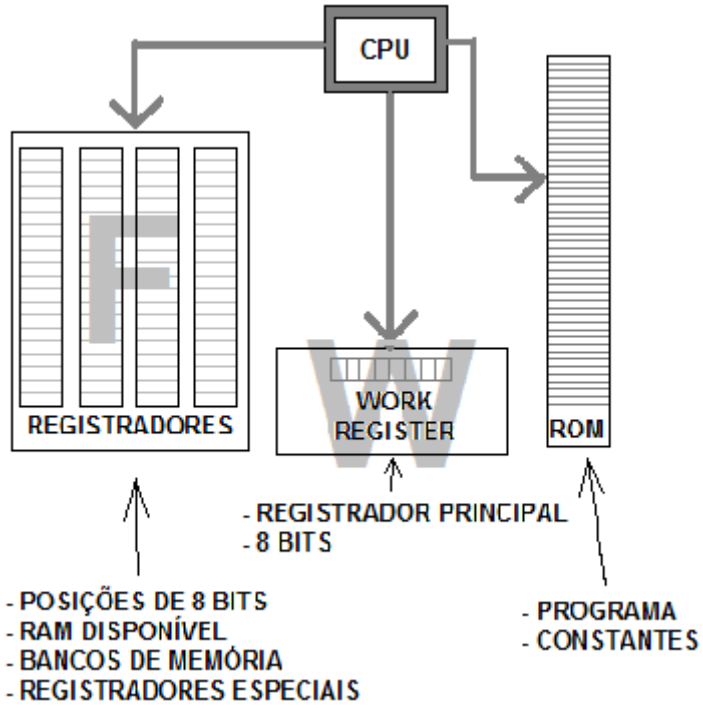
LINGUAGEM DE PROGRAMAÇÃO – ASSEMBLY (continuação)

O WORK REGISTER

Existem 3 tipos de informações manipuladas pelas instruções do PIC: **Registradores de uso geral** (isso inclui memória RAM disponível e registradores de acesso e configuração do hardware), **constantes** (que são dados provenientes da memória ROM) e o **Registrador Principal** (aqui chamado de Work Register).

Muitas operações somente podem ser realizadas com o uso do WORK REGISTER ou **W**. Se você for bom observador, deve ter notado que as instruções não permitem a operação envolvendo diretamente duas posições de memória. Por exemplo, não é possível se movimentar um dado da memória RAM para outra posição da memória RAM com o uso de uma única instrução. Será necessário movimentar este dado da posição de memória inicial para o W, e depois será necessário movimentar W para outra posição de memória.



Veja o exemplo comentado abaixo :

```
ROT1                ; rótulo que identifica o inicio do programa
       BSF STATUS,RP0 ; muda para o banco de memória 01
       MOVLW 0xFF     ; carrega o valor 255 (em hexadecimal FF) em W
       MOVWF TRISB    ; descarrega o valor de W para o registrador TRISB
       BCF STATUS,RP0 ; volta para o banco de memória zero
       MOVF PORTB,W   ; move o valor de PORTB para W
       ADDLW 0x05     ; adiciona 5 ao W
       MOVWF PORTC    ; move o valor de W para PORTC
       GOTO ROT1      ; volta para o inicio do programa
       ...
```

As instruções em destaque utilizam o Work Register. Na segunda linha um valor constante (FFh) é carregado no Work Register. Na linha seguinte, este valor é transferido de W para o Registrador TRISB (Neste caso o Work Register está sendo usado para colocar um valor constante em um Registrador). Na quinta linha, uma instrução é transferida de PORTB para W, e na linha seguinte um valor constante é adicionado em W. Na última linha, o valor de W é movido para o File Register PORTC.

DIRETIVAS DE MONTAGEM

Nos programas podemos utilizar linhas de configuração do montador que servem para orientar a geração do arquivo executável. Estas linhas são chamadas de "diretivas de montagem", e não geram necessariamente código executável.
As mais usadas :
- DEFINE : Define um símbolo que será substituído por uma expressão no programa. Ex:
#define TECLA PORTD, 3
Sempre que a palavra TECLA for utilizada no programa, será substituída pela expressão PORTD,3.

- END : (não usa sustenido e deve aparecer após a primeira coluna) identifica que o final de um programa chegou. Ex:
END

- EQU : Cria uma definição de valores para o montador. Desta forma, podemos associar uma área de memória (geralmente de uso geral) a um nome. Ex:
DISP EQU 0x06

- INCLUDE : Inclui (ao montar) um outro arquivo. Útil para aproveitar o mesmo arquivo para vários programas. Geralmente existe um INCLUDE obrigatório, com informações sobre a configuração do microcontrolador a ser utilizado. Ex :
#INCLUDE <P16F877.INC>

- ORG : Especifica qual o endereço inicial para montagem do programa (onde o programa começa). Ex:
ORG 0x0000

Para saber mais sobre as diretivas de compilação do MPLAB, pesquise no livro [PIC02] na página 327.

Exercícios de fixação :

1. **Associe as colunas**

(a) OPTION_REG	(__) Área de memória para uso geral
(b) BSF	(__) Instrução que testa um bit de um registrador, pulando a próxima instrução se o bit estiver acionado.
(c) RLF	(__) Instrução que causa um desvio do programa.
(d) 20h até 7Fh	(__) Instrução que causa um deslocamento de bits de um registrador para a esquerda.
(e) BTFSS	(__) Instrução que aciona um bit de um registrador.
	(__) Registrador encontrado na posição de memória 81h
(f) GOTO	

2. **Cite 3 instruções que realizam operações bit a bit.**
.....
.....

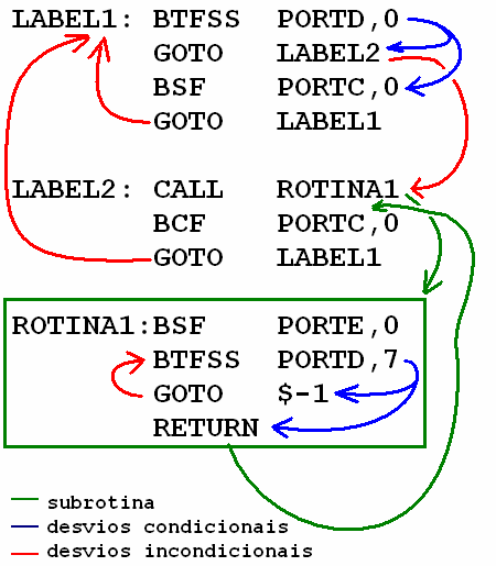
3. Qual a largura, em bits, do registrador principal do microcontrolador estudado neste tópico?
( ) um bit ( ) 8 bits ( ) 16 bits ( ) 32 bits

4. Como pode ser realizado o acesso a um pino do microcontrolador para buscar o estado de um sensor ?
( ) Verificando o estado do bit correspondente ao pino no registrador adequado
( ) Há uma instrução em assembly para cada pino do microcontrolador. Basta usá-la.
( ) Através do registrador principal (W)
( ) Não há como se testar um pino do microcontrolador
( ) Pela instrução BSF

RÓTULO e DESVIOS

Quando esboçamos um programa, seja em descrição narrativa, seja em pesudocódigo ou mesmo através de um fluxograma, observamos a presença de estruturas condicionais e de repetição. Estas estruturas são implementadas através de DESVIOS condicionais. Em outras palavras, em determinados pontos do programa, temos que seguir um NOVO RUMO. Para identificar os pontos do programa para onde deverão ser efetuados estes desvios, existem os RÓTULOS.

No exemplo ao lado, os rótulos LABEL1, LABEL2 e ROTINA1 identificam pontos do programa para onde deve ser desviado o fluxo de execução do programa no momento da execução. Observe que a instrução de desvio (GOTO) e de chamada de subrotina (CALL) são usadas para promover os desvios incondicionais. **No caso da instrução CALL, o retorno para o ponto de chamada é realizado pela instrução RETURN,** e o retorno será automaticamente realizado para a linha seguinte do ponto de chamada, ou seja, abaixo do último CALL realizado.



Em todos os casos, o desvio é realizado para um ponto do programa demarcado pelo rótulo especificado. Os rótulos devem sempre ser escritos na 1ª coluna de texto (mais a esquerda), e as instruções devem começar em uma margem mais avançada (mais a direita).

Podem ser usados dois pontos juntos ao nome dos rótulos, embora isso seja uma característica relacionada com o programa montador escolhido. (no MPLAB isso é opcional, e portando não usaremos dois pontos em nossos exemplos)

Importante : Um rótulo deve ser determinado **por um único nome**, ou seja, **uma única palavra**. ***Não use espaços no nome de um rótulo***. Por exemplo, um rótulo chamado ABRIR VÁLVULA **não** é válido, mas um rótulo ABRIRVALVULA ou ABRIR_VALVULA é válido. ***Evite também usar cedilha, acentuação ou outros caracteres não alfabéticos, pois isso pode gerar erros na montagem***.

As instrução de desvio incondicional em assembly mais utilizada é o GOTO. Trata-se de um desvio onde sempre que esta instrução é executada, a seqüência de execução é interrompida, e o programa sofre um desvio para um novo ponto.

A instrução pode ser usada junto a um rótulo, como no exemplo acima citado, ou pode ser usada junto aos parâmetros $-x ou $+x, onde x é o número de instruções a deslocar. Por exemplo, a instrução GOTO $-3 causa um retorno de 3 linhas no programa. No exemplo anterior, a instrução GOTO $-1 retorna uma linha.

Para o programa acima, poderíamos usar a instrução GOTO $+3 no lugar da instrução GOTO LABEL2. Observe que esta técnica só é interessante quando forem executados saltos muito pequenos (de 2 ou 3 instruções), e que linhas em branco não são contadas. Para desvios maiores se aconselha o uso de rótulos, para não haver problemas nas possíveis modificações de programa.

Para desvios condicionais, geralmente usamos as instruções BTFSS, BTFSZ, DECFSZ. Estas instruções permitem realizar desvios condicionais através do recurso de ignorar ou não a próxima instrução. Em poucas palavras, com estes testes, podemos ter o desvio para a segunda linha após a instrução, ou podemos não ter o desvio.

Por exemplo, se executarmos a instrução que testa se um bit está ativado, seguido de duas linhas com instruções de desvios incondicionais, da seguinte forma:

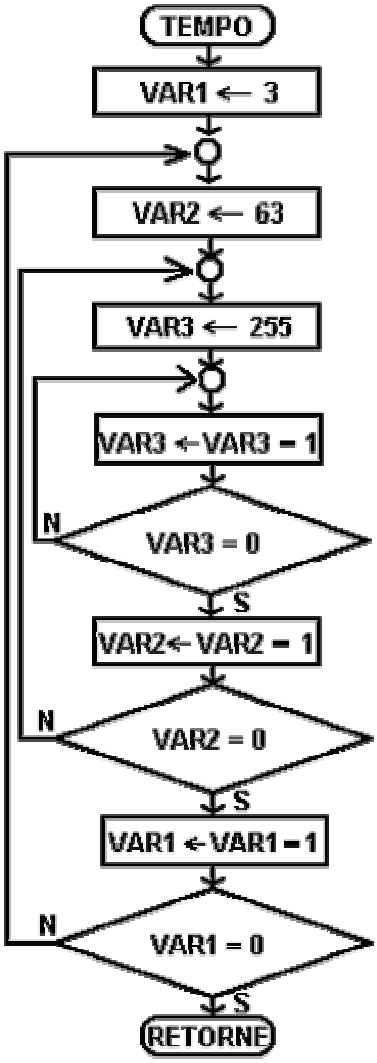
```
BTFSS PORTD,1
GOTO  ROTULO1
GOTO  ROTULO2
```

Caso o primeiro teste seja verdadeiro, ou seja, caso o bit 1 do registrador PORTD esteja ativado, a segunda linha (GOTO ROTULO1) não será executada, pulando-se a execução do programa para a instrução GOTO ROTULO2.

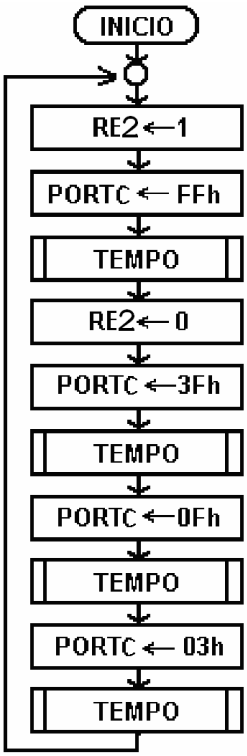
Porém, caso a instrução BTFSS PORTD,1 verifique que o bit testado está desativado, a segunda linha será executada normalmente, e o programa desviará para o ROTULO1.

ESCREVENDO PROGRAMAS ASSEMBLY

Para melhor compreensão, apresentaremos inicialmente um programa descrito em linguagem alto-nível (um fluxogama), e posteriormente a tradução do mesmo em linguagem assembly. Na seqüência, descreveremos as partes do programa.



Rotina de tempo. Chamada a partir da função principal, esta rotina tem por objetivo causar um atraso de alguns milisegundos



Rotina principal. Este programa tem por objetivo causar um efeito visual nos LEDS conectados ao PORTC (saídas digitais da estação CUSCOPIC). Os valores são jogados no PORTC em hexadecimal, o que nos permite perceber que o efeito visual é dado por todos os leds ligados (FF = 11111111), e após um tempo os dois leds mais significativos são desligados (3F = 00111111), seguido de metade dos leds desligados (0F = 00001111), e finalmente os 6 leds mais significativos desligados (03 = 00000011). Note também que o pino RE2 permanece um período de tempo ativado. Como este pino está conectado ao BUZZER (alerta sonoro), deve-se esperar que o programa resulte também um beep intermitente.

Transcrevendo o programa em assembly

PROGRAMA	*COMENTÁRIOS*

`; OBJETIVO : EFEITO VISUAL NO PORTD, COM BEEP`
`; AUTOR : DANIEL CORTELETTI`
`; FEVEREIRO DE 2003`
`; REVISADO E MODIFICADO EM JULHO DE 2009`
`; CENTRO TECNOLÓGICO DE MECATRONICA SENAI`
	Linhas iniciadas por um ponto e vírgula são "comentários", ou seja, servem de documentação para que possamos registrar idéias (texto) no programa, sem interferir na montagem do código. Este recurso impacta diretamente sobre a manutenibilidade do programa.

`LIST p=16F877`	Esta linha indica o microcontrolador alvo
`#include "P16F877.INC"`	Aqui, é solicitada a inclusão (que acontece no momento da montagem) do arquivo P16f877.INC, onde estão pré-incluídas algumas instruções básicas.
`__config _CP_OFF & _PWRTE_OFF & _WDT_OFF & _XT_OSC`	Define os fuses

`VAR1 equ 0X20`
`VAR2 equ 0X21`
`VAR3 equ 0X22`
	Aqui são definidas 3 variáveis de 8 bits (VAR1, VAR2 e VAR3 usando os endereços de memória 20H a 22H. Verifique que estas posições da RAM do Microcontrolador estão livres.

`org 0`	Esta linha aparentemente inútil poderá ser necessária quando utilizarmos rotinas de interrupção. Aqui se define em que posição da memória ROM o programa (código executável) começa.
`goto inicio`	Esta linha causa um desvio para o início do programa propriamente dito, pulando a rotina abaixo descrita.

`tempo movlw 0x03`
`movwf VAR1`
`temp1 movlw 0x3F`
`movwf VAR2`
`temp2 movlw 0xFF`
`movwf VAR3`
`decfsz VAR3,F`
`goto $-1`
`decfsz VAR2,F`
`goto temp2`
`decfsz VAR1,F`
`goto temp1`
`return`
	Aqui está descrita a rotina de "tempo". Como foi descrito anteriormente no fluxograma, esta rotina tem por objetivo "queimar tempo", ou seja, ao executar esta rotina, o microcontrolador irá perder ciclos de execução do programa para executar 3 laços aninhados, resultando aproximadamente 50.000 iterações (voltas). Para cada uma destas "voltas" executadas nesta rotina, o microcontrolador perde aproximadamente 2 microsegundos, totalizando em um atraso (para a rotina) de aproximadamente 100.000 microsegundos.

`inicio bsf STATUS, RP0` `bcf STATUS, RP1`	A partir daqui inicia a rotina principal do programa. Esta parte é executada somente uma vez, e se destina à preparação dos PORTs do microcontrolador. Para isso, teremos que acessar o banco de memória 1 do microcontrolador (ação executada através da manipulação dos bits STATUS RP0 e RP1.

`movlw 0xff`
`movwf TRISD`
`movlw 0x00`
`movwf TRISC`
`movwf TRISE`
`bcf STATUS, RP0`
	No banco de memória 1, acessamos os registradores relativos à configuração de direção, definindo se um pino é entrada ou saída. São registradores de direção TRISA, TRISB, TRISC, TRISD e TRISE. Estes registradores configuram, respectivamente, o PORTA, PORTB, PORTC, PORTD e PORTE. Ao final do processo, o registrador STATUS é reconfigurado, de forma a retornar o acesso ao banco de memória padrão (banco 0).

`comeco bsf PORTE,2`
`movlw 0xff`
`movwf PORTC`
`call tempo`
`bcf PORTE,2`
`movlw 0x3f`
`movwf PORTC`
`call tempo`
`movlw 0x0f`
`movwf PORTC`
`call tempo`
`movlw 0x03`
`movwf PORTC`
`call tempo`
`goto comeco`
`END`
	Esta é a parte "cíclica" do programa, onde se encaixa a tradução do fluxograma. Este código será executado ininterruptamente, em um laço infinito. Observe que, entre as instruções aqui definidas, existem algumas chamadas a subrotina **tempo** (através da instrução call). Observe também que no final do programa surge a diretiva END, indicando o final do texto do programa. Este END não indica que o programa termina, mas sim que o arquivo onde está escrito o programa termina. Em outras palavras, todo o programa deverá possuir a diretiva END no final, mesmo que o programa não tenha fim.

Exercícios :

1. Para que serve a instrução **goto** que aparece em vários programas ?
2. Algumas linhas no inicio do programa começam com um ponto e vírgula. Por que ?
3. Que tipos de informações são manipuladas por instruções dos microcontroladores PIC?
4. Alguns operandos possuem nomes como PORTC, PORTD, PORTE. O que eles são ? E o que representam ?
5. No programa 4, existe uma linha "**movlw 0x03**". Em outra linha, aparece "**movlw 0xFF**". O que significa este operando (parâmetro) que começa por um ZERO e um X ?
6. O que faz a instrução GOTO $-1 presente na rotina de tempo?
7. Para que serve a instrução BSF PORTE,2? (qual o efeito na estação CUSCOPIC)

Bom trabalho.