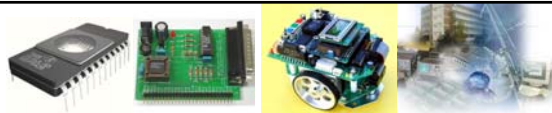


INSTRUCCIONES DE ENSAMBLADOR



ORGANIZACIÓN de la MEMORIA

Dentro del PIC16F877 se distinguen tres bloques de memoria.

Memoria de programa

En sus 8192 posiciones (8K) contiene el programa con las instrucciones que gobiernan la aplicación. Es del tipo no volátil.

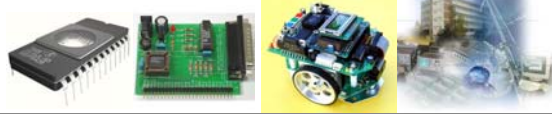
Memoria de datos RAM

Guarda las variables y datos. Consta de varios registros de 8 bits. Es volátil.

Memoria EEPROM de datos

Es una pequeña área de memoria de datos de lectura y escritura no volátil que permite garantizar que determinada información estará siempre disponible al reiniciarse el programa. Se gestiona de manera distinta a la memoria de datos RAM.

Organización memoria



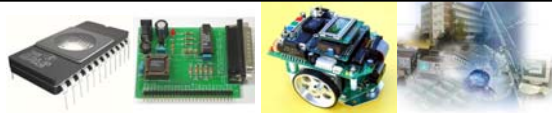
La memoria de programa

0000h	RESET
0001h	
0002h	
0003h	
0004h	INT
0005h	
0006h	
0007h	
0008h	
0009h	
000Ah	
...	
1FFDh	
1FFEh	
1FFFh	

- Almacena todas las instrucciones del **programa de control**, que debe estar grabado de forma permanente.
- La información contenida en esta memoria se graba previamente mediante un equipo físico denominado **programador o grabador**.
- El PIC16F877 tiene una memoria de programa no volátil denominada **ROM Flash** que admite unas 1000 grabaciones.
- La memoria de programa está organizada en **palabras de 14 bits** cada una.
- Todas las instrucciones **ocupan una posición de memoria** de programa



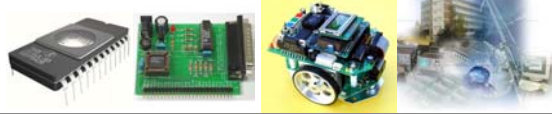
Organización memoria



- Almacena todos los **datos que se manejan** en un programa.
- Se distinguen dos tipos de registro:
 - **Registros de funciones especiales SFR**. Son los primeros registros. Cada uno de ellos cumple un propósito especial en el control del PIC.
 - **Registros de propósito general GPR**. Se pueden usar para guardar datos temporales. El PIC16F877 dispone de 368.
- Cuenta con cuatro bancos de memoria: Bancos 0, 1, 2 y 3.
 - Los SFR aparecen de la dirección 00h a 1Fh del Banco 0, de 80h a 9Fh del Banco 1, de 100h a 10F en el Banco 2 y de 180h a 18Fh del Banco 3. Algunos son accesibles desde dos o más bancos.
 - Los GPR ocupan 368 posiciones de memoria. Algunas posiciones de los Bancos 1 a 3 se mapean sobre el Banco 0.
 - Existen zonas de memoria no empleadas que devuelven '0' en caso de lectura.



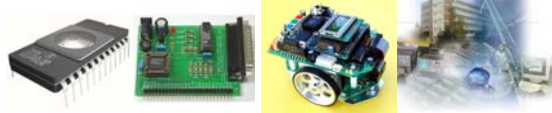
Organización memoria



File Address	File Address	File Address	File Address
Indirect addr (*)	Indirect addr (*)	Indirect addr (*)	Indirect addr (*)
00h	80h	100h	180h
TMR0	OPTION_REG	TMR0	OPTION_REG
01h	81h	101h	181h
PCL	PCL	PCL	PCL
02h	82h	102h	182h
STATUS	STATUS	STATUS	STATUS
03h	83h	103h	183h
FSR	FSR	FSR	FSR
04h	84h	104h	184h
PORTA	TRISA	PORTA	TRISA
05h	85h	105h	185h
PORTB	TRISB	PORTB	TRISB
06h	86h	106h	186h
PORTC	TRISC	PORTC	TRISC
07h	87h	107h	187h
PORTD	TRISD	PORTD	TRISD
08h	88h	108h	188h
PORTF	TRISF	PORTF	TRISF
09h	89h	109h	189h
PCLATH	PCLATH	PCLATH	PCLATH
0Ah	8Ah	10Ah	18Ah
INTCON	INTCON	INTCON	INTCON
0Bh	8Bh	10Bh	18Bh
PIR1	PIE1	EECON1	EECON1
0Ch	8Ch	10Ch	18Ch
PIR2	PIE2	EECON2	EECON2
0Dh	8Dh	10Dh	18Dh
TMR1L	PCON	EEDATH	EEDATH
0Eh	8Eh	10Eh	18Eh
TMR1H	PCON	EEDATH	EEDATH
0Fh	8Fh	10Fh	18Fh
YICON	SSPCON2	EEADR	EEADR
10h	90h	110h	190h
TMR2	PIR2	111h	191h
11h	91h	112h	192h
TZCON	SSPAD0	113h	193h
12h	92h	114h	194h
SSPBUF	SSPSTAT	115h	195h
13h	93h	116h	196h
SSPCON	117h	117h	197h
14h	94h	118h	198h
CCPR1L	119h	119h	199h
CCPR1H	120h	120h	200h
15h	95h	121h	201h
CCP1CON	General Purpose Register 10 Bytes	122h	202h
16h	96h	123h	203h
RCSTA	TXSTA	124h	204h
17h	97h	125h	205h
TXREG	SPBRG	126h	206h
18h	98h	127h	207h
RCREG	128h	128h	208h
19h	99h	129h	209h
CCPR2L	130h	130h	210h
CCPR2H	131h	131h	211h
1Ah	100h	132h	212h
CCP2CON	ADRESL	133h	213h
1Bh	101h	134h	214h
ADRESH	ADCON1	135h	215h
1Ch	102h	136h	216h
ADCON0	103h	137h	217h
1Dh	104h	138h	218h
1Eh	105h	139h	219h
1Fh	106h	140h	220h
20h	107h	141h	221h
General Purpose Register 96 Bytes	General Purpose Register 80 Bytes	General Purpose Register 80 Bytes	General Purpose Register 80 Bytes
7Fh	FFh	17Fh	1FFh
Bank 0	Bank 1	Bank 2	Bank 3



Organización memoria

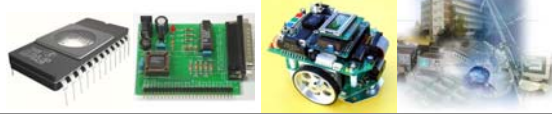


LENGUAJE ENSAMBLADOR

- El único lenguaje que entienden los microcontroladores es el **lenguaje máquina** formado por ceros y unos del sistema binario.
- El lenguaje ensamblador expresa las instrucciones de una forma más natural al hombre a la vez que muy cercana al microcontrolador, ya que **cada una de sus instrucciones se corresponde con otra en código máquina**.
- El lenguaje ensamblador trabaja con **nemónicos**, que son grupos de caracteres alfanuméricos que simbolizan las órdenes o tareas a realizar.
- La traducción de los nemónicos a código máquina entendible por el microcontrolador la lleva a cabo un **programa ensamblador**.
- El programa escrito en lenguaje ensamblador se denomina código fuente (***.asm**). El programa ensamblador proporciona a partir de este fichero el correspondiente código máquina, que suele tener la extensión ***.hex**.



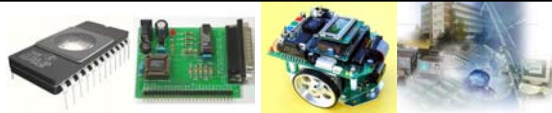
Organización memoria



El DISEÑO con microcontroladores constituye un proceso CÍCLICO:



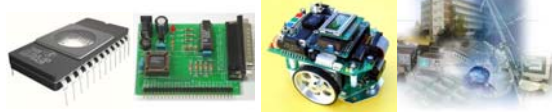
Organización memoria



¿ Dónde estamos ahora dentro del proceso?



Juego de instrucciones



El juego de instrucciones

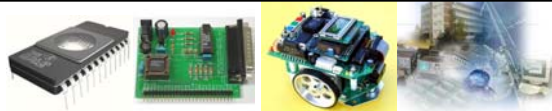
- Es un **juego reducido** de 35 instrucciones muy simples y rápidas.
- La mayoría de las instrucciones se ejecuta en **4 ciclos de reloj**; los saltos se ejecutan en 8.
- Todas las instrucciones tienen la misma longitud: **14 bits**.

Instrucciones de CARGA

NEMÓNICO	DESCRIPCIÓN	FLAGS AFECTADOS
clrf f	00 → (f)	Z
clrw	00 → (W)	Z
movf f,d	(f) → (destino)	Z
movlw k	k → (W)	Ninguno
movwf f	(W) → (f)	Ninguno

SAN VALERO

Juego de instrucciones



Instrucciones de BIT

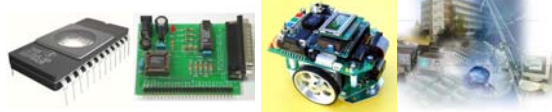
NEMÓNICO	DESCRIPCIÓN	FLAGS AFECTADOS
bcf f,b	Pone a 0 el bit 'b' del registro 'f'	Ninguno
bsf f,b	Pone a 1 el bit 'b' del registro 'f'	Ninguno

Instrucciones ARITMÉTICAS

NEMÓNICO	DESCRIPCIÓN	FLAGS AFECTADOS
addlw k	(W) + k → (W)	C - DC - Z
addwf f,d	(W) + (f) → (destino)	C - DC - Z
decf f,d	(f) - 1 → (destino)	Z
incf f,d	(f) + 1 → (destino)	Z
sublw k	K - (W) → (W)	C - DC - Z
subwf f,d	(f) - (W) → (destino)	C - DC - Z

SAN VALERO

Juego de instrucciones

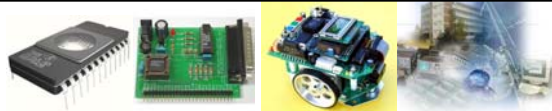


Instrucciones LÓGICAS

NEMÓNICO	DESCRIPCIÓN	FLAGS AFECTADOS
andlw k	(W) AND k → (W)	Z
andwf f,d	(W) AND (f) → (destino)	Z
comf f,d	(/f) → (destino)	Z
iorlw k	(W) OR k → (W)	Z
iorwf f,d	(W) OR (f) → (destino)	Z
rlf f,d	Rota (f) a izquierda → (destino)	C
rrf f,d	Rota (f) a derecha → (destino)	C
swap f,d	Intercambia nibbles (f) → (destino)	Ninguno
xorlw k	(W) XOR k → (W)	Z
xorwf f,d	(W) XOR (f) → (destino)	Z



Juego de instrucciones



Instrucciones de SALTO

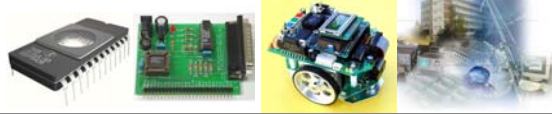
NEMÓNICO	DESCRIPCIÓN	FLAGS AFECTADOS
btfsc f,b	Salta si el bit 'b' de 'f' es 0	Ninguno
btfss f,b	Salta si el bit 'b' de 'f' es 1	Ninguno
decfsz f,d	(f) - 1 → (destino) y salta si es 0	Ninguno
incfsz f,d	(f) + 1 → (destino) y salta si es 0	Ninguno
goto k	Salta a la dirección 'k'	Ninguno

Instrucciones de manejo de SUBROUTINAS

NEMÓNICO	DESCRIPCIÓN	FLAGS AFECTADOS
call k	Llamada a subrutina	Ninguno
retfie	Retorno de una interrupción	Ninguno
retlw k	Retorno con un literal en (W)	Ninguno
return	Retorno de una subrutina	Ninguno



Juego de instrucciones

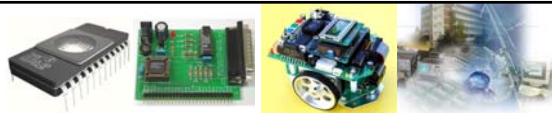


Instrucciones ESPECIALES

NEMÓNICO	DESCRIPCIÓN	FLAGS AFECTADOS
clrwdt	Borra Timer del Watchdog	/TO - /PD
nop	No operación	Ninguno
sleep	Entra en modo de bajo consumo	/TO - /PD



Código fuente



El código fuente

- Está compuesto por una sucesión de líneas de programa.
- Cada línea de programa puede estar compuesta de hasta cuatro campos o columnas separados por uno o más espacios o tabulaciones entre sí.
 - **Campo de etiquetas.** Expresiones alfanuméricas escogidas por el usuario para identificar una determinada instrucción del programa. Todas las etiquetas tienen asignado el valor de la posición de memoria en la que se encuentra la instrucción a la que acompañan.
 - **Campo del código de operación.** Corresponde al nemónico.
 - **Campo de operandos y datos.** Contiene los operandos que precisa el nemónico utilizado. Según la instrucción, puede haber dos, uno o ningún operando.
 - **Campo de comentarios.** Dentro de una línea, todo lo que se encuentre a continuación de un punto y coma (;) será ignorado y considerado como comentario.



Código fuente










- El ensamblador MPASM (distribuido por Microchip) soporta los sistemas de numeración **decimal**, **hexadecimal**, **octal**, **binario** y **ASCII**.
- Los nemónicos que tengan una constante como operando deberán incluirla respetando la sintaxis que se indica a continuación.

TIPO	SINTAXIS		
Decimal	D' <valor>'	d' <valor>'	. <valor>
Hexadecimal	H' <valor>'	h' <valor>'	0x <valor>
	<valor> H	<valor> h	
Octal	O' <valor>'	o' <valor>'	
Binario	B' <valor>'	b' <valor>'	
ASCII	A' <carácter>'	a' <carácter>'	' <carácter>'
Cadena	" <cadena> "		

Las constantes hexadecimales que empiecen por una letra deben ir precedidas de un cero para no confundirlas con una etiqueta. Ejemplo: *movlw 0F7h*



Directivas


Directivas





- Controlan el proceso de ensamblado** del programa, pero no son parte del mismo (también se conocen como pseudoinstrucciones).
- Hay más de 50 directivas reconocidas por MPASM. A continuación se recogen algunas de las más habituales


END
Es la única directiva **obligatoria**. Indica al ensamblador dónde debe detener el proceso. Debe colocarse en la última línea del programa.

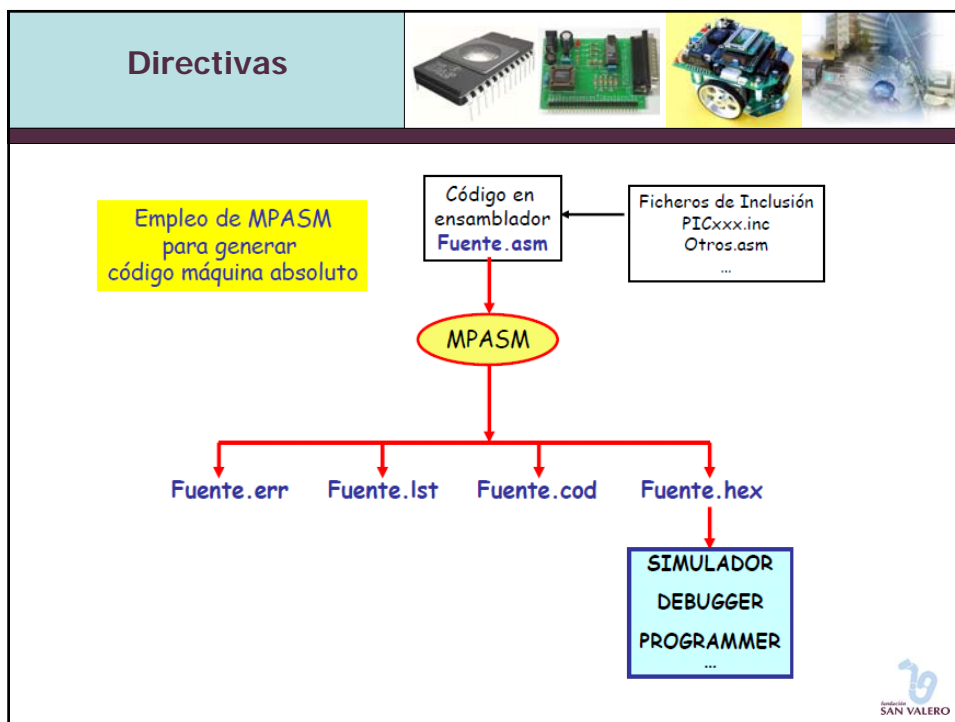
<etiqueta> **EQU** <expresión>
El valor <expresión> es asignado a <etiqueta>. Estas directivas se suelen colocar al principio del programa y habitualmente se usan para definir constantes y direcciones de memoria.

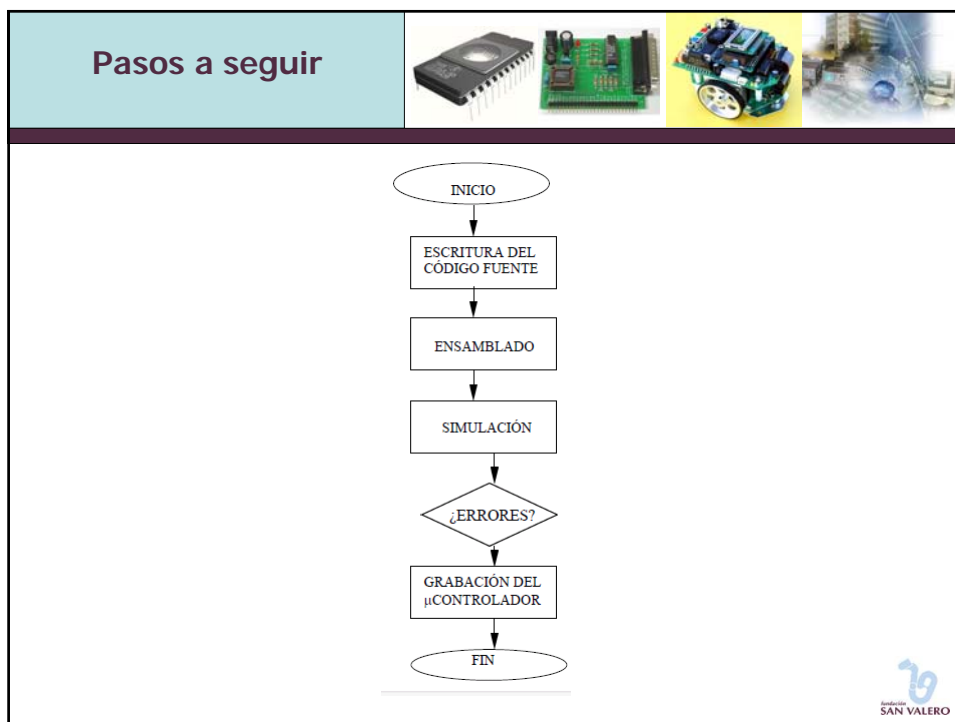
[<etiqueta>] **ORG** <expresión>
Las instrucciones del código fuente que siguen a esta directiva se ensamblan a partir de la posición indicada por <expresión>.



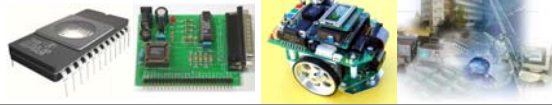
Directivas	   
<p>__CONFIG <expresión> [& <expresión> & ... & <expresión>]</p> <p>Permite indicar la configuración elegida para la grabación del PIC. Ejemplo: __CONFIG _CP_OFF & _WDT_OFF & _XT_OSC</p> <p>LIST P=16F877</p> <p>Indica el tipo de microcontrolador utilizado.</p> <p>INCLUDE <p16F877.inc> o INCLUDE "p16F877.inc"</p> <p>Incluye en el programa un fichero donde se definen las etiquetas con las que se nombra a los diferentes registros y sus bits. Este fichero se encuentra en el directorio principal del programa ensamblador. Puede usarse esta directiva para incluir cualquier otro fichero (¡Ojo! El fichero de inclusión no puede terminar con una directiva END).</p>	





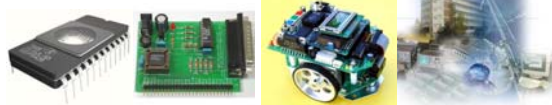


Instrucciones resumidas



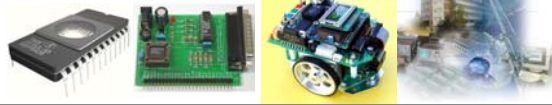
ADDLW Suma un literal Sintaxis: [label] ADDLW k Operandos: $0 \leq k \leq 255$ Operación: $(W) + (k) \Rightarrow (W)$ Flags afectados: C, DC, Z Código OP: 00 0111 dfff ffff Descripción: Suma el contenido del registro W y el registro f. Si d es 0, el resultado se almacena en W, si d es 1 se almacena en f. Ejemplo: ADDLW 0xC2 Antes: W = 0x17 Después: W = 0xD9	ADDWF W + F Sintaxis: [label] ADDWF f,d Operandos: $d \in \{0,1\}, 0 \leq f \leq 127$ Operación: $(W) + (f) \Rightarrow (dest)$ Flags afectados: C, DC, Z Código OP: 00 0111 dfff ffff Descripción: Suma el contenido del registro W y el registro f. Si d es 0, el resultado se almacena en W, si d es 1 se almacena en f. Ejemplo: ADDWF REG,0 Antes: W = 0x17, REG = 0xC2 Después: W = 0xD9, REG = 0xC2	ANDLW W AND literal Sintaxis: [label] ANDLW k Operandos: $0 \leq k \leq 255$ Operación: $(W) \text{ AND } (k) \Rightarrow (W)$ Flags afectados: Z Código OP: 11 1001 kkkk kkkk Descripción: Realiza la operación lógica AND entre el contenido del registro W y k, guardando el resultado en W. Ejemplo: ANDLW 0xC2 Antes: W = 0x17 Después: W = 0xD9	BTFSC Test de bit y salto Sintaxis: [label] BTFSC f,d Operandos: $d \in \{0,1\}, 0 \leq f \leq 127$ Operación: Salto si $(f-b)=0$ Flags afectados: Ninguno Código OP: 01 10bb bfff ffff Descripción: Si el bit b del registro f es 0, se salta una instrucción y se continúa con la ejecución. En caso de salto, ocupará dos ciclos de reloj. Ejemplo: BTFSC REG,6 GOTO NO_ES_0 SI_ES_0 Instrucción NO_ES_0 Instrucción	BTFSS Test de bit y salto Sintaxis: [label] BTFSS f,d Operandos: $d \in \{0,1\}, 0 \leq f \leq 127$ Operación: Salto si $(f-b)=1$ Flags afectados: Ninguno Código OP: 01 11bb bfff ffff Descripción: Si el bit b del registro f es 1, se salta una instrucción y se continúa con la ejecución. En caso de salto, ocupará dos ciclos de reloj. Ejemplo: BTFSS REG,6 GOTO NO_ES_0 SI_ES_0 Instrucción NO_ES_0 Instrucción
ANDWF W AND F Sintaxis: [label] ANDWF f,d Operandos: $d \in \{0,1\}, 0 \leq f \leq 127$ Operación: $(W) \text{ AND } (f) \Rightarrow (dest)$ Flags afectados: Z Código OP: 00 0101 dfff ffff Descripción: Realiza la operación lógica AND entre los registros W y f. Si d es 0, el resultado se almacena en W, si d es 1 se almacena en f. Ejemplo: ANDWF REG,0 Antes: W = 0x17, REG = 0xC2 Después: W = 0x17, REG = 0xC2	BCF Borra un bit Sintaxis: [label] BCF f,b Operandos: $0 \leq f \leq 127, 0 \leq b \leq 7$ Operación: $0 \Rightarrow (f-b) \Rightarrow 0$ Flags afectados: Ninguno Código OP: 01 00bb bfff ffff Descripción: Borra el bit b del registro f. Ejemplo: BCF REG,7 Antes: REG = 0xC7 Después: REG = 0x47	BSF Activa un bit Sintaxis: [label] BSF f,b Operandos: $0 \leq f \leq 127, 0 \leq b \leq 7$ Operación: $1 \Rightarrow (f-b) \Rightarrow 1$ Flags afectados: Ninguno Código OP: 01 01bb bfff ffff Descripción: Activa el bit b del registro f. Ejemplo: BSF REG,7 Antes: REG = 0x0A Después: REG = 0x8A	CALL Salto a subrutina Sintaxis: [label] CALL k Operandos: $0 \leq k \leq 2047$ Operación: $PC \Rightarrow Pila; k \Rightarrow PC$ Flags afectados: Ninguno Código OP: 10 0kkk kkkk kkkk Descripción: Salto a una subrutina. La parte baja de k se carga en PCL, y la alta en PCLATCH. Ocupa 2 ciclos de reloj. Ejemplo: ORIGEN CALL DESTINO Antes: PC = ORIGEN Después: PC = DESTINO	

Instrucciones resumidas



CLRF Borra un registro Sintaxis: [label] CLRF f Operandos: Ninguno Operación: $0x00 \Rightarrow (f), 1 \Rightarrow Z$ Flags afectados: Z Código OP: 00 0001 1fff ffff Descripción: El registro f se carga con 0x00. El flag Z se activa. Ejemplo: CLRF REG Antes: REG = 0x5A Después: REG = 0x00, Z = 1	CLRWF Borra el registro W Sintaxis: [label] CLRWF Operandos: Ninguno Operación: $0x00 \Rightarrow W, 1 \Rightarrow Z$ Flags afectados: Z Código OP: 00 0001 0xxx xxxx Descripción: El registro de trabajo W se carga con 0x00. El flag Z se activa. Ejemplo: CLRWF Antes: W = 0x5A Después: W = 0x00, Z = 1	CLRWDW Borra el WDT Sintaxis: [label] CLRWDW Operandos: Ninguno Operación: $0x00 \Rightarrow WDT, 1 \Rightarrow /TO$ Flags afectados: TO, PD Código OP: 00 0000 0110 0100 Descripción: Esta instrucción borra tanto el WDT como su prescaler. Los bits TO y PD del registro de estado se ponen a 1. Ejemplo: CLRWDW Después: Contador WDT = 0, Prescales WDT = 0, /TO = 1, PD = 1	GOTO Salto incondicional Sintaxis: [label] GOTO k Operandos: $0 \leq k \leq 2047$ Operación: $k \Rightarrow PC$ Flags afectados: Ninguno Código OP: 10 1kkk kkkk kkkk Descripción: Se trata de un salto incondicional. La parte baja de k se carga en PCL, y la alta en PCLATCH. Ocupa 2 ciclos de reloj. Ejemplo: ORIGEN GOTO DESTINO Antes: PC = ORIGEN Después: PC = DESTINO	INCF Decremento de f Sintaxis: [label] INCF f,d Operandos: $d \in \{0,1\}, 0 \leq f \leq 127$ Operación: $(f) - 1 \Rightarrow (dest)$ Flags afectados: Z Código OP: 00 1010 dfff ffff Descripción: Incrementa en 1 el contenido de f. Si d es 0, el resultado se almacena en W, si d es 1 se almacena en f. Ejemplo: INCF CONT,1 Antes: CONT = 0xFF, Z = 0 Después: CONT = 0x00, Z = 1
COMF Complemento de f Sintaxis: [label] COMF f,d Operandos: $d \in \{0,1\}, 0 \leq f \leq 127$ Operación: $(f) - 1 \Rightarrow (dest)$ Flags afectados: Z Código OP: 00 1001 dfff ffff Descripción: El registro f es complementado. El flag Z se activa si el resultado es 0. Si d es 0, el resultado se almacena en W, si d es 1 se almacena en f. Ejemplo: COMF REG,0 Antes: REG = 0x13 Después: REG = 0x13, W = 0x13	DECF Decremento de f Sintaxis: [label] DECF f,d Operandos: $d \in \{0,1\}, 0 \leq f \leq 127$ Operación: $(f) - 1 \Rightarrow (dest)$ Flags afectados: Z Código OP: 00 0011 dfff ffff Descripción: Decrementa en 1 el contenido de f. Si d es 0, el resultado se almacena en W, si d es 1 se almacena en f. Ejemplo: DECF CONT,1 Antes: CONT = 0x01, Z = 0 Después: CONT = 0x00, Z = 1	DECFSZ Decremento y salto Sintaxis: [label] DECFSZ f,d Operandos: $d \in \{0,1\}, 0 \leq f \leq 127$ Operación: $(f) - 1 \Rightarrow d$; Salto si R=0 Flags afectados: Ninguno Código OP: 00 1011 dfff ffff Descripción: Decrementa el contenido del registro f. Si d es 0, el resultado se almacena en W, si d es 1 se almacena en f. Si la resta es 0 salta la siguiente instrucción, en cuyo caso costará 2 ciclos. Ejemplo: DECFSZ REG,0 GOTO NO_ES_0 SI_ES_0 Instrucción NO_ES_0 Salta instrucción anterior	INCFSZ Incremento y salto Sintaxis: [label] INCFSZ f,d Operandos: $d \in \{0,1\}, 0 \leq f \leq 127$ Operación: $(f) + 1 \Rightarrow d$; Salto si R=0 Flags afectados: Ninguno Código OP: 00 1111 dfff ffff Descripción: Incrementa el contenido del registro f. Si d es 0, el resultado se almacena en W, si d es 1 se almacena en f. Si la resta es 0 salta la siguiente instrucción, en cuyo caso costará 2 ciclos. Ejemplo: INCFSZ REG,0 GOTO NO_ES_0 SI_ES_0 Instrucción NO_ES_0 Salta instrucción anterior	

Instrucciones resumidas



IORLW WOR literal

Sintaxis: [label] IORLW k
Operando: $0 \leq k \leq 255$
Operación: $(W) \text{ OR } (k) \Rightarrow (W)$
Flags afectados: Z
Código OP: 11 1000 kkkk kkkk

Descripción: Se realiza la operación lógica OR entre los registros W y k, guardando el resultado en W.

Ejemplo: IORLW 0x35
Antes: W = 0x0A
Después: W = 0xBF

IORWF W AND F

Sintaxis: [label] IORWF f,d
Operando: $d \in [0,1], 0 \leq f \leq 127$
Operación: $(W) \text{ OR } (f) \Rightarrow (\text{dest})$
Flags afectados: Z
Código OP: 00 0100 dfff ffff

Descripción: Realiza la operación lógica OR entre los registros W y f. Si d es 0, el resultado se almacena en W, si d es 1 se almacena en f.

Ejemplo: IORWF REG,0
Antes: W = 0x91, REG = 0x13
Después: W = 0x93, REG = 0x13

MOVLW Cargar literal en W

Sintaxis: [label] MOVLW f
Operando: $0 \leq f \leq 255$
Operación: $(k) \Rightarrow (W)$
Flags afectados: Ninguno
Código OP: 11 00xx kkkk kkkk

Descripción: El literal k pasa al registro W.

Ejemplo: MOVLW 0x5A
Después: REG = 0x4F, W = 0x5A

RETFIE Retorno de interrup.

Sintaxis: [label] RETFIE
Operando: Ninguno
Operación: $1 \Rightarrow \text{GIE}; \text{TOS} \Rightarrow \text{PC}$
Flags afectados: Ninguno
Código OP: 00 0000 0000 1001

Descripción: El PC se carga con el contenido de la cima de la pila (TOS): dirección de retorno. Consume 2 ciclos. Las interrupciones vuelven a ser habilitadas.

Ejemplo: RETFIE
Después: PC = dirección de retorno
GIE = 1

RETLW Retorno, carga W

Sintaxis: [label] RETLW k
Operando: $0 \leq k \leq 255$
Operación: $(k) \Rightarrow (W); \text{TOS} \Rightarrow \text{PC}$
Flags afectados: Ninguno
Código OP: 11 01xx kkkk kkkk

Descripción: El registro W se carga con la constante k. El PC se carga con el contenido de la cima de la pila (TOS): dirección de retorno. Consume 2 ciclos.

Ejemplo: RETLW 0x37
Después: PC = dirección de retorno
W = 0x37

MOVF Mover a f

Sintaxis: [label] MOVF f,d
Operando: $d \in [0,1], 0 \leq f \leq 127$
Operación: $(f) \Rightarrow (\text{dest})$
Flags afectados: Z
Código OP: 00 1000 dfff ffff

Descripción: El contenido del registro f se mueve al destino d. Si d es 0, el resultado se almacena en W, si d es 1 se almacena en f. Permite verificar el registro, puesto que afecta a Z.

Ejemplo: MOVF REG,0
Después: W = REG

MOVWF Mover a f

Sintaxis: [label] MOVWF f
Operando: $0 \leq f \leq 127$
Operación: $W \Rightarrow (f)$
Flags afectados: Ninguno
Código OP: 00 0000 1fff ffff

Descripción: El contenido del registro W pasa al registro f.

Ejemplo: MOVWF REG,0
Antes: REG = 0xFF, W = 0x4F
Después: REG = 0x4F, W = 0x4F

NOP No opera

Sintaxis: [label] NOP
Operando: Ninguno
Operación: No opera
Flags afectados: Ninguno
Código OP: 00 0000 0xxx 0000

Descripción: No realiza operación alguna. En realidad consume un ciclo de instrucción sin hacer nada.

Ejemplo: CLRWD
Después: Contador WDT = 0,
Prescaler WDT = 0,
TO = 1, PD = 1

RETURN Retorno de rutina

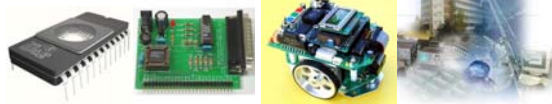
Sintaxis: [label] RETURN
Operando: Ninguno
Operación: $\text{TOS} \Rightarrow \text{PC}$
Flags afectados: Ninguno
Código OP: 00 0000 0000 1000

Descripción: El PC se carga con el contenido de la cima de la pila (TOS): dirección de retorno. Consume 2 ciclos.

Ejemplo: RETURN
Después: PC = dirección de retorno



Instrucciones resumidas



RLF Rota f a la izquierda

Sintaxis: [label] RLF f,d
Operando: $d \in [0,1], 0 \leq f \leq 127$
Operación: Rotación a la izquierda
Flags afectados: C
Código OP: 00 1100 dfff ffff

Descripción: El contenido de f se rota a la izquierda. El bit de menos peso de f pasa al carry (C), y el carry se coloca en el de mayor peso. Si d es 0, el resultado se almacena en W, si d es 1 se almacena en f.

Ejemplo: RLF REG,0
Antes: REG = 1110 0110, C = 0
Después: REG = 1110 0110,
W = 1100 1100, C = 1

RRF Rota f a la derecha

Sintaxis: [label] RRF f,d
Operando: $d \in [0,1], 0 \leq f \leq 127$
Operación: Rotación a la derecha
Flags afectados: C
Código OP: 00 1100 dfff ffff

Descripción: El contenido de f se rota a la derecha. El bit de menos peso de f pasa al carry (C), y el carry se coloca en el de mayor peso. Si d es 0, el resultado se almacena en W, si d es 1 se almacena en f.

Ejemplo: RRF REG,0
Antes: REG = 1110 0110, C = 1
Después: REG = 1110 0110,
W = 0110 0011, C = 0

SLEEP Modo bajo consumo

Sintaxis: [label] SLEEP
Operando: Ninguno
Operación: $(\text{on}) \Rightarrow \text{WDT}, 1 \Rightarrow \text{TO}$
 $0 \Rightarrow \text{WDT Prescaler}, 0 \Rightarrow \text{PD}$
Flags afectados: /PD, TO
Código OP: 00 0000 0110 0011

Descripción: El bit de energía se pone a 0, y a 1 el de descanso. El WDT y su prescaler se borran. El micro para el oscilador, limando al modo "dormiente".

Ejemplo: SLEEP
Prescaler WDT = 0,
TO = 1, PD = 1

XORLW WOR literal

Sintaxis: [label] XORLW k
Operando: $0 \leq k \leq 255$
Operación: $(W) \text{ XOR } (k) \Rightarrow (W)$
Flags afectados: Z
Código OP: 11 1010 kkkk kkkk

Descripción: Se realiza la operación lógica XOR entre el contenido del registro W y k, guardando el resultado en W.

Ejemplo: XORLW 0x4F
Antes: W = 0xB5
Después: W = 0x1A

XORWF W AND F

Sintaxis: [label] XORWF f,d
Operando: $d \in [0,1], 0 \leq f \leq 127$
Operación: $(W) \text{ XOR } (f) \Rightarrow (\text{dest})$
Flags afectados: Z
Código OP: 00 0110 dfff ffff

Descripción: Realiza la operación lógica XOR entre los registros W y f. Si d es 0, el resultado se almacena en W, si d es 1 se almacena en f.

Ejemplo: XORWF REG,0
Antes: W = 0xB5, REG = 0xAF
Después: W = 0xB5, REG = 0x1A

SUBLW Resta Literal - W

Sintaxis: [label] SUBLW k
Operando: $0 \leq k \leq 255$
Operación: $(k) - (W) \Rightarrow (W)$
Flags afectados: Z, C, DC
Código OP: 11 110x kkkk kkkk
Descripción: Mediante el método del complemento a dos el contenido de W es restado al literal. El resultado se almacena en W.

Ejemplo: SUBLW 0x02
Antes: W=1, C=? Después: W=1, C=1
Antes: W=2, C=? Después: W=0, C=1
Antes: W=3, C=? Después: W=FF, C=0
(El resultado es negativo)

SUBWF Resta f - W

Sintaxis: [label] SUBWF f,d
Operando: $d \in [0,1], 0 \leq f \leq 127$
Operación: $(f) - (W) \Rightarrow (\text{dest})$
Flags afectados: C, DC, Z
Código OP: 00 0010 dfff ffff
Descripción: Mediante el método del complemento a dos el contenido de W es restado al de f. Si d es 0, el resultado se almacena en W, si d es 1 se almacena en f.

Ejemplo: SUBWF REG,1
Antes: REG = 0x03, W = 0x02, C = ?
Después: REG = 0x01, W = 0x4F, C = 1
Antes: REG = 0x02, W = 0x02, C = ?
Después: REG = 0x00, W = 0x02, C = 1
Antes: REG = 0x01, W = 0x01, C = ?
Después: REG = 0xFF, W = 0x02, C = 0
(Resultado negativo)

SWAPF Intercambio de f

Sintaxis: [label] SWAPF f,d
Operando: $d \in [0,1], 0 \leq f \leq 127$
Operación: $(f < 3:0) \Rightarrow (f < 7:4)$
Flags afectados: Ninguno
Código OP: 00 1110 dfff ffff
Descripción: Los 4 bits de más peso y los 4 de menos son intercambiados. Si d es 0, el resultado se almacena en W, si d es 1 se almacena en f.

Ejemplo: SWAPF REG,0
Antes: REG = 0xAF
Después: REG = 0x5A, W = 0x5A

