LINGUAGEM DE PROGRAMAÇÃO - ASSEMBLY - PARTE I

Assembly - o que é

"Assembly" significa montagem. "Assembler" significa montador. Montagem é o ato de transformar uma seqüência de código fonte (texto) em código objeto (linguagem de máquina), e montador é o programa que faz isso.

Observe no diagrama abaixo que o PROGRAMADOR escreve uma lista de comandos em forma de texto, onde cada linha realiza uma função específica.

No entanto, para o microcontrolador não entende estes comandos em modo texto. Eles precisam antes ser traduzidos para uma linguagem binária (linguagem de máquina), e isso é feito pelo programa montador (ASSEMBLER).

Programar diretamente em linguagem de máquina até é possível, mas é muito, mas muito mais difícil do que programar em assembly.

Usando esta linguagem e um programa montador, o programador não escreve em linguagem de máquina, e sim em uma linguagem textual, facilitando a construção dos programas. Embora fique mais fácil do que programar direto em linguagem de máquina, programar em ASSEMBLY ainda é uma das formas mais "difíceis" de programação, sendo conhecida como a linguagem de programação de "mais baixo nível".

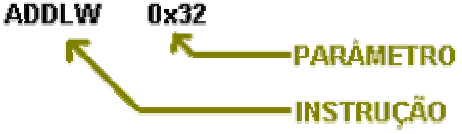
O arquivo fonte do diagrama acima (aquela lista de comandos digitada pelo programador) é composta de instruções (**mnemônicos**)**, parâmetros, rótulos, comentários e diretivas**, e após são transformados em linguagem de máquina por um programa **montador**.

Vamos conhecer cada um destes componentes do programa.

INSTRUÇÃO : É o nome dado a uma operação que o microcontrolador pode realizar. Por exemplo, se o microcontrolador pode realizar a soma de dois valores, dizemos que existe no mínimo uma instrução para soma. No caso do microcontrolador PIC, as instruções que realizam soma são duas, a ADDWF e a ADDLW. Você consegue descobrir que instruções estão disponíveis para programação assembly no datasheet do microcontrolador em questão.

MNEMÔNICO : É uma representação textual de uma instrução. As instruções são, na verdade, códigos binários, e para serem entendidos pelos programador devem ser representados na forma textual. Se os mnemônicos não fossem utilizados, teríamos que programar assembly utilizando códigos numéricos difíceis de memorizar. Desta forma, uma das instruções de adição citadas anteriormente possui o mneumônico ADDWF, e sua representação binária é 000111xxxxxxxx onde x .

PARÂMETROS : São as informações manipuladas por uma instrução. Isso é necessário sempre que precisarmos informar à instrução quais os elementos envolvidos na operação. Por exemplo, se desejarmos somar dois valores, a instrução poderá ser ADDWF ou ADDLW, mas *os valores a serem somados* também deverão ser informados, sendo estes últimos os que chamamos de "parâmetros."



DIRETIVAS : São linhas que determinam como o programa montador irá trabalhar. Não geram efeito direto no código binário gerado. Por exemplo, a diretiva **LIST p=16F877** determina qual o microcontrolador que será usado.

RÓTULOS : São nomes dados as linhas do programa, e servem para que em uma instrução de desvio possa se determinar o ponto para onde se deseja ir no programa. Os

rótulos sempre são alinhados na coluna 0 (sem espaços antes do mesmo), enquanto que as instruções devem ser escritas após uma margem (obrigatoriamente após a coluna 0).

COMENTÁRIOS : São trechos de texto escritos após um sinal de ponto e vírgula (;). São úteis para que possamos adicionar pequenos lembretes no programa, facilitando a manutenção futura. Não interferem no tamanho do programa binário gerado.

Outros termos importantes :

MONTADOR : É o programa que transforma um programa fonte assembly em um programa executável. Um exemplo é o MPASM, que faz parte do MPLAB, uma ferramenta de desenvolvimento distribuída pela MicroChip (fabricante dos microcontroladores PIC)

AS INSTRUÇÕES

Por se tratar de um microcontrolador RISC, o PIC oferece um número reduzido de instruções. No entanto, ainda podemos dividir as instruções utilizadas pela família 16 dos microcontroladores PIC em 6 grupos :

- Instruções para manipulação de bytes de memória (B)
- Instruções para manipulação de bits de memória (b)
- Desvios incondicionais (Di)
- Desvios condicionais (D)
- Instruções com valores constantes (K)
- Instruções de controle (G)

Instrução e Parâmetros Mneumônicos		Descrição	Tipo	Ciclos	Bits de status afetados
ADDWF	f, d	Adição : W + F.	B	1	C, DC, Z
ANDWF	f, d	E binário (AND) entre W e F, bit a bit.	B	1	Z
CLRF	f	Zera todos os bits de F.	B	1	Z
CLRW		Zera todos os bits de W.	B	1	Z
COMF	f, d	Complemento de F. (bits com valores invertidos no byte)	B	1	Z
DECF	f, d	Decrementa F	B	1	Z
DECFSZ	f, d	Decrementa F e pula próxima linha se resultar zero	B,Dc	1 (2)	
INCF	f, d	Incrementa F	B	1	Z
INCFSZ	f, d	Incrementa F e pula próxima linha se resultar zero	B,Dc	1 (2)	
IORWF	f, d	OU inclusivo (OR) de W com F	B	1	Z
MOVF	f, d	Move F (geralmente usado para mover F para W)	B	1	Z
MOVWF	f	Move W para F	B	1	
NOP		Operação nula. Nada é executado.	G	1	
RLF	f, d	Rotaciona F para esquerda com Carry Flag	B	1	C
RRF	f, d	Rotaciona F para direita com Carry Flag	B	1	C
SUBWF	f, d	Subtrai W de F : (f-W)	B	1	C,DC,Z
SWAPF	f, d	Troca os nibbles de f. Ex: (0xA3, após swap fica, 0x3A)	B	1	
XORWF	f, d	Ou exclusivo (XOR) entre W e F	B	1	Z

BCF	f, b	Apaga (clear) um bit de F	b	1	
BSF	f, b	Liga (set) um bit de F	b	1	
BTFSC	f, b	Testa um bit de F, pulando se for zero	b,Dc	1 (2)	
BTFSS	f, b	Testa um bit de F, pulando se for um	b,Dc	1 (2)	
ADDLW	k	Adiciona uma constante K em W	B	1	C,DC,Z
ANDLW	k	E (and) lógico de uma constante com W	B	1	Z
CALL	k	Faz uma chamada a uma subrotina	Di	2	
CLRWDT		Limpa o Watchdog Timer (relógio do cão de guarda)	G	1	~TO, ~PD
GOTO	k	Vá para. Um desvio para um outro ponto do programa.	Di	2	
IORLW	k	Ou inclusivo (OR) de uma constante com W	B	1	Z
MOVLW	k	Move uma constante para W	B	1	
RETFIE		Retorna de uma interrupção	Di	2	
RETLW	k	Retorna de uma subrotina, movendo uma const. para W	B,Di	2	
RETURN		Retorna de uma subrotina	Di	2	
SLEEP		Vai para o modo standby	G	1	~TO, ~PD
SUBLW	k	Subtrai uma constante de W	B	1	C,DC,Z
XORLW	k	Ou exclusivo (OR) entre W e uma constante	B	1	Z

Como entender a tabela acima :

1ª Coluna : INSTRUÇÃO - Descreve todas as instruções utilizadas pelo microcontrolador PIC16F877. As instruções geralmente possuem nomes relacionados a suas funções. Por exemplo, GOTO lembra GO TO, que em inglês significa VÁ PARA ... Já SLEEP significa DORMIR. ADDLW lembra ADD que em inglês significa adicionar.... e assim por diante.

2ª Coluna : PARÂMETROS - Descreve os operandos utilizados pela instrução. Nesta coluna aparecem as letras f, d, b, k.

O **(f)** identifica que o parâmetro deve ser uma posição da memória RAM interna (**que chamaremos de registradores**). Os registradores serão explicados em breve e expressos em uma tabela.

O **(d)** identifica um parâmetro de destino, e pode valer W ou F. W é o registrador principal, e F é qualquer outro registrador.

O **(b)** é um parâmetro de identificação de um bit (0 a 7). Por exemplo, **BSF PORTD,0** onde o (b) vale 0 ativa o bit menos significativo (bit 0) do registrador PORTB.

O **(k)** identifica que o parâmetro em questão é uma constante (rótulo ou valor fixo). Por exemplo, **MOVLW 10** onde o valor de K é 10, move a constante 10 para o registrador principal.

3ª Coluna : DESCRIÇÃO - Descreve a função dos operandos.

4ª Coluna - TIPO. Define o grupo onde a instrução se encaixa. Veja a legenda no texto acima da tabela.

5º Coluna - CICLOS - Uma instrução pode consumir 1 ou 2 ciclos de máquina. Cada ciclo de máquina, no caso dos microcontroladores PIC16F8xx, correspondem a 4 pulsos de clock. Portanto, se o cristal utilizado no microcontrolador for de 4MHz, ocorrerão 1MegaCiclos por segundo, ou seja, 1 milhão de ciclos por segundo (também usa-se 1mips - 1 milhão de instruções por segundo). Algumas instruções, portanto, demorarão 1/1000000 de segundos (1 microsegundo) para serem executadas, e outras demorarão 2/1000000 segundos (2 microsegundos). Algumas instruções (como os desvios condicionais) podem demorar 1 ou 2 ciclos, dependendo da condição avaliada pela instrução.

6ª Coluna - BITS DE STATUS AFETADOS - Inicialmente, devemos entender o que são BITS DE STATUS. De uma forma resumida, são "indicadores" existentes na memória do microcontrolador que registram informações sobre as operações realizadas (Exemplo :se a última operação resultou em zero ou não, se houve estouro no valor computado, etc...). Esta coluna visa descrever quais destes BITS DE STATUS são afetados pela instrução. Para saber mais sobre estes bits de status, procure bibliografia complementar.

Exercício : Observando o trecho de programa abaixo, e tendo em mãos a tabela a pouco descrita, procure descobrir o que cada linha significa.

```
volta
    btfss PORTA,1
    goto deslig
ligado
    movlw 0x0F
    movwf PORTD
    goto volta
deslig
    movlw 0xAA
    movwf PORTD
    goto volta
```

REGISTRADOR

É o nome utilizado para identificar uma posição de memória interna do microcontrolador. No caso do microcontrolador PIC16F877, possuímos capacidade de acesso interno a 512 bytes de memória. Cada byte (8 bits) é um registrador. Temos, portanto, 512 registradores. No entanto, alguns destes registradores são utilizados para fins específicos, estando diretamente relacionados a periféricos internos do microcontrolador (como conversores AD, entradas e saídas digitais, configuração dos periféricos, etc...), e outros destes registradores não são fisicamente implementados (são "buracos" deixados para futuras melhorias do projeto do microcontrolador).

Outros registradores são os denominados "General Purpose Registers", ou "Registradores de Uso Geral". São posições de memória livres, que podem ser utilizados para armazenamento temporário de valores (variáveis). Como são parte da memória RAM do microcontrolador, **todos os dados armazenados nos registradores são voláteis,** ou seja, são perdidos ao se desligar o mesmo.

Observe, na tabela que segue, os FILE REGISTERS (ou Arquivos Registradores) do PIC16F877

FIGURE 2-3: PIC16F877/876 REGISTER FILE MAP

File Address	Bank 0	File Address	Bank 1	File Address	Bank 2	File Address	Bank 3
00h	Indirect addr.(*)	80h	Indirect addr.(*)	100h	Indirect addr.(*)	180h	Indirect addr.(*)
01h	TMR0	81h	OPTION_REG	101h	TMR0	181h	OPTION_REG
02h	PCL	82h	PCL	102h	PCL	182h	PCL
03h	STATUS	83h	STATUS	103h	STATUS	183h	STATUS
04h	FSR	84h	FSR	104h	FSR	184h	FSR
05h	PORTA	85h	TRISA	105h		185h	
06h	PORTB	86h	TRISB	106h	PORTB	186h	TRISB
07h	PORTC	87h	TRISC	107h		187h	
08h	PORTD(1)	88h	TRISD(1)	108h		188h	
09h	PORTE(1)	89h	TRISE(1)	109h		189h	
0Ah	PCLATH	8Ah	PCLATH	10Ah	PCLATH	18Ah	PCLATH
0Bh	INTCON	8Bh	INTCON	10Bh	INTCON	18Bh	INTCON
0Ch	PIR1	8Ch	PIE1	10Ch	EEDATA	18Ch	EECON1
0Dh	PIR2	8Dh	PIE2	10Dh	EEADR	18Dh	EECON2
0Eh	TMR1L	8Eh	PCON	10Eh	EEDATH	18Eh	Reserved(2)
0Fh	TMR1H	8Fh		10Fh	EEADRH	18Fh	Reserved(2)
10h	T1CON	90h		110h	General Purpose Register 16 Bytes	190h	General Purpose Register 16 Bytes
11h	TMR2	91h	SSPCON2	111h		191h	
12h	T2CON	92h	PR2	112h		192h	
13h	SSPBUF	93h	SSPADD	113h		193h	
14h	SSPCON	94h	SSPSTAT	114h		194h	
15h	CCPR1L	95h		115h		195h	
16h	CCPR1H	96h		116h		196h	
17h	CCP1CON	97h		117h		197h	
18h	RCSTA	98h	TXSTA	118h		198h	
19h	TXREG	99h	SPBRG	119h		199h	
1Ah	RCREG	9Ah		11Ah		19Ah	
1Bh	CCPR2L	9Bh		11Bh		19Bh	
1Ch	CCPR2H	9Ch		11Ch		19Ch	
1Dh	CCP2CON	9Dh		11Dh		19Dh	
1Eh	ADRESH	9Eh	ADRESL	11Eh		19Eh	
1Fh	ADCON0	9Fh	ADCON1	11Fh		19Fh	
20h	General Purpose Register 96 Bytes	A0h	General Purpose Register 80 Bytes	120h	General Purpose Register 80 Bytes	1A0h	General Purpose Register 80 Bytes
7Fh		EFh		16Fh		1EFh	
		F0h	accesses 70h-7Fh	170h	accesses 70h-7Fh	1F0h	accesses 70h - 7Fh
		FFh		17Fh		1FFh	

Unimplemented data memory locations, read as '0'.
* Not a physical register.

Note 1: These registers are not implemented on the PIC16F876.
2: These registers are reserved, maintain these registers clear.

(Tabela retirada do datasheet do microcontrolador PIC16F877)

Considerações importantes sobre os registradores :

1. Estão divididos em BANCOS DE MEMÓRIA. Na programação em assembly, precisamos selecionar o banco de memória antes de acessar um determinado registrador. Isso pode ser feito através do registrador STATUS (que aparece em todos os bancos de memória). Nele existem 2 bits que permitem definir qual o banco de memória que está ativo (paginação).
2. Alguns registradores são de uso geral, liberados para uso com variáveis. Ex : registradores 20h a 7Fh do BANK 0.
3. Registradores como o STATUS e o OPTION_REG são utilizados para configuração de características do MC, como mudança de banco de memória, velocidade de incremento de temporizadores, etc... Recomenda-se um aprofundamento maior, através da bibliografia sugerida.
4. Para uma boa programação assembly, o conhecimento sobre estes registradores é indispensável.

OS REGISTRADORES QUE MAIS UTILIZAREMOS NESTA FASE DO CURSO SÃO :

PORTA, PORTB, PORTC, PORTD e PORTE = Registradores que representam o estado dos pinos do PIC. Por exemplo, se você desejar acionar o pino RC3 do microcontrolador, deverá ligar o bit 3 do registrador PORTC.
Obs: Nem todos os bits dos registradores citados possuem representação física. Ex : PORTE, bit 4.

STATUS : este registrador possui 2 bits que servem para definir o BANCO DE MEMÓRIA (VEJA TABELA ACIMA) que será usado nas próximas operações. Também possui informações referentes a sinalizadores de erro, estouro de memória, etc...

TRISA, TRISB, TRISC, TRISD, TRISE = Serve para definir se um determinado PINO será de entrada ou saída. Isso é importante, pois é desta forma que o PIC irá saber se você quer LER sinal de um pino (que flutuará conforme tensão de entrada) ou se quer ENVIAR um sinal para este pino (mantendo assim o nível de tensão desejado).

ADCON1 = Serve para configurar o conversor AD (ligar ou desligar, especificar o padrão de conersão, etc.)

Exercícios :

1 - Aponte as instruções, os registradores, parâmetros, rótulos, comentários e diretivas encontrados no programa abaixo.

```
; AUTOR : DANIEL CORTELETTI
; FEVEREIRO DE 2003
; CENTRO TECNOLÓGICO DE MECATRONICA SENAI

        LIST    p=16F877
   #include "P16F877.INC"
   __config _CP_OFF & _PWRTE_OFF & _WDT_OFF & _XT_OSC
VAR1    equ     20
VAR2    equ     21
VAR3    equ     22
CONT    equ     23
        org     0
        goto    inicio

inicio
        bcf     STATUS, RP0
        bcf     STATUS, RP1
        clrf    PORTA
        bsf     STATUS, RP0
        movlw   0x06
        movwf   ADCON1
        movlw   0xff
        movwf   TRISA
        movlw   0x00
```

```
            movwf   TRISD
            bcf     STATUS, RP0

comeco      movlw   0x01
            movwf   PORTD
            movlw   0x07
            movwf   CONT
volta1      call    tempo
            rlf     PORTD,f
            decfsz  CONT,f
            goto    volta1
            movlw   0x07
            movwf   CONT
volta2      call    tempo
            rrf     PORTD,f
            decfsz  CONT,f
            goto    volta2
            goto    comeco

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
```

2 - Para que serve o registrador TRISD ?

3 - Para que serve o registrador PORTD ?

4 - Que instruções deveria ser executadas pra que o pino RD5 fosse acionado ?

5 - A memória do microcontrolador PIC é segmentada em quantos bancos ?

7 - Que tipo de programa você vai precisar para programar em assembly ?
( ) Um editor de textos como o WORD
( ) Um editor de imagens como o PAINT
( ) Um montador como o MPLAB
( ) Um navegador como o INTERNET EXPLORER
( ) Uma planilha eletrônica como o Excel

8 - Ao escrever um programa assembly, é possível se realizar anotações (trechos de texto) no meio do programa. Como isso é possível ? Isso aumenta o tamanho do programa binário (arquivo HEX) gerado ?