE**	Inicio e situação da conversão (somente ADON=1) 1=conversão sendo realizada(setar para iniciar a conversão) 0=conversão finalizada ou ainda não está em progresso(o hardware zera este bit ao fim da conversão)
Bit 1-RW	**ADIF**	Flag (bit) de requisição de interrupção do A/D 1=conversão completa.(deve ser zerado pelo software) 0=conversão não completa
Bit 0-RW	**ADON**	Bit para ligar e desligar o módulo A/D 1=módulo A/D operando 0=módulo A/D desligado. Não consome corrente

21.6.2-ADCON1

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
-	-	-	-	-	-	**PCFG1**	**PCFG0**

Endereço: 88H valor de reset:00000000

<table>
<tr><td colspan="2">Bit 7 a 2</td><td>Não implementados. Lidos como 0</td></tr>
<tr><td>Bit 1-RW
Bit 0-RW</td><td>PCFG1
PCFG0</td><td>Configuração das portas do A/D
<table>
<tr><th>PCFG1</th><th>PCFG0</th><th>RA1 e RA1</th><th>RA2</th><th>RA3</th><th>Vref</th></tr>
<tr><td>0</td><td>0</td><td>analógica</td><td>analógica</td><td>analógica</td><td>Vdd</td></tr>
<tr><td>0</td><td>1</td><td>analógica</td><td>analógica</td><td>Vref</td><td>RA3</td></tr>
<tr><td>1</td><td>0</td><td>analógica</td><td>I/O digital</td><td>I/O digital</td><td>Vdd</td></tr>
<tr><td>1</td><td>1</td><td>I/O digital</td><td>I/O digital</td><td>I/O digital</td><td>Vdd</td></tr>
</table></td></tr>
</table>

21.6.3-ADRES

Este registro de 8 bits, presente nos bancos 0 e 1 contém ovalor digital do sinal cuja entrada analógica foi habilitada e convertida.

21.7-Utilizando o conversor A/D

Quando uma conversão for finalizadas pelo módulo A/D (o mesmo opera durante o SLEEP) o resultado é armazenado no registro ADRES, o bit GO/DONE\ é zerado e o bit ADIF é setado, sinalizando a interrupção de conversão (somente atendida se ADIE=1).

Os bits do PORTA configurados como entrada analógica devem ter seus controles em TRISA setados.

Para efetuar uma conversão devemos seguis alguns passos:
-configurar ADCON1, selecionando o modo dos pinos do PORTA
-Selecionar qual o canal a converter em ADCON0 (CHS1 e CHS0)
-Selecionar a freqüência de conversão em ADCON0 (ADCS1 E ADCS0)
-Ligar o módulo A/D em ADCON0(ADON=1)
-Ajustar a interrupção(se for utilizada) com ADIF=0, ADIE=1 e GIE=1
-Esperar o tempo de aquisição por parte do A/D (ver item 21.7.1)
-Iniciar a conversão(GO/DONE\=1)
-Esperar o fim da conversão (pela interrupção ou por varredura até que GO/DONE\= 0 ou ADIF=1.
-Ler o resultado da conversão em ADRES
-Esperar para efetuar nova conversão(ver item 2.7.2)

21.7.1-Tempo de aquisição para o A/D

Após selecionarmos o canal desejado no módulo A/D devemos esperar um tempo para que a amostragem seja realizada e o capacitor interno carregado.

Não entraremos em detalhes. Basta que o usuário deixe um tempo de aproximadamente 15 us que deverá ser suficiente para a maioria das aplicações.

21.7.2-Tempo de espera para novas conversões

Rodando a 4 MHz (caso dos exemplos) devemos esperar um tempo antes de iniciar nova conversão(6 us no pior dos casos para 20MHz), para qualquer seleção do clock do A/D.

21.8-Brown-out reset

O Brown-out reset é um reset que ocorre sempre que a alimentação cai a um valor mínimo, mas não chega a zero, retornando ao valor normal.

Nestas condições o reset pelo pino MCLR\ não ocorre, mas a CPU pode ter se perdido ou o estado de algum pino pode ter sido alterado erroneamente.

O 16C711 possui internamente um circuito detector de Brown reset que além de resetar a CPU sinaliza esta condição pelo bit BOR\ (item 21.5).

21.8.1-Reconhecimento dos tipos do reset

POR\	BOR\	TO\	PD\	Tipo de reset
0	X	1	1	Power down reset
1	0	X	X	Brown-out reset
1	1	0	1	Reset pelo watch dog
1	1	0	0	Sai do SLEEP pelo watch dog
1	1	U	U	Reset pelo MCLR\ em operação normal
1	1	1	0	Sai do SLEEP pelo MCLR\ ou por interrupção

21.8.2-Tensão de Brown-out reset

A tensão do pino Vdd que gera o brown-out reset é de 4,0V com tolerância de mais ou menos 0,3V.

21.8.3-Palavra de configuração do 16C711.

Assim como o 16F84 (cap 9) o 16C711 também possui uma palavra de configuração na memória de programa para ajuste do funcionamento do hardware.

Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
CP	CP	CP	CP	CP	CP	CP	BODEN\	CP	CP	PWRTE	WDTE	FOSC1	FOSC0

Endereço valor do reset: uuuuuuuuuuuuuu

Bit 13 a 7 5 e 4	CP	Bit de proteção do código 1= código desprotegido 0= código protegido
Bit 6	BODEN	Habilitação do Brown-out reset 1=Brown-out reset habilitado (garantir que PWRTE\=0) 0=Brown-out reset desabilitado
Bit 3	PWRTE	Habilitação do power-up timer 1= power-up timer desabilitado 0= power-up timer habilitado
Bit 2	WDTE	Bit de habilitação do WATCH DOG 1= WDT habilitado 0= WDT desabilitado
Bit 1	FOSC1	Seleciona tipo de oscilador 11= modo RC externo 10= modo cristal ou ressonador de alta velocidade HS 01= cristal ressonador de baixa velocidade XT 00= cristal de baixa potência LP
Bit 0	FOSC0	

21.8.4-Considerações sobre os bits de status de reset

Conforme as necessidades do sistema, o usuário deve proceder a uma verificação dos bits POR\, BOR\, TO\, PD\ a cada reset para poder identificar qual o evento responsável para poder identificar qual o evento responsável pelo mesmo e agir conforme tal.

21.9-Exemplo de leitura no canal 0

Para este exemplo, considere o circuito da figura 21.1, onde um PIC 16C711 com clock de 4 MHz, 8 leds ligados ao PORTB e um potenciômetro ligado ao pino RA0/AN0.

Neste exemplo, o valor presente em RA0 será convertido para seu equivalente digital, com 8 bits de resolução, escrito no PORTB.

Temos então que 8 bits são 256 variações para a escala de 0 a 5 V, logo, cada incremento nos bits indicados pelos leds significa um incremento de aproximadamente 19.53 mV no sinal de entrada.

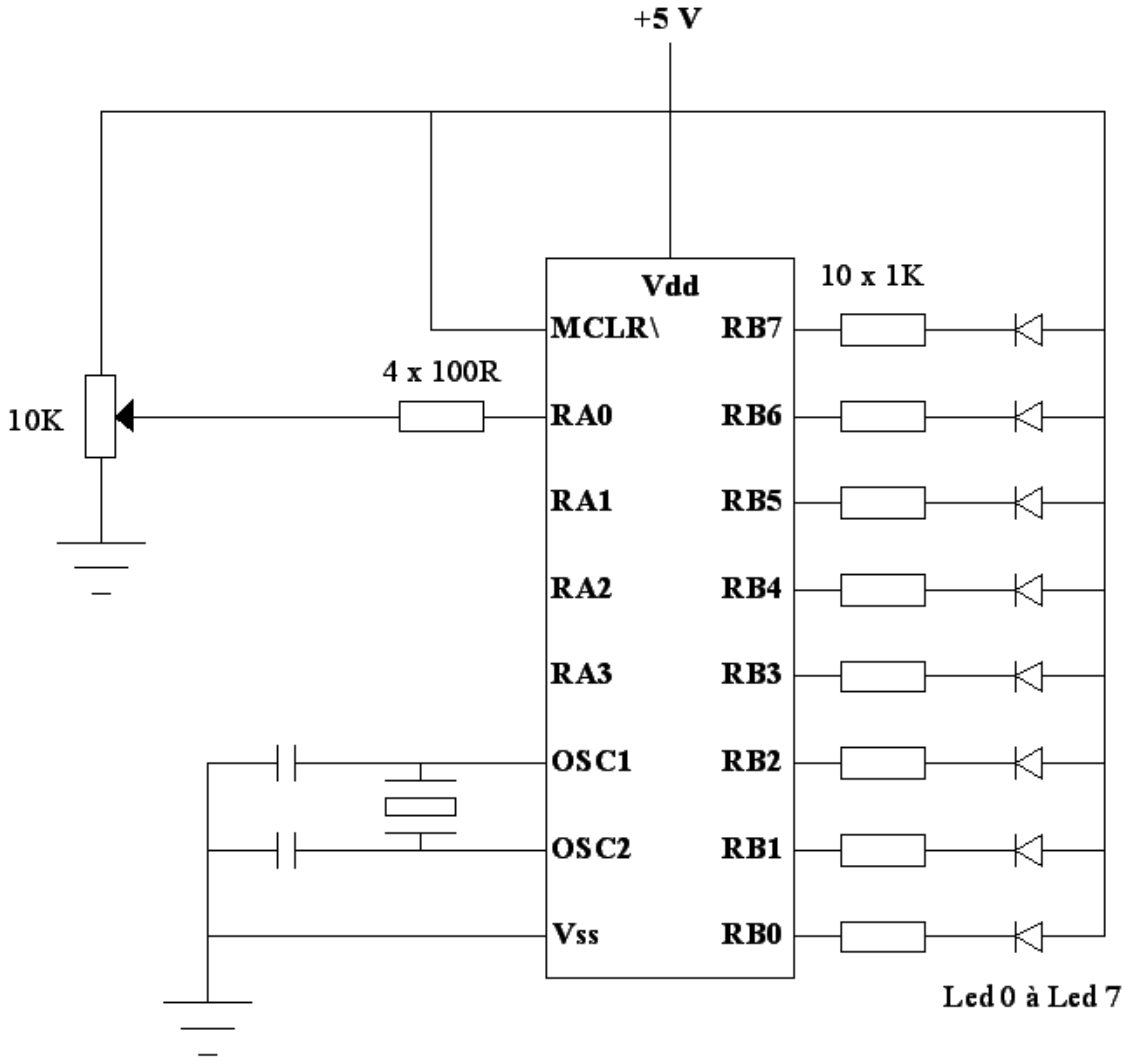


Figura 21.2-Circuito de testes para o módulo A/D do PIC 16C711

```
;===============================================================================
;       Programa exemplo 1 do capitulo 21
;
;       Objetivo:               mostrar o uso do módulo A/D
;       Funcionamento           Conforme variamos o potenciômetro temos nos leds ligados
;                               ao PORTB o valor digital convertido. Espera fim de
;                               varredura em ADIF
;===============================================================================
        list p=16C711           ;para qual processador o codigo sera gerado
        radix   dec             ;padrão DECIMAL para valores sem identificação
        include<16C711.INC>     ;anexa arquivo com definições do 16F84

;===============================================================================
;Tabela de definições de RAM e constantes

Led7    equ     7               ;led 7 está em RB7
tempo   equ     0CH             ;variável tempo na RAM 0CH
dt1     equ     0DH             ;define variavel auxiliar X na RAM 0CH
                                ;(primeira posição)
valor   equ     0EH             ;
W       equ     0               ;facilita a referencia a w quando necessário
;===============================================================================
;memória de programa

        org     0               ;define o trecho a seguir em 000
                                ;não foi reservada definição para interrupção, pois não
                                ;serão utilizadas
;===============================================================================
;inicio do programa logo após o reset

inicio:
        movlw   B'00000000'     ;W = ajuste para os bits do intcon
        movwf   INTCON          ;INTCON = W = 0

        clrf    PORTA           ;
        clrf    PORTB           ;inicializa portas A e B

        bsf     STATUS,RP0      ;seleciona banco 1 para OPTION e TRIS

        movlw   B'11011111'     ;W = ajuste para os bits do OPTION
        movwf   OPTION_REG      ;OPTION = W.Definiu-se o nome OPTION_REG
                                ;para evitar conflito com a istrução OPTION

        movlw   B'11111111'     ;W=ajuste para od bits do TRISA
        movwf   TRISA           ;TRISA = W; bits= 1 entrada;bits=0 saida

        movwf   B'00000000'     ;W = ajuste para os bits do TRISB
        movwf   TRISB           ;TRISB = W; bits= 1 entrada;bits=0 saída

        movlw   B'00000000'     ;W = ajuste para os bits do OPTION
        movwf   OPTION_REG      ;OPTION = W.Definiu-se o nome OPTION_REG
                                ;para evitar conflito com a istrução OPTION

        movlw   B'00000010'     ;W=ajuste para os bits do ADCON1
        movwf   ADCON1          ;ADCON1 = W; bits= 1 entrada analógica;bits=0 entrada
                                ;digital

        bcf     STATUS,RP0      ;retorna ao banco 0 (padrao do Reset)

        movlw   B'10000001'     ;W=ajuste para os bits do ADCON0
        movwf   ADCON0          ;clock do A/D =Fosc/8, canal 0, módulo ligado
;===============================================================================
principal:

u10:                            ;perde 10 us
        movlw   3               ;1 ciclo                        (1)
        movwf   dt1             ;1 ciclo                        (2)

esp:
        decfsz  dt1             ;1 ciclo se >0 senão 2 ciclos   (3)
        goto    esp             ;2 ciclos se >0 senão 0         (4)

;===============================================================================
;explicando o loop
;
;perde 1 us em (1)
;perde 1 us em (2)
;perde 1 us em (3)    porque não resultou >0 (3-1=2)
;perde 2 us em (4)
;perde 1 us em (3)
;perde 2 us em (4)    porque não resultou >0 (2-1=1)
;perde 2 us em (3)    porquw resultou em 0 (1-1=0), o goto é ignorado
;===============================================================================

        bsf     ADCON0,GO       ;inicia a conversão
espera:
        btfss   ADCON0,ADIF     ;se ADIF=1 pula próxima isntrução
```

```
        goto    espera          ;ADIF=0 espera ir a 1
                                ;terminou a conversão
        bcf     ADCON0,ADIF     ;reseta flag de interrupção
        movf    ADRES,W         ;W=ADRES
        movwf   valor           ;valor=ADRES
        comf    valor,W         ;w=valor complementado(invertido) pois os leds acendem em
                                ;nível 0
        movwf   PORTB           ;escreve no PORTB
        call    ms250           ;espera 250 ms antes de repetir
                                ;evita que os leds fiquem instáveis devido a ruído
                                ;e imprecisão no potenciômetro
        goto    principal       ;repete todo o ciclo de leitura
;================================================================================
;rotina de tempo de 250 ms

ms250:                          ;o call para a rotina leva 2 us
        movlw   250             ;+1
        movwf   tempo           ;+1     total1= 4 us

ms250a:
        movlw   248             ;+1
        movwf   dt1             ;+1     total2=2 us

ms250b:
        nop                     ;1
        decfsz  dt1             ;1      248 x4 us + (1us nop + 2 us quando decfsz da 0

        goto    ms250b          ;2      da um tempo total3 = 995 us
        decfsz  tempo           ;1      1 us pq tempo > 0+2 us goto da um total4=3 us

        goto    ms250a          ;2      250 x(total2+total3+total4)+2 us
                                ;       quando tempo=0=total5=250.002 us
        return                  ;       delay=total 1+total5+2 us do return = 250008 us

        END                     ;       fim do programa
```

22-O revolucionário PIC 12C508, com apenas 8 pinos

22.1-Introdução

O PIC 12C508 é um microcontrolador baseado na família 16C5X, possuindo limitações quanto a seus periféricos, mas em compensação usando apenas 8 pinos.

As versões mais usadas no desenvolvimento são EPROM e OTP(grava apenas uma vez).

Suas principais características são:

-Funciona de DC até 4 MHz(tempo de ciclo de 1 us)
-33 instruções
-512 palavras de programa (1K no 12C509)
-25 bytes de RAM (41 no 12C509)
-Opção para oscilador interno fixo em 4 MHz
-Reset interno

Neste capítulo vamos estudar apenas as diferenças entre o PIC 16F84 e o PIC 12C508, sendo que os demais periféricos(Watch Dog ,timer ,....) que existem neste funcionam de modo semelhante.

IMPORTANTE:

O 12C508 não possui memória de programa EEPROM, tendo disponível para desenvolvimento uma versão EPROM com janela, apagável por luz ultravioleta.

A Microchip não recomenda a proteção de código de dispositivos com janela.

22.2-Pinagem e características elétricas básicas

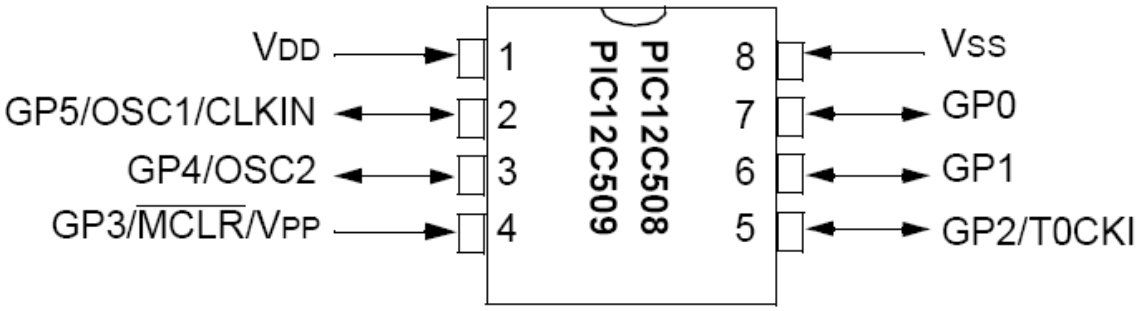


Figura 22.1-Pino do PIC 12C508 no encapsulamento DIP

Faixa de tensão de alimentação: 2,5V a 5,5V (típica 5,0V)
Consumo de corrente:1)<2 mA a 5V, 4MHz
2)15 uA a 3V, 32KHz
3)1 uA em standby

Descrição dos pinos agrupados por blocos

Pino 1	Vdd	Tensão de alimentação
Pino 8	Vss	Referência de Terra
Pino 7	GPIO0	Porta A, bit 0. Entrada ou saída digital
Pino 6	GPIO 1	Porta A, bit 1. Entrada ou saída digital
Pino 5	GPIO2/TOCKI	Porta A, bit 2. Entrada ou saída digital entrada TIMER 0
Pino 4	GPIO3/MCLR\	Porta A, bit 3. Entrada digital ou MCLR\
Pino 3	GPIO4/OSC2	Porta A, bit 4. Entrada ou saída digital. Entrada de clock
Pino 2	GPIO5/OSC1/CLKIN	Porta A, bit 5. Entrada ou saída digital. Entrada de clock

22.3-Diferenças principais a nível de software e hardware

O PIC 12C508 tem memória de programa de apenas 12 bits de comprimento, seu set de instruções é ligeiramente diferente, seu stack tem apenas 2 niveis.

Devemos ainda considerar que o PIC 12C508 possui como periféricos apenas 6 bits de I/O e o timer 0, sendo os demais (watch dog. POR,....)são inerentes a todos os PIC´s.

22.4-Porta I/O np PIC 12C508

No PIC 12C508 existe apenas uma porta de I/O, chamada de GPIO, mapeada no mesmo endereço do PORTB do PIC 16F84, embora com apenas 6 bits mapeados(0 à 5).

Nos membros da família 15C5X os registros TRIS e OPTION não são endereçáveis como na família 16CXXX, sendo necessário acessa-los pelas instruções equivalentes (ver set de instruções do 16F84), embora tenham funcionamento equivalente.

Como o estudo da família d16C5X não é o objetivo principal deste curso, vamos apresentar apenas um circuito mínimo de testes e um programa bem simples.

Neste circuito utilizamos clock de 4 MHz, reset externo e temos então 5 pinos disponíveis.

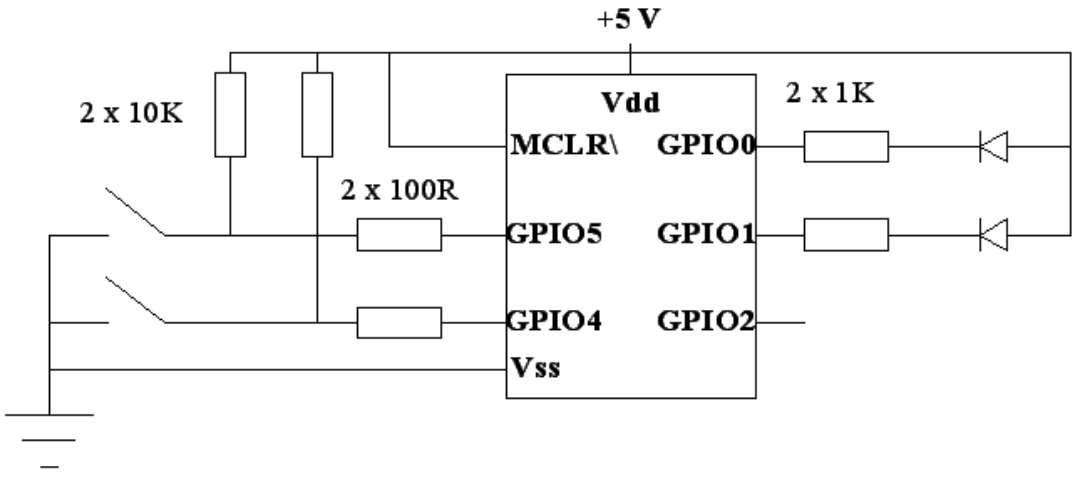


Figura 22.2-Circuito de exemplo com PIC 12C508

22.5-Exemplo com PIC 12C508. Acender leds conforme a situação das chaves.

Neste exemplo temos 2 chaves ligadas em GPIO5 (S2) e GPIO4 (S1) e dois leds ligados a GPIO0 (led6) e GPIO1 (led7).

A nomenclatura S2, S3, led6 e led7 é a mesma utilizada no exemplo do 16F84.

```
;*****************************************************************************
;       Programa exemplo do 12C508
;
;       Objetivo:       Ler teclas e acionar leds
;       Funcionamento   se S2 e S3 soltas o led 6 pisca 1 vez e espera 1s
;                       se S2 pressionada e S3 solta, led6 pisca 2 vezes e espera 1s
;                       se S2 solta e S3 pressionada; led7   pisca 2 vezes e espera 1s
;                       se S2 e S3 presionadas, led6 e led7 piscam 2 vezes e esperam 1s
;*****************************************************************************
        list p=12c508           ;para qual processador o codigo sera gerado
        radix   dec             ;padrão DECIMAL para valores sem identificação
        include<P12C508.INC>    ;anexa arquivo com definições do 12C508
;*****************************************************************************
;Tabela de definições de RAM e constantes

Led6    equ     0               ;led 6  está em GPIO0
Led7    equ     1               ;led 7 está em GPIO1
S2      equ     5               ;S2 está em GPIO5
S3      que     4               ;S3 está em GPIO4
tempo   equ     0CH             ;variável tempo na RAM 0CH
X       equ     0DH             ;define variavel auxiliar X na RAM 0CH
                                ;(primeira posição)
Y       equ     0EH             ;
W       equ     0               ;facilita a referencia a w quando necessário
;*****************************************************************************
;memória de programa

        org     0               ;define o trecho a seguir em 000

        movlw   B'00000000'     ;W = ajuste para os bits do intcon
        OPTION                  ;na família 12C5xxx e 16C5x os registros OPTION e TRIS não
                                ;são endereçáveis(não estão na RAM)
                                ;ver isntruções equivalentes (em cinza no cap 18)

        movlw   B'11111100'     ;W=ajuste para od bits do GPIO
        movwf   GPIO            ; GPIO = W; bits= 1 entrada;bits=0 saida

;*****************************************************************************
principal:
        btfss   GPIO,S2         ;se S2=off pula
        goto    S2on            ;S2 pressionado
        btfss   GPIO,S3         ;se S3=off pula
        goto    S3on            ;S3 pressionado
Pisca1:
        bcf     GPIO,Led7       ;
        bcf     GPIO,Led6       ;
        call    ms250           ;
        bsf     GPIO,Led7       ;
        bsf     GPIO,Led6       ;
        call    ms250           ;
        call    ms250           ;
        call    ms250           ;
        call    ms250           ;
        goto    principal
S2on:
        btfss   GPIO,S3         ;
        goto    S2S3on          ;
S0S2:
        bcf     GPIO,Led7       ;
        call    ms250           ;
        bsf     GPIO,Led7       ;
        call    ms250           ;
        bcf     GPIO,Led7       ;
        call    ms250           ;
        bsf     GPIO,Led7       ;
        call    ms250           ;
        call    ms250           ;
        call    ms250           ;
        call    ms250           ;
        goto    principal
```

```
;================================================================================================
S3on:
       btfss   GPIO,S2
       goto    S2S3on
;================================================================================================
S0S3:
       bcf     GPIO,Led6       ;
       call    ms250           ;
       bsf     GPIO,Led6       ;
       call    ms250           ;
       bcf     GPIO,Led6       ;
       call    ms250           ;
       bsf     GPIO,Led6       ;
       call    ms250           ;
       call    ms250           ;
       call    ms250           ;
       call    ms250           ;
       goto    principal
;================================================================================================
S2S3on:
       bcf     GPIO,Led6       ;
       bcf     GPIO,Led7       ;
       call    ms250           ;
       bsf     GPIO,Led6       ;
       bsf     GPIO,Led7       ;
       call    ms250           ;
       bcf     GPIO,Led6       ;
       bcf     GPIO,Led7       ;
       call    ms250           ;
       bsf     GPIO,Led6       ;
       bsf     GPIO,Led7       ;
       call    ms250           ;
       call    ms250           ;
       call    ms250           ;
       call    ms250           ;
       goto    principal

;================================================================================================
;rotina de tempo de 250 ms

ms250:                         ;o call para a rotina leva 2 us
       movlw   250             ;+1
       movwf   tempo           ;+1     total1= 4 us

ms250a:
       movlw   248             ;+1
       movwf   X               ;+1     total2=2 us

ms250b:
       nop                     ;1
       decfsz  X               ;1      248 x4us +(1us nop+2us quando decfsz da 0

       goto    ms250b          ;2      da um tempo total3 = 995 us
       decfsz  tempo           ;1      1us pq tempo>0+2us goto da um total4=3 us

       goto    ms250a          ;2      250 x(total2+total3+total4)+2 us
                               ;       quando tempo=0=total5=250.002 us
       return                  ;delay =total1+total5+2us do return = 250008 us

       END                     ;fim do programa
```

23-Visão geral das famílias PIC12CXXX, 16CXXX e 16C5X

23.1-Família 12CXXX

Item	Palavras de Programa	RAM	Clock max.(MHz)	Pinos I/O	Principais periféricos
12C508	512x12	25	4	6	1 timer, Watch Dog
12C509	1024x12	41	4	6	1 timer, Watch Dog
12C671	1024x14	128	4	6	1 timer, Watch Dog 4 A/D
12C672	2048x14	128	4	6	1 timer, Watch Dog 4 A/D

23.2-Família 16C5X

Item	Palavras de Programa	RAM	Clock max.(MHz)	Pinos I/O	Principais periféricos
16C52	384x12	25	4	12	1 timer
16C54	512x12	25	20	12	1 timer, Watch Dog
16C54A	512x12	25	20	12	1 timer, Watch Dog
16C154	512x12	25	20	12	1 timer, Watch Dog
16C55	512x12	24	20	20	1 timer, Watch Dog
16C56	1024x12	25	20	12	1 timer, Watch Dog
16C156	1024x12	25	20	12	1 timer, Watch Dog
16C57	2048x12	72	20	20	1 timer, Watch Dog
16C58A	2048x12	73	20	12	1 timer, Watch Dog
16C158	2048x12	73	20	12	1 timer, Watch Dog

23.3-Família 16CXXX

Item	Palavras de Programa	RAM	Clock max.(MHz)	Pinos I/O	Principais periféricos
1400	4096x14	192	20	20	8 A/D, 12C, 2 D/A, 2 comparadores, 2 timers, watch dog
16C554	512x14	80	20	13	1 timer, watch dog
16C556	1024x14	80	20	13	1 timer, watch dog
16C558	2048x14	128	20	13	1 timer, watch dog
16C61	1024x14	36	20	13	1 timer, watch dog
16C62	2048x14	128	20	22	12c, spi, 1pwm, 3 timers, watch dog
16C63	4096x14	192	20	22	12c, spi, usart, 2 pwm, 3 timers, watch dog Brown out
16C64	2048x14	128	20	33	12c, spi, pwm, 3 timers, watch dog
16C64A	2048x14	128	20	33	12c, spi, 1 pwm, 3 timers, watch dog Brown out detect
16C65	4096x14	192	20	33	12c, spi, usart, 2 pwm, 3 timers, watch dog
16C65A	4096x14	192	20	33	12c, spi, usart, 2 pwm, 3 timers, watch dog, Brown out
16C66	8192x14	368	20	22	12c, spi, usart, 2 pwm, 3 timers, watch dog, Brown out
16C67	8192x14	368	20	33	12c, spi, 2 pwm, 3 timers, watch dog Brown out detect
16C620	514x14	80	20	13	2 comparadores, 1 timer, watch dog Brown out detect
16C621	1024x14	80	20	13	2 comparadores, 1 timer, watch dog Brown out detect
16C622	2048x14	128	20	13	2 comparadores, 1 timer, watch dog Brown out detect
16C642	4096x14	176	20	22	2 comparadores, 1 timer, watch dog Brown out detect
16C662	4096x14	176	20	33	2 comparadores, 1 timer, watch dog Brown out detect
16C710	512x14	36	20	13	4 A/D,1 timer, watch dog, Brown out detect
16C71	1024x14	36	20	13	4 A/D, 1 timer, watch dog
16C711	1024x14	68	20	13	4 A/D, 1 timer, watch dog, Brown out detect
16C715	2048x14	128	20	13	4 A/D ,1 timer, watch dog, Brown out detect
16C72	2048x14	128	20	22	5 A/D, 12C, spi, 1 pwm, 3 timers, watch dog Brown out detect
16C73	4096x14	192	20	22	5 A/D, usart, 12c, spi, 2 pwm, 3 timers, watch dog
16C73A	4096x14	192	20	22	5 A/D, usart, 12c, spi, 2 pwm, watch dog, 3 timers, Brown out detect
16C74	4096x14	192	20	33	8 A/D, usart, 12c, spi, 2 pwm, 3 timers, watch dog
16C74A	4096x14	192	20	33	8 A/D, usart, 12c, spi, 2 pwm, 3 timers, watch dog
16C76	8192x14	368	20	22	5 A/D, usart, 12c, spi, 2 pwm, 3 timers, watch dog Brown out detect
16C77	8192x14	368	20	33	8 A/D, usart, 12c, spi, 2 pwm, 3 timers, watch dog Brown out detect
16F83	512x14 (flash)	36+ 64 eeprom	10	13	1 timer, watch dog
16C84	1024x14 (eeprom)	36+ 64 eeprom	10	13	1 timer, watch dog
16F84	1024x14 (flash)	36+ 64 eeprom	10	13	1 timer, watch dog
16C923	4096x14	176	8	52	12c, spi, 1 pwm, 3 timers watch dog
16C924	4096x14	176	8	52	5 A/D, 12c, spi, 1 pwm, 3 timers, watch dog

IMPORTANTE:

Estas tabelas foram baseadas em informações de abril de 1997.

Devido a grande velocidade no desenvolvimento de novos produtos, estas tabelas podem estar ligeiramente desatualizadas.

APÊNDICE 1- Uso do compilador MpAsmWin

Este apêndice tem apenas por objetivo auxiliar o leitor no inicio de seus trabalhos com o compilador MPASMWIN da Microchip.

Este compilador está disponível gratuitamente para download no site www.microchip.com, assim como outras ferramentas como o MPLAB, bastando copiá-los e usar.

Apenas copie-o para um novo diretório, por exemplo, c:\MPASMWIN

Neste diretório, execute o programa MPASMWIN (menu iniciar→executar→c:\MPASMWIN\MPASMWIN.exe) ou através de seu ícone em seu grupo de programas. É conveniente criar um atalho para o mesmo em sua área de trabalho, para evitar toda a seqüência de chamá-lo repetidas vezes.

Usando o MPASMWIN pela primeira vez.

Apenas execute o programa e veja a tela inicial. Nas opções disponíveis, selecione:

Radix:	default
Warning level:	default
Hex output:	INHX8M
Generated files:	error file
	List file
Macro expansion:	default
Processor:	default

Desmarque a opção “case sensitive”
Marque a opção “save settings on exit”
Selecione exit e saia do programa.

Escrevendo e compilando o primeiro programa

Com qualquer editor de texto que salve sem formatção(bloco de notas, Word-“salvar como somente texto”,e outros) crie um arquivo de nome teste1.asm e salve-o com as seguintes informações:

```
        listp=16F84             ;define o processador
        radix   dec
        org     0
inicio:
        movlw 10H
        addlw  20H
        goto    inicio
        END
```

Compilando o primeiro programa

Salve e execute o MPASMWIN

No campo “source file name” escreva: c:\MPASMWIN\teste1.asm e selecione o botão “ASSEMBLE”. Uma janela mostrará o trabalho do compilador, indicando se houve erros e quantos foram, mensagens, ...

Selecione OK e pronto.

O compilador irá gerar os seguintes arquivos:

teste1.cód – possui informações para o simulador, emulador, ...
teste1.err – lista de erros, se houverem.
teste1.hex – fornece o arquivo no formato hexa, para o gravador
teste1.lst – mostra o trabalho e o arquivo gerado, com os includes que houverem, e o código gerado.

Estude bem o arquivo teste1.lst. É muito útil no aprendizado.