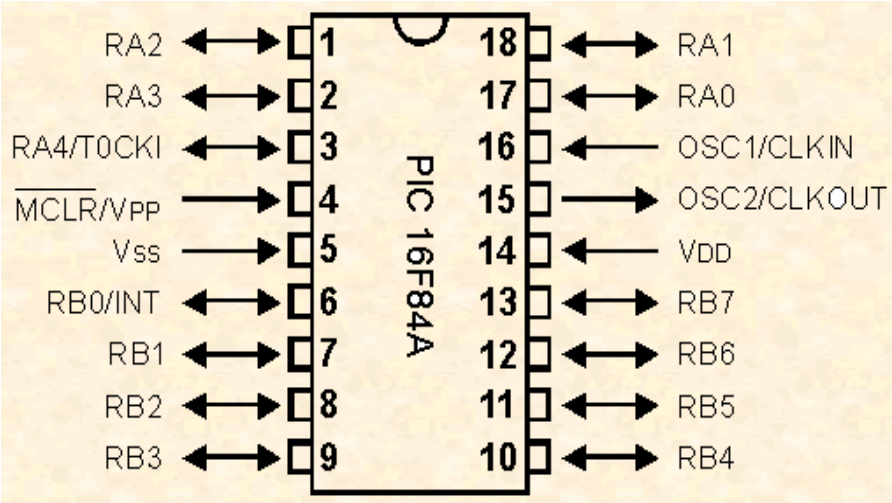


Diagrama de Pinagem do PIC16F84

	BANCO 0	BANCO 1	
000h	INDF	INDF	080h
001h	TMR0	OPTION	081h
002h	PCL	PCL	082h
003h	STATUS	STATUS	083h
004h	FSR	FSR	084h
005h	PORTA	TRISA	085h
006h	PORTB	TRISB	086h
007h	**Sem Utilização**	**Sem Utilização**	087h
008h	EEDATA	EECON1	088h
009h	EEADR	EECON2	089h
00Ah	PCLATH	PCLATH	08Ah
00Bh	INTCON	INTCON	08Bh
00Ch 004Fh	Uso Geral 68Bytes	Espelho da Memória do BANCO 0	08Ch 0CFh
050H 07Fh	Não Disponivel	Não Disponivel	0D0Fh 0FFh

Organização da memória de dados do PIC16F84

Registradores

Registrador – INDF

Registrador: INDF	Endereços: 000h e 080h
Valor do endereçamento indireto	

Registrador – TMR0: Registrador Temporizador / Contador de 8 bits

Registrador: TMR0				Endereço: 001h			
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Contador automático de 8 bits							

Registrador – PCL: Registrador dos 8 bits Menos Significativos do Contador de Programa.

Registrador: PCL				Endereço: 002h e 082h			
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Parte baixa do contador de programa.							

Registrador – STATUS

Registrador: STATUS				Endereço: 003h e 083h			
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
IRP	RP1	RP0	/T0	/PD	Z	DC	C
Parte baixa do contador de programa.							

***IRP**:* Seletor de banco de memória usado para endereçamento indireto

Bit	Descrição
0	Bancos 0 e 1 (000h a 0FFh)
1	Bancos 2 e 3 (100h a 1FFH)

OBS: Para o PIC16F84 o bit do IRP e sempre 0, pois so e utiliza 2 BANCOS, o BANCO 0 e o BANCO 1.

***RP1 e RP0**:* Seletor de banco de mémoria usando para endereçamento direto.

RP1 Bit	RP0 Bit	Descrição
0	0	Banco 0 (000h a 07Fh)
0	1	Banco 1 (080h a 0FFh)
1	0	Banco 2 (000h a 17Fh)
1	1	Banco 3 (180h a 1FFh)

OBS: Para o PIC16F84 o bit do RP1 e sempre 0, apenas sera mudado o bit do RP0 para se selecionar entre o BANCO 0 e BANCO 1.

***/TO**:* Indicação de Time-out

Bit	Descrição
0	Indica que ocorreu estouro de WDT
1	Indica que ocorreu power-up ou foram executadas as instruções CLRWDT ou SLEEP

***/PD**:* Indicação de Power-down.

Bit	Descrição
0	Indica que a instrução SLEEP foi executada
1	Indica que ocorreu Power-up ou foi executada a instrução CLRWDT

***/Z**:* Indicação de Zero

Bit	Descrição
0	Indica que o resultado da última operação (lógica ou aritmética) não foi zero
1	Indica que o resultado da última operação (lógica ou aritmética) foi zero

***DC**:* Digit Carry / Borrow

Bit	Descrição
0	Indica que a última operação da ULA não ocasionou um estouro de digito
1	Indica que a última operação da ULA ocasionou um estouro (carry) entre os bit 3 e 4, quer dizer, o resultado ultrapassou a capacidade dos 4 bits menos significativos. Esse bit de verificação é comumente utilizado quando se trabalha com números de 4 bits.

***C**:* Carry / Borrow

Bit	Descrição
0	Indica que a última operação da ULA não ocasionou um estouro (carry)
1	Indica que a última operação da ULS ocasionou um estouro (carry) no bit mais significativo, quer dizer, o resultado ultrapassou os 8 bits disponiveis.

Registrador – FSR: Registrador de Seleção (File Select Register) é um ponteiro para o registrador INDF.

Registrador: FSR				Endereço: 004h e 084h			
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Ponteiro para endereçamento indireto							

Registrador – PORTA: Registrador de I/O

Registrador: PCL				Endereço: 002h e 082h			
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
-	-	-	R/W	R/W	R/W	R/W	R/W
-	-	-	RA4	RA3	RA2	RA1	RA0

Registrador – PORTAB: Registrador de I/O

Registrador: PCL				Endereço: 002h e 082h			
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
RB7	RB6	RB5	RB4	RB3	RB2	RB1	RB0

Registrador EEDATA

Registrador: EEDATA				Endereço: 008h			
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Dados para escrita e leitura da EEPROM							

Registrador EEADR

Registrador: EEADR				Endereço: 009h			
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Endereço de acesso à EEPROM							

Registrador PCLATH

Registrador: PCLATH				Endereço: 00Ah e 08Ah			
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Porte alta do contador de programa							

Registrador INTCON

Registrador: INTCON				Endereço: 00Bh e 08Bh			
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
GIE	EEIE	TOIE	INTE	RBIE	TOIF	INTF	RBIF

GIE: Chave-geral das interrupções

Bit	Descrição
0	Nenhuma interrupção será tratada
1	As interrupções habilitadas serão tratadas

EEIE: Habilitação da interrupção de escrita na EEPROM (chave individual)

Bit	Descrição
0	A interrupção não será tratada
1	A interrupção será tratada

TOIE: Habilitação da interrupção de estouro do TMRO (chave individual)

Bit	Descrição
0	A interrupção não será tratada
1	A interrupção será tratada

INTE: Habilitação da interrupção externo no pino RB0 (chave individual)

Bit	Descrição
0	A interrupção não será tratada
1	A interrupção será tratada

RBIE: Habilitação da interrupção por mudança de estado nos pinos RB4 a RB7(chave individual)

Bit	Descrição
0	A interrupção não será tratada
1	A interrupção será tratada

TOIF: Identificação da interrupção de estouro da TMRO

Bit	Descrição
0	A interrupção não ocorreu
1	A interrupção ocorreu

INTF: Identificação da interrupção externa no pino RB0

Bit	Descrição
0	A interrupção não ocorreu
1	A interrupção ocorreu

RBIF: Identificação da interrupção por mudança de estado nos pinos RB4 a RB7.

Bit	Descrição
0	A interrupção não ocorreu
1	A interrupção ocorreu

Registrador OPTION

Registrador: OPTION				Endereço: 081h			
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
/RBPU	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0

/RBPU: Habilitação dos pull-ups internos para o PORTB.

Bit	Descrição
0	Pull-ups habilitados para todos os pinos do PORTB configurados como entrada
1	Pull-ups desabilitados

INTDEG: Configuração da borda que gerará a interrupção externa no RB0.

Bit	Descrição
0	A interrupção ocorrerá na borda de descida.
1	A interrupção ocorrerá na borda de subida

TOCS: Configuração do incremento para o TMR0

Bit	Descrição
0	TMR0 será incrementado internamente pelo clock da máquina
1	TMR0 será incrementado internamente pela mudança do pino RA4/T0CKIN

TOSE: Configuração da borda que incrementará o TMR0 no pino RA4/T0CKIN, quando T0CS=1

Bit	Descrição
0	O incremento será na borda de subida
1	O incremento será na borda de descida

PSA: Configuração de aplicação do prescaler (pré-escalador)

Bit	Descrição
0	O prescaler será aplicado ao TMR0
1	O prescaler será aplicado ao WDT

PS2, PS1 e PS0: Configuração do valor do prescaler

Bits 2, 1, 0	TMR0	WDT
000	1:2	1:1
001	1:4	1:2
010	1:8	1:4
011	1:16	1:8
100	1:32	1:16
101	1:64	1:32
110	1:128	1:64
111	1:256	1:128

Registrador TRISA: Registrador de I/0

Registrador: TRISA				Endereço: 085h			
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
-	-	-	R/W	R/W	R/W	R/W	R/W
-	-	-	Ref. RA4	Ref. RA3	Ref. RA2	IRef. RA1	Ref. RA0

Registrador TRISB: Registrador de I/0

Registrador: TRISB				Endereço: 086h			
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Ref. RA4	Ref. RA4	Ref. RA4	Ref. RA4	Ref. RA3	Ref. RA2	IRef. RA1	Ref. RA0

Registrador EECON1

Registrador: EECON1				Endereço: 081h			
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
-	-	-	R/W	R/W	R/W	R/W	R/W
-	-	-	EEIF	WRERR	WREN	WR	RD

EEIF: Idenficação da interrupção de final de escrita da EEPROM

Bit	Descrição
0	Esta interrupção não ocorreu
1	Esta interrupção ocorreu

WRERR: Idenficação de erro durante a escrita na EEPROM

Bit	Descrição
0	Não ocorreu erro, a escrita foi completada
1	Um erro ocorreu por uma escrita não terminada (um reset poder ter ocorrido)

WREN: Habilitação de escrita na EEPROM (bit se segurança)

Bit	Descrição
0	Não disponibiliza a escrita na EEPROM
1	Disponibiliza a escrita na EEPROM

WR: Ciclo de escrita na EEPROM

Bit	Descrição
0	Este bit só pode ser zerado pelo hardware quando o ciclo de escrita termina
1	Inicia o ciclo de escrita.

RD: Ciclo de leitura da EEPROM

Bit	Descrição
0	Este bit só pode ser zerado pelo hardware quando o ciclo de leitura termina
1	Inicia o ciclo de escrita.

Registrador EECON2

Registrador: EECON2	Endereços: 089h
Utilizado para inicializar corretamente a escrita na EEPROM	

NOVA ELETRÔNICA

Conjunto de Instruções

• **Instruções Aritméticas** ADDLW ADDWF SUBLW SUBWF INCF DECF CLRF CLRW	• **Instruções de Movimentação de Dados** MOVLW MOVWF MOVF RLF RRF	• **Instruções Lógicas** ANDLW ANDWF IORLW IORWF XORLW XORWF COMF SWAPF
• **Instruções de atendimento a Rotinas** GOTO CALL RETURN RETLW RETFIE	• **Instruções de controle de Bit e Teste de variáveis** BCF BSF BTFSS BTFSC INCFSZ DECFSZ	• **Instruções de controle ao Processamento** NOP CLRWDT SLEEP

MOVLW — Escrever constante no registro W

Sintaxe:	*[rótulo]* MOVLW **k**
Descrição:	A constante de 8-bits **k** vai para o registro **W**.
Operação:	$\mathbf{k} \Rightarrow (\mathbf{W})$
Operando:	$0 \leq \mathbf{k} \leq 255$
Flag:	-
Número de palavras:	1
Número de ciclos:	1
Exemplo 1:	MOVLW 0x5A
	Depois da instrução: W= 0x5A
Exemplo 2:	MOVLW REGISTRAR
	Antes da instrução: W = 0x10 e REGISTRAR = 0x40
	Depois da instrução: W = 0x40

MOVWF Copiar W para f

Sintaxe:	*[rótulo]* MOVWF **f**
Descrição:	O conteúdo do registro **W** é copiado para o registro **f**
Operação:	$\mathbf{W} \Rightarrow (\mathbf{f})$
Operando:	$0 \leq \mathbf{f} \leq 127$
Flag:	-
Número de palavras:	1
Número de ciclos:	1
Exemplo 1:	MOVWF OPTION_REG
	Antes da instrução: OPTION_REG = 0x20
	W = 0x40
	Depois da instrução: OPTION_REG = 0x40
	W = 0x40
Exemplo 2:	MOVWF INDF
	Antes da instrução: W = 0x17
	FSR = 0xC2
	Conteúdo do endereço 0xC2 = 0x00
	Depois da instrução: W = 0x17
	FSR = 0xC2
	Conteúdo do endereço 0xC2 = 0x17

MOVF Copiar f para d

Sintaxe:	*[rótulo]* MOVF **f, d**
Descrição:	O conteúdo do registo **f** é guardado no local determinado pelo operando **d**
	Se **d = 0**, o destino é o registro **W**
	Se **d = 1**, o destino é o próprio registro **f**
	A opção **d = 1**, é usada para testar o conteúdo do registro **f**, porque a execução desta instrução afeta a flag Z do registro STATUS.
Operação:	$\mathbf{f} \Rightarrow (\mathbf{d})$
Operando:	$0 \leq \mathbf{f} \leq 127, \mathrm{d} \in [0, 1]$
Flag:	Z
Número de palavras:	1
Número de ciclos:	1
Exemplo 1:	MOVF FSR, 0
	Antes da instrução: FSR = 0xC2
	W = 0x00
	Depois da instrução: W = 0xC2
	Z = 0
Exemplo 2:	MOVF INDF, 0
	Antes da instrução: W = 0x17
	FSR = 0xC2
	conteúdo do endereço 0xC2 = 0x00
	Depois da instrução: W = 0x00
	FSR = 0xC2
	conteúdo do endereço 0xC2 = 0x00
	Z = 1

CLRW Escrever 0 em W

Sintaxe:	*[rótulo]* CLRW
Descrição:	O conteúdo do registo **W** passa para 0 e a flag Z do registro STATUS toma o valor 1.
Operação:	$0 \Rightarrow (\mathbf{W})$
Operando:	-
Flag:	Z
Número de palavras:	1
Número de ciclos:	1
Exemplo:	CLRW
	Antes da instrução: W = 0x55
	Depois da instrução: W = 0x00
	Z = 1

CLRF Escrever 0 em f

Sintaxe:	*[rótulo]* CLRF **f**
Descrição:	O conteúdo do registro '**f**' passa para 0 e a flag Z do registro STATUS toma o valor 1.
Operação:	$0 \Rightarrow f$
Operando:	$0 \leq \mathbf{f} \leq 127$
Flag:	Z
Número de palavras:	1
Número de ciclos:	1
Exemplo 1:	CLRF STATUS
	Antes da instrução: STATUS = 0xC2
	Depois da instrução: STATUS = 0x00
	Z = 1
Exemplo 2:	CLRF INDF
	Antes da instrução: FSR = 0xC2
	conteúdo do endereço 0xC2 = 0x33
	Depois da instrução: FSR = 0xC2
	conteúdo do endereço 0xC2 = 0x00
	Z = 1

SWAPF — Copiar o conteúdo de f para d, trocando a posição dos 4 primeiros bits com a dos 4 últimos

Sintaxe:	*[rótulo]* SWAPF **f, d**
Descrição:	Os 4 bits + significativos e os 4 bits – significativos de f, trocam de posições.
	Se **d = 0**, o resultado é guardado no registro **W**
	Se **d = 1**, o resultado é guardado no registro **f**
Operação:	**f** <0:3>⇒ **d** <4:7>, **f** <4:7>⇒ **d** <0:3>,
Operando:	$0 \leq \mathbf{f} \leq 127, d \in [0, 1]$
Flag:	-
Número de palavras:	1
Número de ciclos:	1
Exemplo 1:	SWAPF REG, 0
	Antes da instrução: REG = 0xF3
	Depois da instrução: REG = 0xF3
	W = 0x3F
Exemplo 2:	SWAPF REG, 1
	Antes da instrução: REG = 0xF3
	Depois da instrução: REG = 0x3F

ADDLW — Adicionar W a uma constante

Sintaxe:	*[rótulo]* ADDLW **k**
Descrição:	O conteúdo do registro **W**, é adicionado à constante de 8-bits **k** e o resultado é guardado no registro **W**.
Operação:	$(\mathbf{W}) + \mathbf{k} \Rightarrow \mathbf{W}$
Operando:	$0 \leq \mathbf{k} \leq 255$
Flag:	**C, DC, Z**
Número de palavras:	1
Número de ciclos:	1
Exemplo 1:	ADDLW 0x15
	Antes da instrução: W= 0x10
	Depois da instrução: W= 0x25
Exemplo 2:	ADDLW REG
	Antes da instrução: W = 0x10
	REG = 0x37
	Depois da instrução: W = 0x47

ADDWF Adicionar W a f

Sintaxe:	*[rótulo]* ADDWF **f**, **d**
Descrição:	Adicionar os conteúdos dos registros **W** e **f**
	Se **d=0**, o resultado é guardado no registro **W**
	Se **d=1**, o resultado é guardado no registro **f**
Operação:	$(\mathbf{W}) + (\mathbf{f}) \Rightarrow \mathbf{d}, \mathbf{d} \in [0, 1]$
Operando:	$0 \leq \mathbf{f} \leq 127$
Flag:	**C, DC, Z**
Número de palavras:	1
Número de ciclos:	1
Exemplo 1:	ADDWF FSR, 0
	Antes da instrução: W = 0x17
	FSR = 0xC2
	Depois da instrução: W = 0xD9
	FSR = 0xC2
Exemplo 2:	ADDWF INDF,0
	Antes da instrução: W = 0x17
	FSR = 0xC2
	conteúdo do endereço 0xC2 = 0x20
	Depois da instrução: W = 0x37
	FSR = 0xC2
	Conteúdo do endereço 0xC2 = 0x20

SUBLW Subtrair W a uma constante

Sintaxe:	*[rótulo]* SUBLW **k**
Descrição:	O conteúdo do registro **W**, é subtraído à constante **k** e, o resultado, é guardado no registro **W**.
Operação:	$\mathbf{k} - (\mathbf{W}) \Rightarrow \mathbf{W}$
Operando:	$0 \leq \mathbf{k} \leq 255$
Flag:	**C, DC, Z**
Número de palavras:	1
Número de ciclos:	1
Exemplo 1:	SUBLW 0x03
	Antes da instrução: W= 0x01, C = x, Z = x
	Depois da instrução: W= 0x02, C = 1, Z = 0 Resultado > 0
	Antes da instrução: W= 0x03, C = x, Z = x
	Depois da instrução: W= 0x00, C = 1, Z = 1 Resultado = 0
	Antes da instrução: W= 0x04, C = x, Z = x
	Depois da instrução: W= 0xFF, C = 0, Z = 0 Resultado < 0
Exemplo 2:	SUBLW REG
	Antes da instrução: W = 0x10
	REG = 0x37
	Depois da instrução: W = 0x27
	C = 1 Resultado > 0

SUBWF Subtrair W a f

Sintaxe:	*[rótulo]* SUBWF **f**, **d**
Descrição:	O conteúdo do registro **W** é subtraído ao conteúdo do registro **f**
	Se **d=0**, o resultado é guardado no registro **W**
	Se **d=1**, o resultado é guardado no registro **f**
Operação:	$(\mathbf{f}) - (\mathbf{W}) \Rightarrow \mathbf{d}$
Operando:	$0 \leq \mathbf{f} \leq 127, \mathbf{d} \in [0, 1]$
Flag:	**C, DC, Z**
Número de palavras:	1
Número de ciclos:	1
Exemplo:	SUBWF REG, 1
	Antes da instrução: REG= 3, W= 2, C = x, Z = x
	Depois da instrução: REG= 1, W= 2, C = 1, Z = 0 Resultado > 0
	Antes da instrução: REG= 2, W= 2, C = x, Z = x
	Depois da instrução: REG=0, W= 2, C = 1, Z = 1 Resultado = 0
	Antes da instrução: REG=1, W= 2, C = x, Z = x
	Depois da instrução: REG= 0xFF, W=2, C = 0, Z = 0 Resultado < 0

ANDLW Fazer o “E” lógico de W com uma constante

Sintaxe:	*[rótulo]* ANDLW **k**
Descrição:	É executado o “E” lógico do conteúdo do registro **W**, com a constante **k**
	O resultado é guardado no registro **W**.
Operação:	$(\mathbf{W})$ .AND. $\mathbf{k} \Rightarrow \mathbf{W}$
Operando:	$0 \leq \mathbf{k} \leq 255$
Flag:	Z
Número de palavras:	1
Número de ciclos:	1
Exemplo 1:	ANDLW 0x5F
	Antes da instrução: W= 0xA3 ; 0101 1111 (0x5F)
	; 1010 0011 (0xA3)
	Depois da instrução: W= 0x03; 0000 0011 (0x03)
Exemplo 2:	ANDLW REG
	Antes da instrução: W = 0xA3 ; 1010 0011 (0xA3)
	REG = 0x37 ; 0011 0111 (0x37)
	Depois da instrução: W = 0x23 ; 0010 0011 (0x23)

ANDWF — Fazer o "E" lógico de W com f

Sintaxe:	*[rótulo]* ANDWF **f**, **d**
Descrição:	Faz o "E" lógico dos conteúdos dos registros **W** e **f**
	Se **d=0**, o resultado é guardado no registro **W**
	Se **d=1**, o resultado é guardado no registro **f**
Operação:	$(\mathbf{W})$ .AND. $(\mathbf{f}) \Rightarrow \mathbf{d}$
Operando:	$0 \leq \mathbf{f} \leq 127, \mathbf{d} \in [0, 1]$
Flag:	Z
Número de palavras:	1
Número de ciclos:	1
Exemplo 1:	ANDWF FSR, 1
	Antes da instrução: W= 0x17, FSR= 0xC2; 0001 1111 (0x17)
	; 1100 0010 (0xC2)
	Depois da instrução: W= 0x17, FSR= 0x02 ; 0000 0010 (0x02)
Exemplo 2:	ANDWF FSR, 0
	Antes da instrução: W= 0x17, FSR= 0xC2; 0001 1111 (0x17)
	; 1100 0010 (0xC2)
	Depois da instrução: W= 0x02, FSR= 0xC2; 0000 0010 (0x02)

IORLW — Fazer o "OU" lógico de W com uma constante

Sintaxe:	*[rótulo]* IORLW **k**
Descrição:	É executado o "OU" lógico do conteúdo do registro **W**, com a constante de 8 bits **k**, o resultado é guardado no registro **W**.
Operação:	$(\mathbf{W})$ .OR. $\mathbf{k} \Rightarrow \mathbf{W}$
Operando:	$0 \leq \mathbf{k} \leq 255$
Flag:	Z
Número de palavras:	1
Número de ciclos:	1
Exemplo 1:	IORLW 0x35
	Antes da instrução: W= 0x9A
	Depois da instrução: W= 0xBF
	Z= 0
Exemplo 2:	IORLW REG
	Antes da instrução: W = 0x9A
	conteúdo de REG = 0x37
	Depois da instrução: W = 0x9F
	Z = 0

IORWF — Fazer o "OU" lógico de W com f

Sintaxe:	*[rótulo]* IORWF **f**, **d**
Descrição:	Faz o "OU" lógico dos conteúdos dos registos **W** e **f**
	Se **d=0**, o resultado é guardado no registro **W**
	Se **d=1**, o resultado é guardado no registro **f**
Operação:	$(\mathbf{W})\ .\mathrm{OR}.\ (\mathbf{f}) \Rightarrow \mathbf{d}$
Operando:	$0 \le \mathbf{f} \le 127,\ \mathbf{d} \in [0, 1]$
Flag:	Z
Número de palavras:	1
Número de ciclos:	1
Exemplo 1:	IORWF REG, 0
	Antes da instrução: REG= 0x13, W= 0x91
	Depois da instrução: REG= 0x13, W= 0x93
	Z= 0
Exemplo 2:	IORWF REG, 1
	Antes da instrução: REG= 0x13, W= 0x91
	Depois da instrução: REG= 0x93, W= 0x91
	Z= 0

XORLW — "OU- EXCLUSIVO" de W com uma constante

Sintaxe:	*[rótulo]* XORLW **k**
Descrição:	É executada a operação "OU-Exclusivo" do conteúdo do registro **W**, com a constante **k.** O resultado é guardado no registro **W.**
Operação:	$(\mathbf{W})\ .\mathrm{XOR}.\ \mathbf{k} \Rightarrow \mathbf{W}$
Operando:	$0 \le \mathbf{k} \le 255$
Flag:	Z
Número de palavras:	1
Número de ciclos:	1
Exemplo 1:	XORLW 0xAF
	Antes da instrução: W= 0xB5 ; 1010 1111 (0xAF)
	; 1011 0101 (0xB5)
	Depois da instrução: W= 0x1A; 0001 1010 (0x1A)
Exemplo 2:	XORLW REG
	Antes da instrução: W = 0xAF ; 1010 1111 (0xAF)
	REG = 0x37 ; 0011 0111 (0x37)
	Depois da instrução: W = 0x98 ; 1001 1000 (0x98)
	Z = 0

XORWF “OU-EXCLUSIVO” de W com f

Sintaxe:	*[rótulo]* XORWF **f**, **d**
Descrição:	Faz o “OU-EXCLUSIVO” dos conteúdos dos registros **W** e **f**
	Se **d=0**, o resultado é guardado no registro **W**
	Se **d=1**, o resultado é guardado no registro **f**
Operação:	$(\mathbf{W})\ .\mathrm{XOR}.\ (\mathbf{f}) \Rightarrow \mathbf{d}$
Operando:	$0 \leq \mathbf{f} \leq 127, \mathbf{d} \in [0, 1]$
Flag:	Z
Número de palavras:	1
Número de ciclos:	1
Exemplo 1:	XORWF REG, 1
	Antes da instrução: REG= 0xAF, W= 0xB5 ; 1010 1111 (0xAF)
	; 1011 0101 (0xB5)
	Depois da instrução: REG= 0x1A, W= 0xB5 001 1010 (0x1A)
Exemplo 2:	XORWF REG, 0
	Antes da instrução: REG= 0xAF, W= 0xB5; 1010 1111 (0xAF)
	; 1011 0101 (0xB5)
	Depois da instrução: REG= 0xAF, W= 0x1A ; 0001 1010 (0x1A)

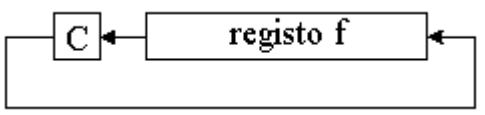
INCF Incrementar f

Sintaxe:	*[rótulo]* INCF **f**, **d**
Descrição:	Incrementar de uma unidade, o conteúdo do registro f.
	Se **d=0**, o resultado é guardado no registro **W**
	Se **d=1**, o resultado é guardado no registro **f**
Operação:	$(\mathbf{f}) + 1 \Rightarrow \mathbf{d}$
Operando:	$0 \leq \mathbf{f} \leq 127, \mathbf{d} \in [0, 1]$
Flag:	Z
Número de palavras:	1
Número de ciclos:	1
Exemplo 1:	INCF REG, 1
	Antes da instrução: REG = 0xFF
	Z = 0
	Depois da instrução: REG = 0x00
	Z = 1
Exemplo 2:	INCF REG, 0
	Antes da instrução: REG = 0x10
	W = x
	Z = 0
	Depois da instrução: REG = 0x10
	W = 0x11
	Z = 0

DECF Decrementar f

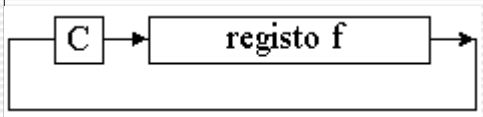
Sintaxe:	*[rótulo]* DECF **f**, **d**
Descrição:	Decrementar de uma unidade, o conteúdo do registro f.
	Se **d=0**, o resultado é guardado no registro **W**
	Se **d=1**, o resultado é guardado no registro **f**
Operação:	$(\mathbf{f}) - 1 \Rightarrow \mathbf{d}$
Operando:	$0 \leq \mathbf{f} \leq 127, \mathbf{d} \in [0, 1]$
Flag:	Z
Número de palavras:	1
Número de ciclos:	1
Exemplo 1:	DECF REG, 1
	Antes da instrução: REG = 0x01
	Z = 0
	Depois da instrução: REG = 0x00
	Z = 1
Exemplo 2:	DECF REG, 0
	Antes da instrução: REG = 0x13
	W = x
	Z = 0
	Depois da instrução: REG = 0x13
	W = 0x12
	Z = 0

RLF — Rodar f para a esquerda através do Carry

Sintaxe:	*[rótulo]* RLF **f**, **d**	
Descrição:	O conteúdo do registro **f** é rodado um espaço para a esquerda, através de C (flag do Carry).	
	Se **d=0**, o resultado é guardado no registro **W**	
	Se **d=1**, o resultado é guardado no registro **f**	
Operação:	(**f** <n>) ⇒ **d**<n+1>, f<7> ⇒ C, C ⇒ **d**<0>;	
Operando:	$0 \le \mathbf{f} \le 127$, $\mathbf{d} \in [0, 1]$	
Flag:	C	
Número de palavras:	1	
Número de ciclos:	1	
Exemplo 1:	RLF REG, 0	
	Antes da instrução: REG = 1110 0110	
	C = 0	
	Depois da instrução: REG = 1110 0110	
	W = 1100 1100	
	C = 1	
Exemplo 2:	RLF REG, 1	
	Antes da instrução: REG = 1110 0110	
	C = 0	
	Depois da instrução: REG = 1100 1100	
	C = 1	

RRF — Rodar f para a direita através do Carry

Sintaxe:	*[rótulo]* RRF **f**, **d**
Descrição:	O conteúdo do registro **f** é rodado um espaço para a direita, através de C (flag do Carry).
	Se **d=0**, o resultado é guardado no registro **W**
	Se **d=1**, o resultado é guardado no registro **f**
Operação:	(**f** <n>) ⇒ **d**<n-1>, f<0> ⇒ C, C ⇒ **d**<7>;
Operando:	$0 \le \mathbf{f} \le 127$, $\mathbf{d} \in [0, 1]$
Flag:	C
Número de palavras:	1
Número de ciclos:	1



Exemplo 1:	RRF REG, 0
	Antes da instrução: REG = 1110 0110
	W = x
	C = 0
	Depois da instrução: REG = 1110 0110
	W = 0111 0011
	C = 0
Exemplo 2:	RRF REG, 1
	Antes da instrução: REG = 1110 0110
	C = 0
	Depois da instrução: REG = 0111 0011
	C = 0

COMF — Complementar f

Sintaxe:	*[rótulo]* COMF **f**, **d**
Descrição:	O conteúdo do registro **f** é complementado.
	Se **d=0**, o resultado é guardado no registro **W**
	Se **d=1**, o resultado é guardado no registro **f**
Operação:	() ⇒ d
Operando:	$0 \le \mathbf{f} \le 127$, $\mathbf{d} \in [0, 1]$
Flag:	Z
Número de palavras:	1
Número de ciclos:	1
Exemplo 1:	COMF REG, 0
	Antes da instrução: REG= 0x13 ; 0001 0011 (0x13)
	Depois da instrução: REG= 0x13 ; complementar
	W = 0xEC ; 1110 1100 (0xEC)
Exemplo 2:	COMF INDF, 1
	Antes da instrução: FSR= 0xC2
	conteúdo de FSR = (FSR) = 0xAA
	Depois da instrução: FSR= 0xC2
	conteúdo de FSR = (FSR) = 0x55

BCF — Pôr a “0” o bit b de f

Sintaxe:	*[rótulo]* BCF **f**, **b**
Descrição:	Limpar (pôr a ‘0’), o bit **b** do registro **f**
Operação:	$0 \Rightarrow$ f<b>
Operando:	$0 \leq \mathbf{f} \leq 127, 0 \leq \mathbf{b} \leq 7$
Flag:	-
Número de palavras:	1
Número de ciclos:	1
Exemplo 1:	BCF REG, 7
	Antes da instrução: REG = 0xC7 ; 1100 0111 (0xC7)
	Depois da instrução: REG = 0x47 ; 0100 0111 (0x47)
Exemplo 2:	BCF INDF, 3
	Antes da instrução: W = 0x17
	FSR = 0xC2
	conteúdo do endereço em FSR (FSR) = 0x2F
	Depois da instrução: W = 0x17
	FSR = 0xC2
	conteúdo do endereço em FSR (FSR) = 0x27

BSF — Pôr a “1” o bit b de f

Sintaxe:	*[rótulo]* BSF **f**, **b**
Descrição:	Pôr a ‘1’, o bit **b** do registro **f**
Operação:	$1 \Rightarrow$ f<b>
Operando:	$0 \leq \mathbf{f} \leq 127, 0 \leq \mathbf{b} \leq 7$
Flag:	-
Número de palavras:	1
Número de ciclos:	1
Exemplo 1:	BSF REG, 7
	Antes da instrução: REG = 0x07 ; 0000 0111 (0x07)
	Depois da instrução: REG = 0x17 ; 1000 0111 (0x87)
Exemplo 2:	BSF INDF, 3
	Antes da instrução: W = 0x17
	FSR = 0xC2
	conteúdo do endereço em FSR (FSR) = 0x2F
	Depois da instrução: W = 0x17
	FSR = 0xC2
	conteúdo do endereço em FSR (FSR) = 0x28

BTFSC — Testar o bit b de f, saltar por cima se for = 0

Sintaxe:	*[rótulo]* BTFSC **f**, **b**
Descrição:	Se o bit **b** do registro **f** for igual a zero, ignorar instrução seguinte. Se este bit **b** for zero, então, durante a execução da instrução atual, a execução da instrução seguinte não se concretiza e é executada, em vez desta, uma instrução NOP, fazendo com que a instrução atual, demore dois ciclos de instrução a ser executada.
Operação:	Ignorar a instrução seguinte se (**f**<**b**>) = 0
Operando:	$0 \leq \mathbf{f} \leq 127, 0 \leq \mathbf{b} \leq 7$
Flag:	-
Número de palavras:	1
Número de ciclos:	1 ou 2 dependendo do valor lógico do bit **b**
Exemplo:	LAB_01 BTFSC REG, 1; Testar o bit 1 do registo REG
	LAB_02 ..... ; Ignorar esta linha se for 0
	LAB_03 ..... ; Executar esta linha depois da anterior, se for 1
	Antes da instrução, o contador de programa contém o endereço LAB_01. Depois desta instrução, se o bit 1 do registo REG for zero, o contador de programa contém o endereço LAB_03. Se o bit 1 do registo REG for 'um', o contador de programa contém o endereço LAB_02.

BTFSS — Testar o bit b de f, saltar por cima se for = 1

Sintaxe:	*[rótulo]* BTFSS **f**, **b**
Descrição:	Se o bit **b** do registro **f** for igual a um, ignorar instrução seguinte. Se durante a execução desta instrução este bit **b** for um, então, a execução da instrução seguinte não se concretiza e é executada, em vez desta, uma instrução NOP, assim, a instrução atual demora dois ciclos de instrução a ser executada.
Operação:	Ignorar a instrução seguinte se (**f**<**b**>) = 1
Operando:	$0 \leq \mathbf{f} \leq 127, 0 \leq \mathbf{b} \leq 7$
Flag:	-
Número de palavras:	1
Número de ciclos:	1 ou 2 dependendo do valor lógico do bit **b**
Exemplo:	LAB_01 BTFSS REG, 1; Testar o bit 1 do registro REG
	LAB_02 ..... ; Ignorar esta linha se for 1
	LAB_03 ..... ; Executar esta linha depois da anterior, se for 0
	Antes da instrução, o contador de programa contém o endereço LAB_01. Depois desta instrução, se o bit 1 do registro REG for 'um', o contador de programa contém o endereço LAB_03. Se o bit 1 do registro REG for zero, o contador de programa contém o endereço LAB_02.

INCFSZ Incrementar f, saltar por cima se der = 0

Sintaxe:	*[rótulo]* INCFSZ **f**, **d**
Descrição:	**Descrição:** O conteúdo do registro **f** é incrementado de uma unidade.
	Se **d = 0**, o resultado é guardado no registro **W**.
	Se **d = 1**, o resultado é guardado no registro **f**.
	Se o resultado do incremento for = 0, a instrução seguinte é substituída por uma instrução NOP, fazendo com que a instrução atual, demore dois ciclos de instrução a ser executada.
Operação:	$(\mathbf{f}) + 1 \Rightarrow \mathbf{d}$
Operando:	$0 \leq \mathbf{f} \leq 127, \mathbf{d} \in [0, 1]$
Flag:	-
Número de palavras:	1
Número de ciclos:	1 ou 2 dependendo do resultado
Exemplo:	LAB_01 INCFSZ REG, 1; Incrementar o conteúdo de REG de uma unidade
	LAB_02 ..... ; Ignorar esta linha se resultado = 0
	LAB_03 ..... ; Executar esta linha depois da anterior, se der 0
	Conteúdo do contador de programa antes da instrução, PC = endereço LAB_01. Se o conteúdo do registro REG depois de a operação REG = REG + 1 ter sido executada, for REG = 0, o contador de programa aponta para o rótulo de endereço LAB_03. Caso contrário, o contador de programa contém o endereço da instrução seguinte, ou seja, LAB_02.

DECFSZ — Decrementar f, saltar por cima se der = 0

Sintaxe:	*[rótulo]* DECFSZ **f**, **d**
Descrição:	O conteúdo do registro **f** é decrementado uma unidade.
	Se **d = 0**, o resultado é guardado no registro **W**.
	Se **d = 1**, o resultado é guardado no registro **f**.
	Se o resultado do decremento for = 0, a instrução seguinte é substituída por uma instrução NOP, fazendo assim com que a instrução atual, demore dois ciclos de instrução a ser executada.
Operação:	$(\mathbf{f}) - 1 \Rightarrow \mathbf{d}$
Operando:	$0 \leq \mathbf{f} \leq 127, \mathbf{d} \in [0, 1]$
Flag:	-
Número de palavras:	1
Número de ciclos:	1 ou 2 dependendo do resultado
Exemplo:	LAB_01 DECFSZ REG, 1; Decrementar o conteúdo de REG de uma unidade
	LAB_02 ..... ; Ignorar esta linha se resultado = 0
	LAB_03 ..... ; Executar esta linha depois da anterior, se der 0
	Conteúdo do contador de programa antes da instrução, PC = endereço LAB_01. Se o conteúdo do registro REG depois de a operação REG = REG – 1 ter sido executada, for REG = 0, o contador de programa aponta para o rótulo de endereço LAB_03. Caso contrário, o contador de programa contém o endereço da instrução seguinte, ou seja, LAB_02.

GOTO — Saltar para o endereço

Sintaxe:	*[rótulo]* GOTO **k**
Descrição:	Salto incondicional para o endereço **k**.
Operação:	$\mathbf{k} \Rightarrow PC<10:0>, (PCLATH<4:3>) \Rightarrow PC<12:11>$
Operando:	$0 \leq \mathbf{k} \leq 2048$
Flag:	-
Número de palavras:	1
Número de ciclos:	2
Exemplo:	LAB_00 GOTO LAB_01; Saltar para LAB_01
	:
	LAB_01 .....
	Antes da instrução: PC = endereço LAB_00
	Depois da instrução: PC = endereço LAB_01

CALL — Chamar um programa

Sintaxe:	*[rótulo]* CALL **k**
Descrição:	Esta instrução, chama um subprograma. Primeiro, o endereço de retorno (PC+1) é guardado na pilha, a seguir, o operando **k** de 11 bits, correspondente ao endereço de início do subprograma, vai para o contador de programa (PC).
Operação:	PC+1 ⇒ Topo da pilha (TOS – Top Of Stack)
Operando:	$0 \leq \mathbf{k} \leq 2048$
Flag:	-
Número de palavras:	1
Número de ciclos:	2
Exemplo:	LAB_00 CALL LAB_02 ; Chamar a subrotina LAB_02
	LAB_01 :
	:
	LAB_02 .....
	Antes da instrução: PC = endereço LAB_00
	TOS = x
	Depois da instrução: PC = endereço LAB_02
	TOS = LAB_01

RETURN — Retorno de um subprograma

Sintaxe:	*[rótulo]* RETURN
Descrição:	O conteúdo do topo da pilha é guardado no contador de programa.
Operação:	TOS ⇒ Contador de programa PC
Operando:	-
Flag:	-
Número de palavras:	1
Número de ciclos:	2
Exemplo:	RETURN
	Antes da instrução: PC = x
	TOS = x
	Depois da instrução: PC = TOS
	TOS = TOS - 1

RETLW — Retorno de um subprograma com uma constante em W

Sintaxe:	*[rótulo]* RETLW **k**
Descrição:	A constante **k** de 8 bits, é guardada no registro **W**.
Operação:	$(\mathbf{k}) \Rightarrow \mathbf{W}$; TOS $\Rightarrow$ PC
Operando:	$0 \leq k \leq 255$
Flag:	-
Número de palavras:	1
Número de ciclos:	2
Exemplo:	RETLW 0x43
	Antes da instrução: W = x
	PC = x
	TOS = x
	Depois da instrução: W = 0x43
	PC = TOS
	TOS = TOS – 1

RETFIE — Retorno de uma rotina de interrupção

Sintaxe:	*[rótulo]* RETLW **k**
Descrição:	Retorno de uma subrotina de atendimento de interrupção. O conteúdo do topo de pilha (TOS), é transferido para o contador de programa (PC). Ao mesmo tempo, as interrupções são habilitadas, pois o bit GIE de habilitação global das interrupções, é posto a '1'.
Operação:	TOS $\Rightarrow$ PC ; 1$\Rightarrow$ GIE
Operando:	-
Flag:	-
Número de palavras:	1
Número de ciclos:	2
Exemplo:	RETFIE
	Antes da instrução: PC = x
	GIE = 0
	Depois da instrução: PC = TOS
	GIE = 1

NOP Nenhuma operação

Sintaxe:	*[rótulo]* NOP
Descrição:	Nenhuma operação é executada, nem qualquer flag é afectada.
Operação:	-
Operando:	-
Flag:	-
Número de palavras:	1
Número de ciclos:	1
Exemplo:	NOP

CLRWDT Iniciar o temporizador do watchdog

Sintaxe:	*[rótulo]* CLRWDT
Descrição:	O temporizador do watchdog é reposto a zero. O prescaler do temporizador de Watchdog é também reposto a 0 e, também, os bits do registro de estado e são postos a 'um'.
Operação:	0 ⇒ WDT
	0 ⇒ prescaler de WDT
	1⇒
	1⇒
Operando:	-
Flag:	
Número de palavras:	1
Número de ciclos:	1
Exemplo:	CLRWDT
	Antes da instrução: Contador de WDT = x
	Prescaler de WDT = 1:128
	Depois da instrução: Contador do WDT = 0x00
	Prescale do WDT = 0

SLEEP Modo de repouso

Sintaxe:	*[rótulo]* SLEEP
Descrição:	O processador entra no modo de baixo consumo. O oscilador pára. O bit (Power Down) do registro Status é reposto a '0'. O bit (Timer Out) é posto a '1'. O temporizador de WDT (Watchdog) e o respectivo prescaler são repostos a '0'.
Operação:	0 ⇒ WDT
	0 ⇒ prescaler do WDT
	1 ⇒ $\overline{TO}$
	0 ⇒ $\overline{PD}$
Operando:	-
Flag:	
Número de palavras:	1
Número de ciclos:	1
Exemplo 1:	SLEEP
	Antes da instrução: Contador do WDT = x
	Prescaler do WDT = x
	Depois da instrução: Contador do WDT = 0x00
	Prescaler do WDT = 0

Condiçõe dos registradores após reset

Endereço	Nome	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Power-On Reset	Outros Resets (3)
Banco 1											
00h	INDF	Utiliza contéudos do FSR para endereçar a memória de dados (não é um registrador físico)								---- ----	---- ----
01h	TMR0	Contador / Temporizador de 8 bits								xxxx xxxx	uuuu uuuu
02h	PCL	8 bits menos significativos do contador de programa								0000 0000	0000 0000
03h	STATUS (2)	IRP	RP1	RP0	/TO	/PD	Z	DC	C	0001 1xxx	000q quuu
04h	FSR	Ponteiro para endereçamento indireto de programa								xxxx xxxx	uuuu uuuu
05h	PORTA	-	-	-	RA4/TOCKI	RA3	RA2	RA1	RA0	---x xxxxx	---u uuuu
06h	PORTB	RB7	RB6	RB5	RB4	RB3	RB2	RB1	RB0	xxxx xxxx	uuuu uuuu
07h		Localização não implementada, ler com '0'								---- ----	---- ----
08h	EEDATA	Registrador de dado da EEPROM								xxxx xxxx	uuuu uuuu
09h	EEADR	Registrador de endereço da EEPROM								xxxx xxxx	uuuu uuuu
0Ah	PCLATH	-	-	-	5 bits mais significativos do contador de programa (1)					---0 0000	---0 0000
0Bh	INTCON	GIE	EEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF	0000 000x	0000 000x
Banco 2											
80h	INDF	Utiliza contéudos do FSR para endereçar a memória de dados (não é um registrador físico)								---- ----	---- ----
81h	OPTION	/RBPU	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0	1111 1111	1111 1111
82h	PCL	8 bits menos significativos do contador de programa								0000 0000	0000 0000
83h	STATUS (2)	IRP	RP1	RP0	/TO	/PD	Z	DC	C	0001 1xxx	000q quuu
84h	FSR	Ponteiro para endereçamento indireto de programa								xxxx xxxx	uuuu uuuu
85h	TRISA	-	-	-	Ref. RA4	Ref. RA3	Ref. RA2	Ref. RA1	Ref. RA0	---1 1111	---1 1111
86h	TRISB	Ref. RB7	Ref. RB6	Ref. RB5	Ref. RB4	Ref. RB3	Ref. RB2	Ref. RB1	Ref. RB0	1111 1111	1111 1111
87h		Localização não implementada, ler com '0'								---- ----	---- ----
88h	EECON1	-	-	-	EEIF	WRERR	WREN	WR	RD	---0 x000	---0 q000
89h	EECON2	Registrador de controle da EEPROM (não é um registrador físico)									
8Ah	PCLATH	-	-	-	5 bits mais significativos do contador de programa (1)					---0 0000	---0 0000
8Bh	INTCON	GIE	EEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF	0000 000x	0000 000x

Legenda: x = desconhecido, u = inalterado, - não implementado (ler como '0'), q = valor depende da condição

Notas:
1) O Byte superior do contador de programa não esta diretamente acessível. O PCLATH é um registrador – escravo para PC<12:8>. Os conteúdos do PALTH podem ser transferidos para o byte superior do contador de programa mas os conteúdos do PC<1:8> nunca são transferidos para o PCLATH
2) Os bits status /TO e /PD no registrador STATUS não são afetados por um reset no pino /MCLR.
3) Outros resets (não power-up) incluem: o reset externo através /MCLR e o reset do WatchDog Timer.

APÊNDICE C

DIRETRIZES DA LINGUAGEM MPASM

__BADRAM - CONFIGURA REGIÕES DA RAM NÃO DISPONÍVEIS

Sintaxe

```
__BADRAM      <expr>[-<expr>[,<expr>-[<expr>]]]
```

Descrição

Determina blocos de memória RAM que não podem ser utilizados pelo microcontrolador. Os valores de `<expr>` devem estar sempre dentro do limite imposto pela diretriz `__MAXRAM`. Caso o programa tente utilizar um endereço englobado por esses limites, uma mensagem de erro será gerada. O programador também não precisa se preocupar com essa diretriz, pois ela está definida nos arquivos de INCLUDE fornecidos pela Microchip.

Exemplo

```
Veja exemplo de __MAXRAM
```

Veja também

__MAXRAM

BANKISEL - GERA CÓDIGO PARA ACERTAR BANCO DE MEMÓRIA (ACESSO INDIRETO)

Sintaxe

```
BANKISEL      <nome>
```

Descrição

Essa diretriz é utilizada para acertar automaticamente o banco de memória para acesso indireto. Na verdade, o compilador irá gerar o código necessário para acertar o banco. Para PICs de 14 bits, serão utilizadas instruções para setar ou limpar o bit IRP do registrador de STATUS. Já para os PICs de 16 bits, as instruções MOVLB e

MOVLR serão implementadas. O argumento <nome> representa a variável com a qual se trabalhará e que deve servir de referência para a seleção do banco.

Exemplo

```
MOVLW  VAR1
MOVWF  FSR
BANKISEL       VAR1
...
MOVWF  INDF
```

Veja também

PAGESEL, BANKSEL

BANKSEL - GERA CÓDIGO PARA ACERTAR BANCO DE MEMÓRIA (ACESSO DIRETO)

Sintaxe

```
BANKSEL        <nome>
```

Descrição

Esta diretriz é utilizada para acertar automaticamente o banco de memória para acesso direto. Na verdade, o compilador irá gerar o código necessário para acertar o banco. Para PICs de 12 bits, as instruções para acertar os bits do FSR serão inseridas no código. Para PICs de 14 bits, serão utilizadas instruções para setar ou limpar os bits RP0 e RP1 do registrador de STATUS. Já para os PICs de 16 bits, as instruções MOVLB e MOVLR serão implementadas. Entretanto, se o PIC em uso contém somente um banco de RAM, nenhuma instrução adcional é gerada. O argumento <nome> representa a variável com a qual se trabalhará e que deve servir de referência para a seleção do banco.

Exemplo

```
MOVLW  .10
BANKSEL        VAR1
MOVWF  VAR1
```

Veja também

PAGESEL, BANKISEL

CBLOCK - DEFINE UM BLOCO DE CONSTANTES

Sintaxe

```
CBLOCK [<expr>]
       <nome> [:<incremento>] [,<nome> [:<incremento>]]
ENDC
```

Descrição

Define uma série de constantes. O primeiro <nome> é associado ao valor de <expr>. Já o segundo é associado ao valor seguinte e assim sucessivamente. Caso não seja determinado o valor de <incremento>, a próxima constante receberá o valor imediatamente seguinte, isto é, incremento unitário. Caso contrário, a próxima constante receberá o valor da constante anterior mais o <incremento>. A definição de constantes deve ser terminada pela diretriz ENDC.

Exemplo

```
CBLOCk 0X20
            W_TEMP              ;W_TEMP=0X20
            STATUS_TEMP         ;STATUS_TEMP=0X21
            TEMP1,TEMP2         ;TEMP1=0X22, TEMP2=0X23
            END:0,END_H,END_L   ;END=0X24, END_H=OX24, END_L=0X25
            CODIGO:2            ;CODIGO=0X26
            CONTA               ;CONTA=0X28
ENDC
```

Veja também

ENDC

CODE - DECLARA O INÍCIO DE UM BLOCO DE PROGRAMA

Sintaxe

```
[<nomel>]    CODE   [<ROM endereço>]
```

Descrição

Usado para objetos. Serve para declarar o início de um bloco de programa. Caso o argumento <nome> não seja definido, o bloco será chamado CODE. O endereço inicial é passado pelo argumento <ROM endereço>. Se este não for especificado, o compilador assume o endereço zero.

Exemplo

```
RESET  CODE         H'01FF'
            GOTO          START
```

Veja também

IDATA, UDATA, UDATA_OVR, UDATA_SHR, EXTERN, GLOBAL

__CONFIG - CONFIGURA OS DADOS PARA GRAVAÇÃO DO MICROCONTROLADOR

Sintaxe

```
__CONFIG      <expr>
```

Descrição

Utilizado para configurar previamente os dados para a gravação do PIC. Para facilitar a vida do programador, a Microchip já definiu uma série de símbolos para essas opções nos arquivos de INCLUDE de cada modelo de PIC. Basta dar uma conferida nesses arquivos para saber quais os símbolos pertinentes ao tipo de oscilador, WDT, etc. A <expr> deve ser montada pela junção destes símbolos por meio do operador &.

Exemplo

```
__CONFIG      _WDT_ON & _XT_OSC & _CP_OFF
```

Veja também

LIST, PROCESSOR, __IDLOCS

CONSTANT - DEFINE UMA CONSTANTE

Sintaxe

```
CONSTANT      <nome> = <expr> [,<nome> = <expr>]
```

Descrição

Define uma constante para ser utilizada nas expressões que serão interpretadas pelo compilador. Como o próprio nome diz, o valor determinado por <expr> é associado de maneira definitiva ao <nome>, não podendo mais ser alterado no decorrer do programa.

Exemplo

```
CONSTANT      TAM_NOME = 10 , TAM_CODIGO = 5
.
.
.
.
CONSTANT      TAM_TOTAL = TAM_NOME + TAM_CODIGO
```

Veja também

SET, VARIABLE

DATA - PREENCHE A MEMÓRIA COM NÚMEROS OU TEXTOS

Sintaxe

```
DATA   <expr>[,<expr>,...,<expr>]
DATA   "texto"[,"texto",...,"texto"]
```

Descrição

Preenche a memória com o valor determinado por `<expr>` ou `"texto"`, começando na posição atual e avançando a quantidade de posições necessárias para caber o dado. As expressões ou caracteres unitários são colocados um em cada posição de memória. Já os textos ASCII são arquivados com dois caracteres para cada posição da Memória, sendo o primeiro no byte mais significativo. Se o texto possui um número ímpar de caracteres, um zero é escrito no último byte.

Exemplo

```
DATA   1,2,CONTA
DATA   'N'
DATA   "TEXTO DE EXEMPLO"
```

Veja também

DW, DB, DE, DT

DB - PREENCHE A MEMÓRIA BYTE A BYTE

Sintaxe

```
DB     <expr>[,<expr>,...,<expr>]
```

Descrição

Preenche a memória com o valor determinado por `<expr>`, começando na posição atual e guardando os dados a cada byte, avançando a quantidade de posições necessárias para caber todas as `<expr>`.

Exemplo

```
DB          .1,'T',0x0f,CONTA,'s'
```

Veja também

DATA, DW, DE, DT

DE - PREENCHE A MEMÓRIA EEPROM BYTE A BYTE

Sintaxe

```
DE      <expr>[,<expr>,...,<expr>]
DE      "texto"[, "texto",..., "texto"]
```

Descrição

Preenche a memória EEPROM com o valor determinado por `<expr>` ou `"texto"`, começando na posição atual e guardando os dados a cada byte, avançando a quantidade de posições necessárias para caber todas as `<expr>`. No caso do `"texto"`, cada caractere será gravado em um byte. Caso essa diretriz seja utilizada para gravar dados na área de programação, os bits mais significativos (acima do 7) serão zerados.

Exemplo

```
ORG             H'2100'                 ;aponta para o início da EEPROM
DE              "Exemplo, V1.0"
```

Veja também

DATA, DW, DB, DT

#DEFINE - DEFINE UMA SUBSTITUIÇÃO DE TEXTO

Sintaxe

```
#DEFINE         <nome> [<texto>]
```

Descrição

Essa diretriz define uma substituição de texto. Sempre que o compilador encontrar `<nome>`, ele será substituído pelo `<texto>` associado a ele. Ao contrário da diretriz EQU, que só pode associar valores, essa diretriz pode associar qualquer coisa ao nome, inclusive um comando ou o endereço de um bit. Símbolos criados desta maneira não podem ser monitorados pelas ferramentas do MPLab (simulador). Definindo um nome sem um `<texto>` associado, ele poderá ser usado com a diretriz IFDEF.

Exemplo

```
#DEFINE         LED             PORTA,0
#DEFINE         LED_ON          BSF             LED
#DEFINE         LED_OFF         BCF             LED
```

Veja também

IFDEF, IFNDEF, #UNDEFINE

DT - PREENCHE A MEMÓRIA COM UMA TABELA

Sintaxe

```
DT     <expr>[,<expr>,...,<expr>]
DT   "texto"
```

Descrição

Preenche a memória com uma série de instruções RETLW, uma para cada `<expr>` ou caractere do `"texto"`, começando na posição atual e avançando a quantidade de posições necessárias para caber todas as instruções.

Exemplo

```
DT            "MOSAICO"
DT            VALOR1,VALOR2,VALOR3
```

Veja também

DATA, DE, DB, DW

DW - PREENCHE A MEMÓRIA PALAVRA A PALAVRA

Sintaxe

```
DW     <expr>[,"texto",...,<expr>]
```

Descrição

Preenche a memória com o valor determinado por `<expr>` ou `"texto"`, começando na posição atual e avançando a quantidade de posições necessárias para caber todos os dados. As expressões ou caracteres unitários são colocados um em cada posição de memória. Já os textos ASCII são arquivados com dois caracteres para cada posição, sendo o primeiro no byte mais significativo. Se o texto possui um número ímpar de caracteres, um zero é escrito no último byte.

Exemplo

```
DW            39,"MOSAICO"
```

Veja também

DATA, DE, DB, DT

ELSE - BLOCO ALTERNATIVO PARA TESTE CONDICIONAL

Sintaxe

```
ELSE
```

Descrição

Usado em conjunto com as diretrizes IF, IFDEF e IFNDEF para executar um bloco específico no caso do teste condicional apresentar um resultado negativo.

Exemplo

Veja o exemplo de IF.

Veja também

IF, IFDEF, IFNDEF, ENDIF

END - FIM DO PROGRAMA

Sintaxe

```
END
```

Descrição

Usado para determinar o fim do programa. Essa diretriz é obrigatória na última linha do código-fonte.

Exemplo

```
INICIO
        .
        (corpo do programa)
        .
END
```

ENDC - FINALIZA UM BLOCO DE CONSTANTES

Sintaxe

```
ENDC
```

Descrição

Finaliza o bloco de constantes iniciado por CBLOCK.

Exemplo

```
Veja o exemplo de CBLOCK.
```

Veja também

CBLOCK

ENDIF - FIM DOS BLOCOS DE TESTE CONDICIONAL

Sintaxe

```
ENDIF
```

Descrição

Usado em conjunto com as diretrizes IF, IFDEF, IFNDEF e ELSE para finalizar os blocos de códigos que serão executados de acordo com os testes condicionais.

Exemplo

```
Veja o exemplo de IF.
```

Veja também

IF, IFDEF, IFNDEF, ELSE

ENDM - FINALIZA O BLOCO DE UMA MACRO

Sintaxe

```
ENDM
```

Descrição

Finaliza o bloco de uma macro.

Exemplo

```
Veja o exemplo de MACRO.
```

Veja também

EXITM, MACRO

ENDW - FIM DO BLOCO DE LOOP

Sintaxe

```
ENDW
```

Descrição

Usado em conjunto com a diretriz WHILE para finalizar o bloco de códigos que será repetido enquanto o teste de WHILE for verdadeiro.

Exemplo

```
Veja o exemplo de WHILE.
```

Veja também

WHILE

EQU - DEFINE UMA SUBSTITUIÇÃO

Sintaxe

```
<nome> EQU <expr>
```

Descrição

O `<nome>` será substituído pelo valor determinado por `<expr>` sempre que for encontrado. Este conceito é totalmente similar ao de constantes.

Exemplo

```
TEMPO  EQU    .4
```

Veja também

SET, CONSTANT

ERROR - GERA UMA MENSAGEM DE ERRO

Sintaxe

```
ERROR  "<texto>"
```

Descrição

Gera uma mensagem de erro, que será mostrada como um erro no relatório gerado durante a compilação. O texto pode ter até 80 caracteres.

Exemplo

```
IFNDEF __PIC16F84
            ERROR "ATENCÃO: o PIC escolhido não é o 16F84"
ENDIF
```

Veja também

MESSG

ERRORLEVEL - DETERMINA O NÍVEL DE GERAÇÃO DAS MENSAGENS

Sintaxe

```
ERRORLEVEL   0/1/2/+/- <msg_num>
```

Descrição

Determina que tipo de mensagens deve ser mostrado durante a compilação:

Nível	O que será mostrado
0	Mensagens, alertas e erros (tudo).
1	Alertas e erros.
2	Somente erros.
+<msg_num>	Habilita a mensagem de número <meg_num>.
-<msg_num>	Inibe a mensagem de número <meg_num>.

☞ *Dica: Os números das mensagens, alertas e erros podem ser encontrados no Apêndice C do manual do compilador MPASM. Nas versões mais novas do MPLab, essa diretriz pode ter seu nível acertado e gravado no arquivo de projeto.*

Exemplo

```
ERRORLEVEL   1, -202
```

Veja também

LIST

EXITM - FORÇA A SAÍDA DE UMA MACRO

Sintaxe

```
EXITM
```

Descrição

Força a saída de uma macro. Seus efeitos são similares aos da diretriz ENDM.

Exemplo

```
Veja o exemplo de MACRO.
```

Veja também

ENDM, MACRO

EXPAND - EXPANDE O CÓDIGO DAS MACROS

Sintaxe

```
EXPAND
```

Descrição

Determina que todas as macros encontradas após essa diretriz serão expandidas na listagem do programa (Absolut List - Arquivo .LST).

Veja também

LIST, NOEXPAND, MACRO

EXTERN - DECLARA SÍMBOLOS DEFINIDOS EXTERNAMENTE

Sintaxe

```
EXTERN <nome> [, <nome>]
```

Descrição

Usado para objetos. Declara símbolos que serão definidos no código corrente, mas que são definidos como GLOBAL em outros módulos. Essa diretriz deve ser aplicada antes dos símbolos serem utilizados no código corrente. Mais de um símbolo pode ser definido na mesma linha, por meio dos argumentos `<nome>`.

Exemplo

```
EXTERN TESTE
...
CALL   TESTE
```

Veja também

TEXT, IDATA, UDATA, UDATA_OVR, UDATA_SHR, GLOBAL

FILL - PREENCHE A MEMÓRIA COM UM VALOR OU INSTRUÇÃO

Sintaxe

```
FILL   <expr>,<num>
```

Descrição

Preenche a memória com o valor determinado por `<expr>`, começando na posição atual e avançando `<num>` posições. A `<expr>` pode também ser uma instrução do Assembler, mas neste caso deve ser colocada entre parênteses.

Exemplo

```
FILL   0X1009,5        ;Preenche 5 posições com o valor 0x1009
FILL   (NOP),CONTA     ;Preenche CONTA posições com a instr. NOP
```

Veja também

ORG, DATA, DW

GLOBAL - DECLARA SÍMBOLOS QUE PODEM SER USADOS EXTERNAMENTE

Sintaxe

```
GLOBAL <nome>   [, <nome>]
```

Descrição

Usado com objetos. Declara os símbolos que serão definidos no módulo corrente, mas que poderão ser exportados para outros módulos. Essa diretriz pode ser utilizada antes ou após a definição do símbolo. Mais de um símbolo pode ser declarado na mesma linha por meio dos argumentos <nome>.

Exemplo

```
        UDATA
        VAR1    RES     .1
        VAR2    RES     .1
        GLOBAL VAR1, VAR2

CODE
SOMATRES
        GLOBAL SOMATRES
        ADDLW  .3
        RETURN
```

Veja também

TEXT, IDATA, UDATA, UDATA_OVR, UDATA_SHR, EXTERN

IDATA - DECLARA UMA SEÇÃO DE DADOS JÁ INICIALIZADOS

Sintaxe

```
[<nome>]        IDATA  [<RAM endereço>]
```

Descrição

Usado com objetos. Declara o início de uma seção de dados inicializados. Se o argumento <nome> não for especificado, a seção será chamada de IDATA. O endereço inicial para os dados é definido por <RAM endereço>, ou será adotado o valor zero caso esse argumento não seja especificado. Nenhum código será gerado nesta seção. O linker irá gerar uma tabela de entrada para cada byte especificado. O usuário deverá então definir a inicialização apropriada. Essa diretriz não está disponível para PICs de 12 bits.

Exemplo

```
IDATA
LIMITEL         DW      .0
LIMITEH         DW      .300
GANHO  DW       .5
FLAGS  DW       .0
TEXTO  DB       'MOSAICO'
```

Veja também

TEXT, UDATA, UDATA_OVR, UDATA_SHR, EXTERN, GLOBAL

__IDLOCS - ESPECIFICA OS DADOS PARA GRAVAÇÃO DO ID

Sintaxe

```
__IDLOCS      <expr>
```

Descrição

Utilizado para especificar previamente os dados que serão gravados nas quatro posições de ID.

Exemplo

```
__IDLOCS      H'1234'
```

Veja também

LIST, PROCESSOR, __CONFIG

IF - TESTE CONDICIONAL DE UMA EXPRESSÃO

Sintaxe

```
IF      <expr>
```

Descrição

Testa se a expressão definida por <expr> é verdadeira ou falsa. Caso ela seja verdadeira, então o código existente entre essa diretriz e a ENDIF será executado. Caso contrário, ele será pulado, podendo ainda ser executado o bloco relacionado à diretriz ELSE, caso ela exista.

Exemplo

```
CONTA = 5
.
.
IF            CONTA > 3
              .
              (rotina específica para CONTA > 3)
              .
ELSE
              .
              (rotina específica para CONTA ≤ 3)
              .
ENDIF
```

Veja também

ENDIF, ELSE

IFDEF - TESTE CONDICIONAL DE EXISTÊNCIA DE UM SÍMBOLO

Sintaxe

```
IFDEF <nome>
```

Descrição

Testa se o símbolo especificado por <nome> já foi definido (diretriz #DEFINE, sem o argumento <texto>). Caso o símbolo já tenha sido definido, então o código existente entre essa diretriz e a ENDIF será executado. Caso contrário, ele será pulado, podendo ser executado o bloco associado à diretriz ELSE, caso ela exista.

Exemplo

```
#DEFINE      TESTE
...
IFDEF  TESTE
            .
            (rotina que será executada para teste)
            .
ENDIF
```

Veja também

IFNDEF, #DEFINE, #UNDEFINE, ENDIF, ELSE

IFNDEF - TESTE CONDICIONAL DE NÃO-EXISTÊNCIA DE UM SÍMBOLO

Sintaxe

```
IFNDEF <nome>
```

Descrição

Testa se o símbolo especificado por <nome> não foi definido. Caso o símbolo não tenha sido definido, então o código existente entre essa diretriz e a ENDIF será executado. Caso contrário, ele será pulado, podendo ser executado o bloco associado à diretriz ELSE, caso ela exista.

Exemplo

```
#UNDEFINE    TESTE                ;garante a não-definição de TESTE
...
IFNDEF TESTE
            .
            (rotina que será executada fora do teste)
            .
ENDIF
```

Veja também

IFNDEF, #DEFINE, #UNDEFINE, ENDIF, ELSE

#INCLUDE - INCLUI NO PROGRAMA UM ARQUIVO DE DEFINIÇÕES

Sintaxe

```
#INCLUDE      <nome_do_arquivo>
#INCLUDE      "nome_do_arquivo"
```

Descrição

Usado geralmente no início do programa, para incluir definições especificadas em arquivos auxiliares. Esses arquivos, com mesma formatação dos códigos-fonte, possuem, no entanto, somente definições, variáveis, constantes e similares, para facilitar a vida do programador. Junto com o MPLab, a Microchip fornece vários desses arquivos com a extensão .INC, um para cada tipo de PIC. Se o path for especificado em `nome_do_arquivo`, então o arquivo será procurado somente nesse diretório. Caso contrário, a ordem de procura será a seguinte: diretório atual, diretório dos arquivos de código-fonte e diretório do MPASM.

Exemplo

```
#INCLUDE      "C:\MOSAICO\MOSAICO.INC"
#INCLUDE      <PIC16F84.INC>
```

LIST - OPÇÕES PARA A LISTAGEM DO PROGRAMA

Sintaxe

```
LIST   [<opção>, <opção>]
```

Descrição

Configura uma série de opções que será utilizada para a criação da listagem de programa (arquivo com extensão .LST). As opções possíveis encontram-se ma seguinte tabela:

Opção	Padrão	Descrição
b=nnn	8	Espaços para tabulações.
c=nnn	132	Quantidade de colunas.
f=<formato>	INHX8M	Formato do arquivo de saída. Pode ser INHX32, INHX8M ou INHX8S.
free	fixed	Utiliza formato livre.
fixed	fixed	Utiliza formato fixo.
mm=ON/OFF	On	Mostra o mapa da memória no arquivo de listagem.
n=nnn	60	Número de linhas por página.

MACRO - DEFINE UMA MACRO

Sintaxe

```
<nome> MACRO  [<arg1>, ..., <argx>]
```

Descrição

Uma macro é uma seqüência de instruções que pode ser inserida no seu código com uma simples referência ao nome dela. Por isso, normalmente as macros são utilizadas para economizar digitação de funções muito utilizadas e podem ser definidas em arquivos do tipo INCLUDE. Uma macro pode chamar outra macro de dentro dela. Os argumentos passados à macro serão substituídos no corpo do código. Para terminar uma macro, é necessária a diretriz ENDM. No entanto, ela pode ser finalizada quando é encontrada a diretriz EXITM. Variáveis utilizadas somente dentro das macros podem ser definidas como LOCAL. Maiores informações sobre este poderoso recurso podem ser obtidas no Capítulo 4 do manual do MPASM.

Exemplo

```
;(Definição)
TESTE  MACRO  REG
              IF      REG == 1
                      EXITM
              ELSE
                      ERRO   "REGISTRADOR ERRADO
              ENDIF
              ENDM
;(Utilização)
TESTE  TEMP
```

Veja também

ENDM, EXITM, LOCAL, IF, ELSE, ENDIF, WHILE, ENDW

__MAXRAM - CONFIGURA O TAMANHO MÁXIMO DA RAM

Sintaxe

```
__MAXRAM      <expr>
```

Descrição

Utilizado para configurar o tamanho máximo da memória para o PIC que está sendo utilizado. A <expr> determina o último endereço de memória disponível. O programador não precisa se preocupar com essa diretriz, pois ela está definida nos arquivos de INCLUDE fornecidos pela Microchip.

Exemplo

```
;(para PIC 16F84)
__MAXRAM H'CF'
__BADRAM H'07', H'50'-H'7F', H'87'
```

Veja também

__BADRAM

MESSG - GERA UMA MENSAGEM DEFINIDA PELO USUÁRIO

Sintaxe

```
MESSG  "texto"
```

Descrição

Gera uma mensagem que será mostrada no relatório gerado durante a compilação. O texto pode ter até 80 caracteres.

Exemplo

```
IFNDEF __PIC16F84
            MESSG "ATENÇÃO: o PIC escolhido não é o 16F84"
ENDIF
```

Veja também

ERROR

NOEXPAND - NÃO EXPANDE O CÓDIGO DAS MACROS

Sintaxe

```
NOEXPAND
```

Descrição

Determina que todas as macros encontradas após essa diretriz não serão expandidas na listagem do programa (.LST).

Veja também

LIST, EXPAND, MACRO

NOLIST - DESLIGA A GERAÇÃO DO ARQUIVO DE LISTAGEM

Sintaxe

```
NOLIST
```

Descrição

Inibe a geração automática do arquivo de listagem (Absolut List - Arquivo .LST).

Veja também

LIST

ORG - ACERTA PONTO DA MEMÓRIA DE PROGRAMAÇÃO

Sintaxe

```
[<nome>]      ORG          <expr>
```

Descrição

Usado para determinar o ponto da memória de programação em que a próxima instrução será escrita. Se nenhuma diretriz ORG for colocada no programa, então ele começará a ser escrito na posição 0x00. Caso `<nome>` seja especificado, então o valor de `<expr>` será associado a ele.

Exemplo

```
END_1  ORG    0x10
END_2  ORG    END_1 + 0x10
```

PAGE - INSERE UMA QUEBRA DE PÁGINA

Sintaxe

```
PAGE
```

Descrição

Insere uma quebra de página na geração do arquivo de listagem do programa (.LST).

Veja também

LIST, TITLE, SUBTITLE, SPACE

PAGESEL - SELECIONA A PÁGINA DE PROGRAMAÇÃO

Sintaxe

```
PAGESEL       <nome>
```

Descrição

Essa diretriz é utilizada para acertar automaticamente a página de memória de programação. Na verdade, o compilador irá gerar o código necessário para acertar a página. Para PICs de 12 bits, as instruções para acertar os bits do STATUS serão inseridas no código. Para PICs de 14 e 16 bits, serão utilizadas as instruções MOVLW e MOVWF para acertar o valor de PCLATH. Entretanto, se o PIC em uso contém somente uma página de programação, nenhuma instrução adicional é gerada. O argumento `<nome>` representa a rotina com a qual se trabalhará, e deve ser definida antes do uso dessa diretriz.

Exemplo

```
PAGESEL      GOTODEST
GOTO   GOTODEST
```

Veja também

BANKSEL, BANKISEL

PROCESSOR - DETERMINA O TIPO DE PIC UTILIZADO

Sintaxe

```
PROCESSOR          <tipo>
```

Descrição

Determina o tipo de PIC que será utilizado. Nas versões mais novas do MPLab, essa diretriz não é mais necessária, pois este valor é acertado e gravado no arquivo de projeto.

Exemplo

```
PROCESSOR          PIC16F84
```

Veja também

LIST

RADIX - DETERMINA O RADIX PADRÃO

Sintaxe

```
RADIX <tipo>
```

Descrição

Determina o tipo padrão a ser considerado a todos os números encontrados no programa, quando não devidamente especificados. O valor de `<tipo>` pode ser: HEX, DEC ou OCT. Lembre-se de que, independentemente do RADIX padrão, qualquer número pode ser acertado por meio das predefinições:

Radix	Formato 1	Formato 2
Decimal	D'xx'	.x
Hexadecimal	H'xx'	0Xxx
Octadecimal	O'xx'	
Binário	B'xxxxxxxx'	
ASCII	A'x'	'x'

☞ *Dica: Recomendamos que as predefinições apresentadas acima sejam sempre utilizadas, para evitar problemas caso alguém altere o valor do radix em seus programas.*

Nas versões mais novas do MPLab, essa diretriz não é mais necessária, pois este valor é acertado e gravado no arquivo de projeto.

Exemplo

```
RADIX       DEC
```

Veja também

LIST

RES - RESERVA MEMÓRIA

Sintaxe

```
[<nome>]    RES    <tamanho>
```

Descrição

Reserva memória de programação para uso posterior. O bloco irá começar na posição atual e terá o tamanho especificado pelo argumento `<tamanho>`. A definição de `<nome>` poderá ser usada para referenciar-se ao início do bloco.

Exemplo

```
BUFFER RES    64     ;Reserva 64 palavras
```

Veja também

ORG, FILL

SET - DEFINE UMA VARIÁVEL

Sintaxe

```
<nome> SET <expr>
```

Descrição

Define uma variável para ser utilizada nas expressões que serão interpretadas pelo compilador. Como o próprio nome diz, o valor determinado por `<expr>` será associado ao `<nome>`, podendo ser alterado no decorrer do programa.

Exemplo

```
COMP   SET    .10
LARG   SET    .20
ALTURA SET    .15
AREA   SET    COMP * LARG
VOL            SET    AREA * ALTURA
```

Veja também

EQU, VARIABLE

SPACE - INSERE LINHAS EM BRANCO

Sintaxe

```
SPACE <expr>
```

Descrição

Insere <expr> número de linhas em branco na geração do arquivo de listagem do programa (.LST).

Exemplo

```
SPACE 3              ;insere 3 linhas em branco
```

Veja também

LIST, TITLE, SUBTITLE, SPACE, PAGE

SUBTITLE - DETERMINA O SUBTÍTULO DO PROGRAMA

Sintaxe

```
SUBTITLE     "<texto>"
```

Descrição

O <texto> pode ter até 60 caracteres e será utilizado como segunda linha no cabeçalho de toda página no arquivo de listagem do programa (.LST).

Exemplo

```
SUBTITLE     " Desenvolvido pela MOSAICO ENGENHARIA"
```

Veja também

LIST, TITLE

TITLE - DETERMINA O TÍTULO DO PROGRAMA

Sintaxe

```
TITLE "<texto>"
```

Descrição

O <texto> pode ter até 60 caracteres e será utilizado no cabeçalho de toda página no arquivo de listagem do programa (.LST).

Exemplo

```
TITLE "Programa de exemplo"
```

Veja também

LIST, SUBTITLE

UDATA - DECLARA O INÍCIO DE UMA SEÇÃO DE DADOS NÃO INICIALIZADOS

Sintaxe

```
[<nome>]      UDATA  [<RAM endereço>]
```

Descrição

Usado com objetos. Declara o início de uma seção de dados não inicializados. Se o argumento `<nome>` não for especificado, a seção será chamada de UDATA. O endereço inicial para os dados é definido por `<RAM endereço>`, ou será adotado o valor zero caso esse argumento não seja especificado. Nenhum código será gerado nesta seção. A diretriz RES deve ser usada para reservar espaço para os dados.

Exemplo

```
            UDATA
            VAR1          RES     .1
            DOUBLE        RES     .2
```

Veja também

TEXT, IDATA, UDATA_OVR, UDATA_SHR, EXTERN, GLOBAL

UDATA_OVR - DECLARA O INÍCIO DE UMA SEÇÃO DE DADOS SOBRECARREGADOS

Sintaxe

```
[<nome>]      UDATA_OVR           [<RAM endereço>]
```

Descrição

Usado com objetos. Declara o início de uma seção de dados não inicializados. Se o argumento `<nome>` não for especificado, a seção será chamada de UDATA_OVR. O endereço inicial para os dados é definido por `<RAM endereço>`, ou será adotado o valor zero caso este argumento não seja especificado. O espaço declarado nesta seção pode ser sobrecarregado por outras seções desse tipo que possuam o mesmo nome. Isso é ideal para declarar variáveis temporárias que devem ocupar o mesmo espaço na memória. Nenhum código será gerado nesta seção. A diretriz RES deve ser usada para reservar espaço para os dados.

Exemplo

```
TEMPS  UDATA_OVR
TEMP1  RES     .1
TEMP2  RES     .1
TEMP2  RES     .1

TEMPS  UDATA_OVR
LONGTEMP1      RES     .2
LONGTEMP2      RES     .2
```

Veja também

TEXT, IDATA, UDATA, EXTERN, GLOBAL, UDATA_SHR

UDATA_SHR - DECLARA O INÍCIO DE UMA SEÇÃO DE DADOS COMPARTILHADOS

Sintaxe

```
[<nome>]      UDATA_SHR          [<RAM endereço>]
```

Descrição

Usado com objetos. Declara o início de uma seção de dados compartilhados e não inicializados. Se o argumento `<nome>` não for especificado, a seção será chamada de UDATA_OVR. O endereço inicial para os dados é definido por `<RAM endereço>`, ou será adotado o valor zero caso este argumento não seja especificado. Os dados declarados nesta seção podem ser acessados de qualquer banco da RAM. Isso é ideal para declarar variáveis que são utilizadas em rotinas que trabalham com mais de um banco. Nenhum código será gerado nesta seção. A diretriz RES deve ser usada para reservar espaço para os dados.

Exemplo

```
TEMPS  UDATA_OVR
TEMP1  RES    .1
TEMP2  RES    .1
TEMP2  RES    .1
```

Veja também

TEXT, IDATA, UDATA, EXTERN, GLOBAL, UDATA_OVR

#UNDEFINE - ELIMINA UMA SUBSTITUIÇÃO DE TEXTO

Sintaxe

```
#UNDEFINE          <nome>
```

Descrição

Elimina definição anteriormente criada pela diretriz #DEFINE.

Exemplo

```
#DEFINE      LED          PORTA,0
.
.
.
#UNDEFINE    LED
```

Veja também

IFDEF, IFNDEF, #DEFINE

VARIABLE - DEFINE UMA VARIÁVEL

Sintaxe

```
VARIABLE     <nome> [= <expr> ,<nome> = <expr>]
```

Descrição

Define uma variável para ser utilizada nas expressões que serão interpretadas pelo compilador. Esta diretriz é similar à diretriz SET, com a diferença de que a variável não precisa ser inicializada no momento da sua definição. Como o próprio nome diz, o valor determinado por <expr> será associado ao <nome>, podendo ser alterado no decorrer do programa.

Exemplo

```
VARIABLE      I, CONTA = 5
.
.
I = 0
      WHILE  I < CONTA
      .
      .
I += 1
      ENDW
```

Veja também

SET, CONSTANT

WHILE - LOOP ENQUANTO A EXPRESSÃO FOR VERDADEIRA

Sintaxe

```
WHILE  <expr>
              .
              .
              .
ENDW
```

Descrição

Enquanto o teste da <expr> resultar em verdadeiro, o bloco definido entre essa diretriz e a ENDW será executado. Caso contrário, ele será pulado. Lembre-se de que o valor 0 (zero) será considerado falso, enquanto qualquer outro número será considerado verdadeiro. O bloco poderá ter no máximo cem linhas e poderá ser repetido até 256 vezes.

Exemplo

```
VARIABLE      I
CONSTANT      CONTA = 5
.
.
I = 0
      WHILE  I < CONTA
      .
      (bloco que será repetido 5 vezes)
      .
I += 1
      ENDW
```

Veja também

ENDW

APÊNDICE D

INSTRUÇÕES ESPECIAIS

A tabela seguinte mostra várias instruções especiais aceitas pelo compilador para facilitar a digitação do programa. Estas instruções não fazem parte do set de instruções, pois na verdade o compilador irá substituí-las pelas instruções equivalentes. Entretanto, muitos programadores utilizam-se delas na hora de escrever seus programas, e por isso é bom conhecê-las para possibilitar o entendimento e manutenção de sistemas escritos por terceiros. A tabela mostra ainda os bits de STATUS afetados pela operação.

Instrução	Descrição	Instruções Equivalentes	Status
`ADDCF f,d`	Se houve carry, incrementa o registrador f, guardando o resultado em d.	`BTFSC STATUS,C` `INCF f,d`	Z
`ADDDCF f,d`	Se houve digit carry, incrementa o registrador f, guardando o resultado em d.	`BTFSC STATUS,DC` `INCF f,d`	Z
`B k`	Pula (branch) para a posição definida por k.	`GOTO k`	
`BC k`	Pula (branch) para a posição definida por k, caso tenha ocorrido um carry.	`BTFSC STATUS,C` `GOTO k`	
`BDC k`	Pula (branch) para a posição definida por k, caso tenha ocorrido um digit carry.	`BTFSC STATUS,DC` `GOTO k`	
`BNC k`	Pula (branch) para a posição definida por k, caso não tenha ocorrido um carry.	`BTFSS STATUS,C` `GOTO k`	
`BNDC k`	Pula (branch) para a posição definida por k, caso não tenha ocorrido um digit carry.	`BTFSS STATUS,DC` `GOTO k`	
`BNZ k`	Pula (branch) para a posição definida por k, caso não tenha ocorrido um zero.	`BTFSS STATUS,Z` `GOTO k`	

Instrução	Descrição	Instruções Equivalentes	Status
BZ k	Pula (branch) para a posição definida por k, caso tenha ocorrido um zero.	BTFSC STATUS,Z GOTO k	
CLRC	Limpa o bit de carry.	BCF STATUS,C	C
CLRDC	Limpa o bit de digit carry.	BCF STATUS,DC	DC
CLRZ	Limpa o bit de zero.	BCF STATUS,Z	Z
LCALL k	Chamada de rotina (k) distante. Acerta o PCLATH (bits 3 e 4) automaticamente.	BCF/BSF PCLATH,3 BCF/BSF PCLATH,4 CALL k	
LGOTO k	Desvio de programa (k) distante. Acerta o PCLATH (bits 3 e 4) automaticamente.	BCF/BSF PCLATH,3 BCF/BSF PCLATH,4 GOTO k	
MOVFW f	Move o valor do registrador f para W.	MOVF f,W	Z
NEGF f,d	Acerta o valor do resultado negativo de uma conta.	COMF f,F INCF f,d	Z
SETC	Seta o bit de carry.	BSF STATUS,C	C
SETDC	Seta o bit de digit carry.	BSF STATUS,DC	DC
SETZ	Seta o bit de zero.	BSF STATUS,Z	Z
SKPC	Pula a próxima linha se houve um carry.	BTFSS STATUS,C	
SKPDC	Pula a próxima linha se houve um digit carry.	BTFSS STATUS,DC	
SKPNC	Pula a próxima linha se não houve um carry.	BTFSC STATUS,C	
SKPNDC	Pula a próxima linha se não houve um digit carry.	BTFSC STATUS,DC	
SKPNZ	Pula a próxima linha se não houve um zero.	BTFSC STATUS,Z	
SKPZ	Pula a próxima linha se houve um zero.	BTFSS STATUS,Z	
SUBCF f,d	Se houve carry, decrementa o registrador f, guardando o resultado em d.	BTFSC STATUS,C DECF f,d	Z
SUBDCF f,d	Se houve digit carry, decrementa o registrador f, guardando o resultado em d.	BTFSC STATUS,DC DECF f,d	Z
TSTF f	Testa o registrador f para saber se é zero.	MOVF f,F	Z

APÊNDICE E

OPERADORES DO MPASM

Operador	Descrição	Exemplo
$	Número da atual linha de programa (PC).	`GOTO $ + 3`
(	Abertura de parênteses.	`1 + (R * 4)`
)	Fechamento de parênteses.	`(COMPR + 1) * 256`
!	NÃO lógico.	`IF ! (A - B)`
-	Negativo.	`-1 * COMPR`
~	Complemento.	`FLAGS = ~FLAGS`
high	Byte alto de um símbolo de dois bytes.	`MOVLW high CTR_TABELA`
low	Byte baixo de um símbolo de dois bytes.	`MOVLW low CTR_TABELA`
upper	Byte superior de um símbolo de três bytes.	`MOVLW upper CTR_TABELA`
*	Multiplicação.	`A = B * C`
/	Divisão.	`A = B / C`
%	Resto da divisão.	`COMPR = TOTAL % 16`
+	Soma.	`TOTAL = VALOR * 8 + 1`
-	Subtração.	`TOTAL = VALOR - 10`
<<	Rotação para a esquerda.	`VALOR = FLAGS << 1`
>>	Rotação para a direita.	`VALOR = FLAGS >> 2`
>=	Maior ou igual.	`IF INDICE >= NUM`
>	Maior que.	`IF INDICE > NUM`
<	Menor que.	`IF INDICE < NUM`
<=	Menor ou igual.	`IF INDICE <= NUM`
==	Igual a.	`IF INDICE == NUM`
!=	Diferente de (NÃO igual).	`IF INDICE != NUM`
&	Operação E (AND) bit a bit.	`FLAGS = FLAGS & MASCARA`
^	Operação OU exclusivo (XOR) bit a bit.	`FLAGS = FLAGS ^ MASCARA`
|	Operação OU inclusivo (IOR) bit a bit.	`FLAGS = FLAGS | MASCARA`
&&	E lógico (AND).	`IF (COMPR == 10) && (A > B)`
||	OU lógico (OR).	`IF (COMPR == 10) || (A != B)`
=	Variável = valor.	`INDICE = 0`

Operador	Descrição	Exemplo
+=	Variável = ela mesma + valor.	`INDICE += 1`
-=	Variável = ela mesma - valor.	`INDICE -= 1`
*=	Variável = ela mesma * valor.	`INDICE *= TOTAL`
/=	Variável = ela mesma / valor.	`INDICE /= TOTAL`
%=	Variável = resto da divisão dela mesma pelo valor.	`INDICE %= 16`
<<=	Variável = ela mesma rotacionada à esquerda.	`INDICE <<= 3`
>>=	Variável = ela mesma rotacionada à direita.	`INDICE >>= 2`
&=	Variável = ela mesma AND valor	`INDICE &= MASCARA`
|=	Variável = ela mesma IOR valor	`INDICE |= MASCARA`
^=	Variável = ela mesma XOR valor	`INDICE ^= MASCARA`
++	Variável = ela mesma + 1 (incremento)	`INDICE ++`
--	Variável = ela mesma - 1 (decremento)	`INDICE --`

APÊNDICE F

TABELAS

Caracteres ASCII									
Valores em HEX		Dígito mais significativo							
		0	1	2	3	4	5	6	7
Dígito menos significativo	0			Espaço	0	@	P	`	p
	1	SOH	DC1	!	1	A	Q	a	q
	2	STX	DC2	"	2	B	R	b	r
	3	ETX	DC3	#	3	C	S	c	s
	4	EOT	DC4	$	4	D	T	d	t
	5	ENQ	NAK	%	5	E	U	e	u
	6	ACK	SYN	&	6	F	V	f	v
	7	BELL	ETB	'	7	G	W	g	w
	8	BS	CAN	(	8	H	X	h	x
	9	HT	EM	)	9	I	Y	i	y
	A	LF	SUB	*	:	J	Z	j	z
	B	VT	ESC	+	;	K	[	k	{
	C	FF	FS	,	<	L	\	l	|
	D	CR	GS	-	=	M	]	m	}
	E	SO	RS	.	>	N	^	n	~
	F	SI	US	/	?	O	_	o	DEL

Conversão Hexadecimal → Decimal							
Byte 2				Byte 1			
HEX	DEC	HEX	DEC	HEX	DEC	HEX	DEC
0	0	0	0	0	0	0	0
1	4096	1	256	1	16	1	1
2	8192	2	512	2	32	2	2
3	12288	3	768	3	48	3	3
4	16384	4	1024	4	64	4	4
5	20480	5	1280	5	80	5	5
6	24576	6	1536	6	96	6	6
7	28672	7	1792	7	112	7	7
8	32768	8	2048	8	128	8	8
9	36864	9	2304	9	144	9	9
A	40960	A	2560	A	160	A	10
B	45056	B	2816	B	176	B	11
C	49152	C	3072	C	192	C	12
D	53248	D	3328	D	208	D	13
E	57344	E	3584	E	224	E	14
F	61440	F	3840	F	240	F	15

Para usar esta tabela, some o valor decimal correspondente a cada dígito do número hexadecimal. Por exemplo, o número HEX A38F eqüivale a 41871 em decimal.

HEX (dígito 4)	HEX (dígito 3)	HEX (dígito 2)	HEX (dígito 1)	Resultado
A	3	8	F	A38F
40960	+ 768	+ 128	+ 15	= 41871