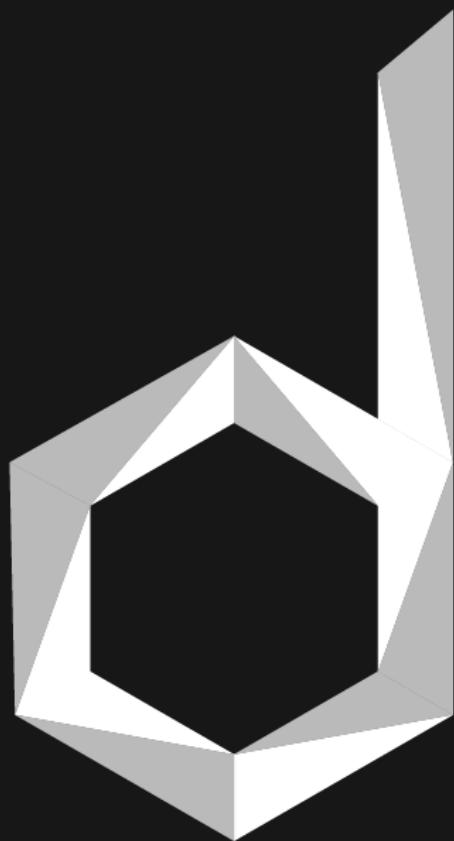


INGENIERÍA MECATRÓNICA



DI_CERO

DIEGO CERVANTES RODRÍGUEZ

ENSAMBLADOR EN MICROCONTROLADORES

MPLAB X IDE v5.50

Funcionamiento, Instrucciones y
Directivas del PIC16F887

Contenido

Funcionamiento PIC	4
Requerimientos Básicos Funcionamiento Microcontrolador:.....	4
• Su alimentación:.....	4
• Un oscilador:	4
• Memorias:.....	4
Registros Microcontrolador:.....	5
• Registro acumulador:.....	5
• Registros de propósito general:.....	5
• Registro Brown-Out:	6
• Banco o página de registros:.....	6
• Registro de propósito específico:	8
• Registro de estado (STATUS) o chismoso:	10
• Bandera:.....	11
• Contador de programa o Program Counter (PC):	12
• Pila o memoria ROM (Read Only Memory):	14
• Watchdog:.....	15
• Ciclo de reloj:	16
• Arquitectura del PIC16F887:.....	16
• Ciclos de máquina:.....	16
• Puertos A, B, C, D y E:.....	17
• Registros ANSEL y ANSELH:.....	21
• Registros TRIS:.....	23
35 instrucciones PIC:.....	25
REGLAS.....	25
1. ADDWF f, d (C, DC, Z):.....	26
2. ANDWF f, d (Z):.....	27
3. CLRF f (Z):	27
4. CLRW k (Z):.....	27
5. COMF f, d (Z):	28
6. DECF f, d (Z):.....	28
7. DECFSZ f, d:	29
8. INCF f, d (Z):.....	29

9.	INCFSZ f, d:	30
10.	IORWF f, d (Z):	31
11.	MOVF f, d (Z):	32
12.	MOVWF f:	32
13.	NOP:	32
14.	RLF f, d (C):	32
15.	RRF f, d (C):	33
16.	SUBWF f, d (C, DC, Z):	33
17.	SWAPF f, d:	34
18.	XORWF f, d (Z):	34
19.	BCF f, b:	35
20.	BSF f, b:	36
21.	BTFSC f, b:	37
22.	BTFSS f, b:	38
23.	ADDLW k (C, DC, Z):	39
24.	ANDLW k (Z):	39
25.	CALL k:	39
26.	CLRWD ^T :	42
27.	GOTO k:	42
28.	IORLW k (Z):	44
29.	MOVLW k:	44
30.	RETFIE :	45
31.	RETLW k:	45
32.	RETURN :	45
33.	SLEEP :	45
34.	SUBLW k (C, DC, Z):	46
35.	XORLW k (Z):	46
Directivas:	47
1.	PROCESSOR NOMBRE_DEL_PIC:	47
2.	__CONFIG PALABRA DE CONFIGURACIÓN:	47
3.	SIGNIFICADO DE LOS 14 BITS EN LA PALABRA DE CONFIGURACIÓN 1	49
•	IESO (bit 10 de la palabra de configuración) Divisor de Reloj	49
•	FOSC (bits 2, 1 y 0 de la palabra de configuración) Tipo de Oscilador	53

4. SIGNIFICADO DE LOS 14 BITS EN LA PALABRA DE CONFIGURACIÓN 2	57
5. INCLUDE <DIRECCIÓN Y NOMBRE DEL ARCHIVO CON SU EXTENSIÓN>:.....	59
6. ORG DIRECCIÓN HEXADECIMAL DE 4 DÍGITOS:	60
7. NOMBRE_A_DECLARAR EQU REGISTRO_ASIGNADO:.....	60
8. CBLOCK REGISTRO INICIAL PALABRA1, ..., PALABRA_N:	60
9. END:	61
Subrutinas.....	61
25. CALL k:.....	62
32. RETURN :	62
Subrutinas de Tiempo	64
Función Antirrebote.....	64
Subrutina de Tiempo: Ciclo de Máquina.....	65
Subrutina de Tiempo de 1 Variable	66
○ MOVLW k:.....	66
○ MOVWF f:.....	66
○ CALL k:	66
■ NOP:	67
■ DECFSZ f, d:	67
■ GOTO k:	67
■ RETURN :	68
Subrutina de tiempo de 2 variables	70
Subrutina de tiempo de 3 variables	71
Interrupciones	71
Tipos de Interrupciones	71
Proceso de una Interrupción en el PIC16F887.....	72
Interrupción por la terminal RB0/INT	73

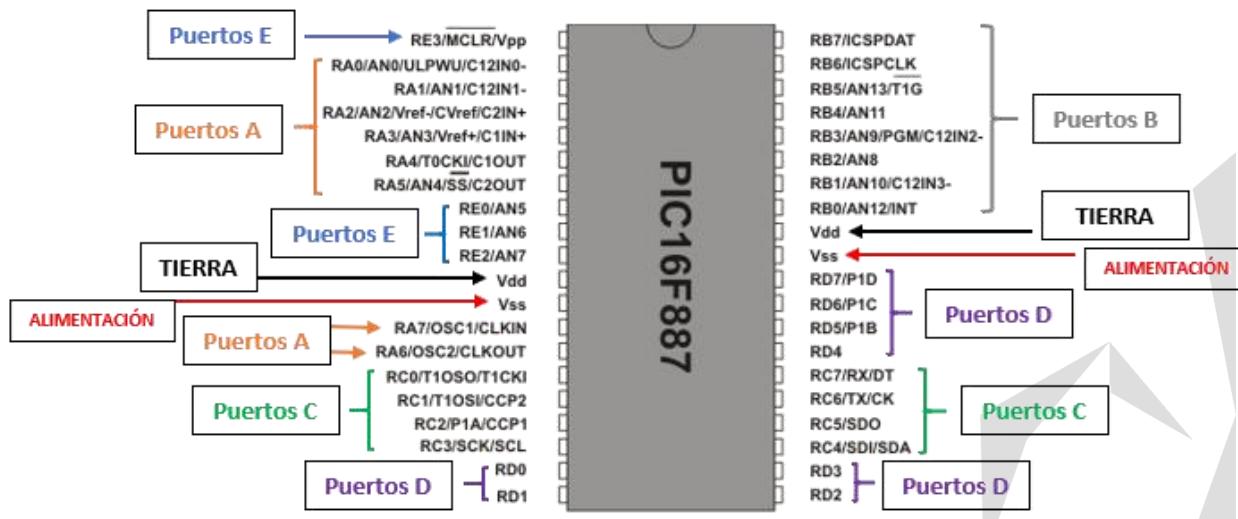


Funcionamiento PIC

El PIC es un microcontrolador y se diferencia de un microprocesador porque éste ya incluye una memoria RAM, una memoria ROM (Read Only Memory o Pila), un propio microprocesador y un bus de datos internamente. Mientras que al microprocesador se le deben conectar todos estos elementos externamente.

Requerimientos Básicos Funcionamiento Microcontrolador:

- **Su alimentación:** Se alimenta de 2 a 5 V, si me paso de 5.5 V se dañarán los módulos internos del dispositivo.
- **Un oscilador:** Que puede ser el interno que existe dentro del PIC o uno externo que construiremos en una protoboard, dependiendo de las necesidades a cubrir por el microcontrolador y sus condiciones de trabajo.
- **Memorias:** Dentro de las cuales existen muchos tipos de registros que nos servirán para ejecutar todas las acciones necesarias en el PIC16F887.
 - **Memoria de datos (RAM):** Esta es la memoria de corto plazo RAM (Random Access Memory) que contiene todo tipo de registros (512 en total) para el almacenamiento de datos que se borran al remover la alimentación del microcontrolador. Donde además se usan el siguiente número de bits para indicar las direcciones de los registros de la RAM que quiero usar y completamente aparte se usan otro número de bits para enviar los datos que requiera.
 - 9 bits de direccionamiento.
 - 8 bits de datos.
 - **Memoria de instrucciones (FLASH):** En los registros de esta memoria es donde se queman las instrucciones del programa que creemos con lenguaje ensamblador, y en el caso del PIC16F887 la memoria de instrucciones es una memoria FLASH. Donde además se usan el siguiente número de bits para indicar las direcciones de los registros de la memoria FLASH que queremos usar y completamente aparte se usan otro número de bits para enviar los datos que quiera.
 - 13 bits de direccionamiento.
 - 14 bits de datos.



Los puertos del PIC entregan 5V y 15mA por pin, pero no puedo usar todos los pines a la vez ya que demandaría mucha potencia que no puede entregar el microcontrolador. Los registros (que son vectores de varios bits) los deberé declarar de forma hexadecimal, esto se hace poniendo: **0Xnúmero_hexadecimal**.

Registros Microcontrolador:

- **Registro acumulador:** En el microcontrolador PIC16F887 se tiene un solo registro acumulador llamado W, aquí son almacenados temporalmente los resultados aritméticos y lógicos que después serán tratados. Estos valores luego deben ser cargados a los registros de la memoria de datos RAM (Random Access Memory) o a los puertos del PIC, por lo que **se considera como un paso intermedio de almacenamiento**.
 - El registro acumulador almacena los resultados de las operaciones ejecutadas por el circuito operacional y se encuentra conectado con el bus de datos del sistema con el propósito de enviar los resultados a la memoria o (RAM) o a algún periférico.
- **Registros de propósito general:** En el PIC16F887 se tiene un tipo de registro que funciona como RAM para almacenar y consultar datos temporalmente (ya que estos datos se borrarán solo cuando yo los borre manualmente o cuando apague el micro), estos registros me pueden dar información del estado actual de los periféricos del micro (osea los pines o puertos de entrada o salida, el convertidor analógico digital, temporizadores, comunicación serial, comparadores analógicos, etc.) o de la ALU (Unidad Lógico Aritmética) que realiza sumas, restas y operaciones lógicas (AND, OR, NOT, etc.).
 - Los registros de propósito general son los encargados de almacenar temporalmente los datos con los que el procesador realice sus operaciones cuando no se quiera usar el acumulador W y de guardar los resultados de dichas operaciones. También se utilizan para guardar direcciones de memoria y en el PIC abarcan un número de registros específicos dependiendo del banco donde nos encontramos.

FIGURE 2-4: PIC16F886/PIC16F887 SPECIAL FUNCTION REGISTERS				
File Address	File Address	File Address	File Address	
Indirect addr. ⁽¹⁾ 00h	Indirect addr. ⁽¹⁾ 80h	Indirect addr. ⁽¹⁾ 100h	Indirect addr. ⁽¹⁾ 180h	
TMR0 01h	OPTION_REG 81h	TMR0 101h	OPTION_REG 181h	
PCL 02h	PCL 82h	PCL 102h	PCL 182h	
STATUS 03h	STATUS 83h	STATUS 103h	STATUS 183h	
FSR 04h	FSR 84h	FSR 104h	FSR 184h	
PORTA 05h	TRISA 85h	PORTB 105h	SRCON 185h	
PORTB 06h	TRISB 86h	PORTB 106h	TRISB 186h	
PORTC 07h	TRISC 87h	CM1CON0 107h	BALIODET 187h	
PORTD ⁽²⁾ 08h	TRISD ⁽²⁾ 88h	CM2CON0 108h	ANSEL 188h	
PORTE 09h	TRISE 89h	CM2CON1 109h	ANSELEH 189h	
PCLATH 0Ah	PCLATH 8Ah	PCLATH 10Ah	PCLATH 18Ah	
INTCON 0Bh	INTCON 8Bh	INTCON 10Bh	INTCON 18Bh	
PIR1 0Ch	PIE1 8Ch	EEDAT 10Ch	EECON1 18Ch	
PIR2 0Dh	PIE2 8Dh	EEADR 10Dh	EECON2 ⁽³⁾ 18Dh	
PIR3 0Eh	PIE3 8Eh	EEADTH 10Eh	Reserved 18Eh	
TMR1L 0Fh	SSCON 8Fh	EEADRH 10Fh	Reserved 18Fh	
T1CON 10h	DSTUNE 90h	General Purpose Registers 16 Bytes		
TMR2 11h	SSPCON2 91h	110h	190h	
T2CON 12h	PR2 92h	111h	191h	
SSPBUF 13h	SSPADD 93h	112h	192h	
SSPCON 14h	SSPSTAT 94h	113h	193h	
CCPR1L 15h	WPUB 95h	114h	194h	
CCPR1H 16h	ICCB 96h	115h	195h	
CCP1CON 17h	VRCON 97h	116h	196h	
RSTA 18h	TXSTA 98h	117h	197h	
TXREG 19h	SPBRG 99h	118h	198h	
RCREG 1Ah	SPBRGH 9Ah	119h	199h	
CCPR2H 1Ch	PVMICON 9Bh	11Ah	19Ah	
CCPR2L 1Dh	ECOPAS 9Ch	11Bh	19Bh	
CCP2CON 1Eh	PSTRCON 9Dh	11Ch	19Ch	
ADRESH 1Fh	ADRESL 9Eh	11Dh	19Dh	
ADCON0 20h	ADCON1 9Fh	11Eh	19Eh	
General Purpose Registers 96 Bytes		11Fh	19Fh	
3Fh	General Purpose Registers 80 Bytes		120h	1A0h
40h	EFh	16Fh	16Fh	
4Fh	F0h	170h	1F0h	
5Fh	FFh	17Fh	1FFh	
60h	General Purpose Registers 80 Bytes		61h	
6Fh	accesses 70h-7Fh	62h	accesses 70h-7Fh	
70h		63h		
7Fh		64h		
		65h		
		66h		
		67h		
		68h		
		69h		
		6Ah		
		6Bh		
		6Ch		
		6Dh		
		6Eh		
		6Fh		
		70h		
		71h		
		72h		
		73h		
		74h		
		75h		
		76h		
		77h		
		78h		
		79h		
		7Ah		
		7Bh		
		7Ch		
		7Dh		
		7Eh		
		7Fh		
		80h		
		81h		
		82h		
		83h		
		84h		
		85h		
		86h		
		87h		
		88h		
		89h		
		8Ah		
		8Bh		
		8Ch		
		8Dh		
		8Eh		
		8Fh		
		90h		
		91h		
		92h		
		93h		
		94h		
		95h		
		96h		
		97h		
		98h		
		99h		
		9Ah		
		9Bh		
		9Ch		
		9Dh		
		9Eh		
		9Fh		
		100h		
		101h		
		102h		
		103h		
		104h		
		105h		
		106h		
		107h		
		108h		
		109h		
		10Ah		
		10Bh		
		10Ch		
		10Dh		
		10Eh		
		10Fh		
		110h		
		111h		
		112h		
		113h		
		114h		
		115h		
		116h		
		117h		
		118h		
		119h		
		11Ah		
		11Bh		
		11Ch		
		11Dh		
		11Eh		
		11Fh		
		120h		



En el PIC16F887 los registros de propósito general de cada banco son los siguientes, indicado en la página 25 del datasheet:

- **Registros de propósito general Banco 0:** Desde la dirección de memoria RAM 20h, 0X20 o 20 hexadecimal (que es lo mismo) hasta la 7F hexadecimal.
 - **20h** - hasta la - **7Fh** (hexadecimal).
 - **Registros de propósito general Banco 1:** Desde la dirección de memoria RAM A0h, 0XA0 u A0 hexadecimal (que es lo mismo) hasta la FF hexadecimal.
 - **A0h** - hasta la - **FFh** (hexadecimal).
 - **Registros de propósito general Banco 2:** Desde la dirección de memoria RAM 110h, 0X110 u 110 hexadecimal (que es lo mismo) hasta la 17F hexadecimal.
 - **110h** - hasta la - **17Fh** (hexadecimal).
 - **Registros de propósito general Banco 3:** Desde la dirección de memoria RAM 190h, 0X190 u 190 hexadecimal (que es lo mismo) hasta la 1FF hexadecimal.
 - **190h** - hasta la - **1FFh** (hexadecimal).
-
- **Registro Brown-Out:** Registro que indica un nivel de tensión mínimo, si la tensión cae debajo de la configurada, se reinicia el microcontrolador, esa tensión puede ser de 4 o 2.1V.
 - **Banco o página de registros:** Con banco o página nos referimos al conjunto de registros que están agrupados en una parte de la memoria RAM, en el PIC16F887 contamos con 4 bancos y debemos movernos de uno a otro dependiendo del registro que queramos alcanzar cambiando el estado de los bits RP1 y RP0 del registro STATUS, a esto se le llama paginación:
 - Banco 0: Contiene la dirección del registro de todos los puertos A, B, C, D y E del PIC16F887.
 - Banco 1: Contiene el registro OSCCON que sirve para dar diferentes frecuencias al oscilador interno y los registros TRISA, TRISB, TRISC, TRISD y TRISE que sirven para hacer que los puertos sean entradas o salidas.
 - Banco 2: Contiene solo la dirección del puerto B.
 - Banco 3: Contiene el registro TRISB que sirve para hacer los pines del puerto B entradas o salidas.

Usando alguna de las 35 instrucciones del PIC debo cambiar el estado de los bits RP1 y RP0 del registro STATUS para poder cambiar de banco en la memoria RAM y así poder accesar a cualquiera de los registros que quiera.

Al registro STATUS puedo acceder desde cualquiera de los bancos de la memoria RAM en el PIC16F887 usando las siguientes direcciones hexadecimales de memoria:

- **Registro STATUS Banco 0:** Dirección de memoria RAM **03h**, 0X03 o 03 hexadecimal que es lo mismo.
- **Registro STATUS Banco 1:** Dirección de memoria RAM **83h**, 0X83 u 83 hexadecimal que es lo mismo.
- **Registro STATUS Banco 2:** Dirección de memoria RAM **103h**, 0X103 o 103 hexadecimal que es lo mismo.
- **Registro STATUS Banco 3:** Dirección de memoria RAM **183h**, 0X183 o 183 hexadecimal que es lo mismo.

Cuando acceda al registro STATUS debo cambiar específicamente los bits RP1 y RP0 para que con la siguiente combinación cambie de un banco a otro (al iniciar o reiniciar el PIC siempre me encuentro en el banco 0). Esto está indicado en la página 22 del datasheet.

RP1 RP0

- 0 0 → Bank 0 is selected
- 0 1 → Bank 1 is selected
- 1 0 → Bank 2 is selected
- 1 1 → Bank 3 is selected

PIC16F882/883/884/886/887

FIGURE 2-6: PIC16F886/PIC16F887 SPECIAL FUNCTION REGISTERS

File Address	File Address	File Address	File Address		
Indirect addr. ⁽¹⁾ 00h	Indirect addr. ⁽¹⁾ 00h	Indirect addr. ⁽¹⁾ 00h	Indirect addr. ⁽¹⁾ 180h		
TMR0 01h	OPTION_REG 01h	TMR0 01h	OPTION_REG 181h		
PCL 02h	PCL 02h	PCL 02h	PCL 182h		
STATUS 03h	STATUS 03h	STATUS 03h	STATUS 183h		
FSR 04h	FSR 04h	FSR 04h	FSR 184h		
PORTA 05h	TRISA 05h	WDTCON 05h	SRCON 185h		
PORTB 06h	TRISB 06h	PORTB 06h	TRISB 186h		
PORTC 07h	TRISC 07h	CM1CON0 07h	BAUDCTL 187h		
PORTD ⁽²⁾ 08h	TRISD ⁽²⁾ 08h	CM2CON0 08h	ANSEL 188h		
PORTE 09h	TRISE 09h	CM2CON1 09h	ANSELH 189h		
PCLATH 0Ah	PCLATH 0Ah	PCLATH 0Ah	PCLATH 18Ah		
INTCON 0Bh	INTCON 0Bh	INTCON 0Bh	INTCON 18Bh		
PIR1 0Ch	PIE1 0Ch	EEDAT 0Ch	EECON1 18Ch		
PIR2 0Dh	PIE2 0Dh	EEADR 0Dh	EECON2 ⁽¹⁾ 18Dh		
TMR1L 0Eh	PCON 0Eh	EEDATH 0Eh	Reserved 18Eh		
TMR1H 0Fh	OSCCON 0Fh	EEADRH 0Fh	Reserved 18Fh		
T1CON 10h	OSCTUNE 90h		190h		
TMR2 11h	SSPCON2 91h		191h		
T2CON 12h	PR2 92h		192h		
SSPBUF 13h	SSPADD 93h		193h		
SSPCON 14h	SSPSTAT 94h		194h		
CCPR1L 15h	WPUB 95h		195h		
CCPR1H 16h	IOCB 96h	General Purpose Registers	196h		
CCP1CON 17h	VRCON 97h		197h		
RCSTA 18h	TXSTA 98h		198h		
TXREG 19h	SPBRG 99h	16 Bytes	199h		
RCREG 1Ah	SPBRGH 9Ah		19Ah		
CCPR2L 1Bh	PWM1CON 9Bh		19Bh		
CCPR2H 1Ch	ECCPAS 9Ch		19Ch		
CCP2CON 1Dh	PSTRCON 9Dh		19Dh		
ADRESH 1Eh	ADRESL 9Eh		19Eh		
ADCNO 1Fh	ADCON1 9Fh		19Fh		
Bank 0		Bank 1			
General Purpose Registers		General Purpose Registers			
96 Bytes		80 Bytes			
3Fh		80 Bytes			
40h		80 Bytes			
6Fh		16Fh			
70h		170h			
7Fh		17Fh			
accesses 70h-7Fh		accesses 70h-7Fh			
EFh		accesses 70h-7Fh			
F0h		16Fh			
FFh		170h			
Bank 2		Bank 3			
General Purpose Registers		General Purpose Registers			
80 Bytes		80 Bytes			
EFh		16Fh			
F0h		170h			
FFh		17Fh			
Bank 2		Bank 3			
Unimplemented data memory locations, read as '0'.					
Note 1: Not a physical register.					
2: PIC16F887 only.					

- Registro de propósito específico: Son los registros de la RAM que tienen un nombre propio, sirven para configurar el microcontrolador, recibir información de resultados o eventos generados por el procesamiento de datos o por eventos acontecidos dentro o fuera del dispositivo y se puede observar cuál es el nombre y dirección de cada uno en la tabla anterior, un ejemplo de estos es el registro STATUS o los Puertos A, B, C, D o E. Esto lo encuentro en la página 25 del datasheet.

- Son registros que se utilizan para una tarea determinada y cumplen tareas específicas como contador, registros de direcciones de memoria, registro de instrucciones y registro de memoria intermedia.

Hay algunos bits de los registros de propósito específico que no se pueden acceder y otros que sí, esto viene indicado desde la página 29 hasta la 36 del datasheet para cada registro individualmente, especificando si los bits del registro se pueden acceder o no:

- **R = Read (leible):** Estos bits no los puedo cambiar, solo leer.
- **W = Write (escribible):** Estos bits si los puedo cambiar (o escribir que es lo mismo).
- **R/W = Read/Write (leible/escribible):** Estos bits los puedo leer y escribir.

Además de lo anterior, en la columna Value on POR, BOR podemos visualizar el valor inicial que adopta cada bit del registro después de un reset, x significa que no sabemos qué valor adoptará el bit después de un reinicio del microcontrolador.

BANCO 0:

TABLE 2-1: PIC16F882/883/884/886/887 SPECIAL FUNCTION REGISTERS SUMMARY BANK 0

Addr	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on POR, BOR	Page
Bank 0											
00h	INDF	Addressing this location uses contents of FSR to address data memory (not a physical register)							xxxxx xxxx	37,213	
01h	TMR0	Timer0 Module Register							xxxxx xxxx	73,213	
02h	PCL	Program Counter's (PC) Least Significant Byte							0000 0000	37,213	
03h	STATUS	IRP	RP1	RP0	TO	PD	Z	DC	Q	0001 1xxxx	29,213
04h	FSR	Indirect Data Memory Address Pointer							xxxxx xxxx	37,213	
05h	PORTA ⁽³⁾	RA7	RA6	RA5	RA4	RA3	RA2	RA1	RA0	xxxxx xxxx	39,213
06h	PORTB ⁽³⁾	RB7	RB6	RB5	RB4	RB3	RB2	RB1	RB0	xxxxx xxxx	48,213
07h	PORTC ⁽³⁾	RC7	RC6	RC5	RC4	RC3	RC2	RC1	RC0	xxxxx xxxx	53,213
08h	PORTD ^(3,4)	RD7	RD6	RD5	RD4	RD3	RD2	RD1	RD0	xxxxx xxxx	57,213
09h	PORTE ⁽³⁾	—	—	—	RE3	RE2 ⁽⁴⁾	RE1 ⁽⁴⁾	RE0 ⁽⁴⁾	—	---- 0xxxx	59,213
0Ah	PCLATH	—	—	—	Write Buffer for upper 5 bits of Program Counter						
0Bh	INTCON	GIE	PEIE	TOIE	INTE	RBIE	TOIF	INTF	RBIF ⁽¹⁾	0000 000x	31,213
0Ch	PIR1	—	ADIF	RCIF	TXIF	SSPIF	CCP1IF	TMR2IF	TMR1IF	-000 0000	34,213
0Dh	PIR2	OSFIF	C2IF	C1IF	EEIF	BCLIF	ULPWUIF	—	CCP2IF	0000 00-0	35,213
0Eh	TMR1L	Holding Register for the Least Significant Byte of the 16-bit TMR1 Register							xxxxx xxxx	76,213	
0Fh	TMR1H	Holding Register for the Most Significant Byte of the 16-bit TMR1 Register							xxxxx xxxx	76,213	
10h	T1CON	T1GINV	TMR1GE	T1CKPS1	T1CKPS0	T1OSCEN	T1SYNC	TMR1CS	TMR1ON	0000 0000	79,213
11h	TMR2	Timer2 Module Register							0000 0000	81,213	
12h	T2CON	—	TOUTPS3	TOUTPS2	TOUTPS1	TOUTPS0	TMR2ON	T2CKPS1	T2CKPS0	-000 0000	82,213
13h	SSPBUF	Synchronous Serial Port Receive Buffer/Transmit Register							xxxxx xxxx	179,213	
14h	SSPCON ⁽²⁾	WCOL	SSPOV	SSPEN	CKP	SSPM3	SSPM2	SSPM1	SSPM0	0000 0000	177,213
15h	CCPR1L	Capture/Compare/PWM Register 1 Low Byte (LSB)							xxxxx xxxx	126,213	
16h	CCPR1H	Capture/Compare/PWM Register 1 High Byte (MSB)							xxxxx xxxx	126,213	
17h	CCP1CON	P1M1	P1M0	DC1B1	DC1B0	CCP1M3	CCP1M2	CCP1M1	CCP1M0	0000 0000	124,213
18h	RCSTA	SPEN	RX9	SREN	CREN	ADDEN	FERR	OERR	RX9D	0000 000x	159,213
19h	TXREG	EUSART Transmit Data Register							0000 0000	151,213	
1Ah	RCREG	EUSART Receive Data Register							0000 0000	156,213	
1Bh	CCPR2L	Capture/Compare/PWM Register 2 Low Byte (LSB)							xxxxx xxxx	126,213	
1Ch	CCPR2H	Capture/Compare/PWM Register 2 High Byte (MSB)							xxxxx xxxx	126,214	
1Dh	CCP2CON	—	—	DC2B1	DC2B0	CCP2M3	CCP2M2	CCP2M1	CCP2M0	-000 0000	125,214
1Eh	ADRESH	A/D Result Register High Byte							xxxxx xxxx	99,214	
1Fh	ADCON0	ADC1	ADCS0	CHS3	CHS2	CHS1	CHS0	GO/DONE	ADON	0000 0000	104,214

BANCO 1:

TABLE 2-2: PIC16F882/883/884/886/887 SPECIAL FUNCTION REGISTERS SUMMARY BANK 1

Addr	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on POR, BOR	Page
Bank 1											
80h	INDF	Addressing this location uses contents of FSR to address data memory (not a physical register)							xxxx xxxx	37,213	
81h	OPTION_REG	RBPU	INTEDG	TOCS	TOSE	PSA	PS2	PS1	PS0	1111 1111	30,214
82h	PCL	Program Counter's (PC) Least Significant Byte							0000 0000	37,213	
83h	STATUS	IRP	RP1	RP0	TO	PD	Z	DC	C	0001 1xxx	29,213
84h	FSR	Indirect Data Memory Address Pointer							xxxx xxxx	37,213	
85h	TRISA	TRISA7	TRISA6	TRISA5	TRISA4	TRISA3	TRISA2	TRISA1	TRISA0	1111 1111	39,214
86h	TRISB	TRISB7	TRISB6	TRISB5	TRISB4	TRISB3	TRISB2	TRISB1	TRISB0	1111 1111	48,214
87h	TRISC	TRISC7	TRISC6	TRISC5	TRISC4	TRISC3	TRISC2	TRISC1	TRISC0	1111 1111	53,214
88h	TRISD ⁽³⁾	TRISD7	TRISD6	TRISD5	TRISD4	TRISD3	TRISD2	TRISD1	TRISD0	1111 1111	57,214
89h	TRISE	—	—	—	—	TRISE3	TRISE2 ⁽³⁾	TRISE1 ⁽³⁾	TRISE0 ⁽³⁾	— 1111	59,214
8Ah	PCLATH	—	—	—	Write Buffer for the upper 5 bits of the Program Counter					---0 0000	37,213
8Bh	INTCON	GIE	PEIE	TOIE	INTE	RBIE	TOIF	INTF	RBIF ⁽¹⁾	0000 000x	31,213
8Ch	PIE1	—	ADIE	RCIE	TXIE	SSPIE	CCP1IE	TMR2IE	TMR1IE	~000 0000	32,214
8Dh	PIE2	OSFIE	C2IE	C1IE	EEIE	BCLIE	ULPWUIE	—	CCP2IE	0000 00-0	33,214
8Eh	PCON	—	—	ULPWUE	SBOREN	—	—	POR	BOR	~01 ~qq	36,214
8Fh	OSCCON	—	IRCF2	IRCF1	IRCF0	OSTS	HTS	LTS	SCS	~10 q000	62,214
90h	OSCTUNE	—	—	—	TUN4	TUN3	TUN2	TUN1	TUN0	---0 0000	66,214
91h	SSPCON2	GCEN	ACKSTAT	ACKDT	ACKEN	RCEN	PEN	RSEN	SEN	0000 0000	177,214
92h	PR2	Timer2 Period Register							1111 1111	81,214	
93h	SSPADD ⁽²⁾	Synchronous Serial Port (I ² C mode) Address Register							0000 0000	185,214	
93h	SSPMASK ⁽²⁾	MSK7	MSK6	MSK5	MSK4	MSK3	MSK2	MSK1	MSK0	1111 1111	204,214
94h	SSPSTAT	SMP	CKE	D/A	P	S	R/W	UA	BF	0000 0000	185,214
95h	WPUB	WPUB7	WPUB6	WPUB5	WPUB4	WPUB3	WPUB2	WPUB1	WPUB0	1111 1111	49,214
96h	IOCB	IOCB7	IOCB6	IOCB5	IOCB4	IOCB3	IOCB2	IOCB1	IOCB0	0000 0000	49,214
97h	VRCN	VREN	VRQE	VRSS	VR3	VR2	VR1	VR0	0000 0000		97,214
98h	TXSTA	CSRC	TX9	TXEN	SYNC	SENDB	BRGH	TRMT	TX9D	0000 0010	158,214
99h	SPBRG	BRG7	BRG6	BRG5	BRG4	BRG3	BRG2	BRG1	BRG0	0000 0000	161,214
9Ah	SPBRGH	BRG15	BRG14	BRG13	BRG12	BRG11	BRG10	BRG9	BRG8	0000 0000	161,214
9Bh	PWM1CON	PRSEN	PDC6	PDC5	PDC4	PDC3	PDC2	PDC1	PDC0	0000 0000	144,214
9Ch	ECCPAS	ECCPASE	ECCPAS2	ECCPAS1	ECCPAS0	PSSAC1	PSSAC0	PSSBD1	PSSBD0	0000 0000	141,214
9Dh	PSTRCON	—	—	—	STRSYNC	STRD	STRC	STRB	STRA	---0 0001	145,214
9Eh	ADRESL	A/D Result Register Low Byte							xxxx xxxx	99,214	
9Fh	ADCON1	ADFM	—	VCFG1	VCFG0	—	—	—	—	0-00 ----	105,214

BANCO 2:

TABLE 2-3: PIC16F882/883/884/886/887 SPECIAL FUNCTION REGISTERS SUMMARY BANK 2

Addr	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on POR, BOR	Page
Bank 2											
100h	INDF	Addressing this location uses contents of FSR to address data memory (not a physical register)							xxxx xxxx	37,213	
101h	TMR0	Timer0 Module Register							xxxx xxxx	73,213	
102h	PCL	Program Counter's (PC) Least Significant Byte							0000 0000	37,213	
103h	STATUS	IRP	RP1	RP0	TO	PD	Z	DC	C	0001 1xxx	29,213
104h	FSR	Indirect Data Memory Address Pointer							xxxx xxxx	37,213	
105h	WDTCON	—	—	—	WDTPS3	WDTPS2	WDTPS1	WDTPS0	SWDTEN	---0 1000	221,214
106h	PORTB	RB7	RB6	RB5	RB4	RB3	RB2	RB1	RB0	xxxx xxxx	48,213
107h	CM1CON0	C1ON	C1OUT	C1OE	C1POL	—	C1R	C1CH1	C1CHO	0000 -000	88,214
108h	CM2CON0	C2ON	C2OUT	C2OE	C2POL	—	C2R	C2CH1	C2CHO	0000 -000	89,214
109h	CM2CON1	MC1OUT	MC2OUT	C1RSEL	C2RSEL	—	—	T1GSS	C2SYNC	0000 --10	91,215
10Ah	PCLATH	—	—	—	Write Buffer for the upper 5 bits of the Program Counter					---0 0000	37,213
10Bh	INTCON	GIE	PEIE	TOIE	INTE	RBIE	TOIF	INTF	RBIF ⁽¹⁾	0000 000x	31,213
10Ch	EEDAT	EEDAT7	EEDAT6	EEDAT5	EEDAT4	EEDAT3	EEDAT2	EEDAT1	EEDAT0	0000 0000	112,215
10Dh	EEADR	EEADR7	EEADR6	EEADR5	EEADR4	EEADR3	EEADR2	EEADR1	EEADR0	0000 0000	112,215
10Eh	EEDATH	—	—	EEDATH5	EEDATH4	EEDATH3	EEDATH2	EEDATH1	EEDATH0	-00 0000	112,215
10Fh	EEADRH	—	—	—	EEADRH4 ⁽²⁾	EEADR3	EEADR2	EEADR1	EEADR0	-+0 0000	112,215

BANCO 3:

TABLE 2-4: PIC16F882/883/884/886/887 SPECIAL FUNCTION REGISTERS SUMMARY BANK 3

Addr	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on POR, BOR	Page
Bank 3											
180h	INDF									xxxx xxxx	37,213
181h	OPTION_REG	RBPU	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0	1111 1111	30,214
182h	PCL									0000 0000	37,213
183h	STATUS	IRP	RP1	RP0	TO	PD	Z	DC	C	0001 1xxx	29,213
184h	FSR									xxxx xxxx	37,213
185h	SRCON	SR1	SR0	C1SEN	C2REN	PULSS	PULSR	—	FVREN	0000 00-0	93,215
186h	TRISB	TRISB7	TRISB6	TRISB5	TRISB4	TRISB3	TRISB2	TRISB1	TRISB0	1111 1111	48,214
187h	BAUDCTL	ABDOVF	RCIDL	—	SCKP	BRG16	—	WUE	ABDEN	01-0 0-00	160,215
188h	ANSEL	ANS7 ⁽²⁾	ANS6 ⁽²⁾	ANS5 ⁽²⁾	ANS4	ANS3	ANS2	ANS1	ANS0	1111 1111	40,215
189h	ANSELH	—	—	ANS13	ANS12	ANS11	ANS10	ANS9	ANS8	-11 1111	99,215
18Ah	PCLATH	—	—	—						---0 0000	37,213
18Bh	INTCON	GIE	PEIE	TOIE	INTE	RBIIE	TOIF	INTF	RBIFF ⁽¹⁾	0000 000x	31,213
18Ch	EECON1	EEPGD	—	—	—	WRERR	WREN	WR	RD	x--- x000	113,215
18Dh	EECON2									-----	111,215

Para ello además se toman en cuenta algunos conceptos usados en las columnas Bit7, Bit6, Bit5, Bit4, Bit3, Bit2, Bit1 y Bit0 para describir que hace cada uno.

TABLE 15-1: OPCODE FIELD DESCRIPTIONS

Field	Description
f	Register file address (0x00 to 0x7F)
W	Working register (accumulator)
b	Bit address within an 8-bit file register
k	Literal field, constant data or label
x	Don't care location (= 0 or 1). The assembler will generate code with x = 0. It is the recommended form of use for compatibility with all Microchip software tools.
d	Destination select; d = 0: store result in W, d = 1: store result in file register f. Default is d = 1 .
PC	Program Counter
TO	Time-out bit
C	Carry bit
DC	Digit carry bit
Z	Zero bit
PD	Power-down bit

- **Registro de estado (STATUS) o chismoso:** Además de para cambiar de banco en la memoria RAM, el registro STATUS es un registro con banderas que dejan constancia de algunas condiciones que se dieron en la última operación realizada, son los que toman en cuenta acarreos en la suma o resta binaria que podrán ser tomados en cuenta en operaciones posteriores.
 - Se conoce como registro de estado (STATUS) a los registros de memoria (se dice registros en plural porque existe 1 registro de este tipo en cada banco o página de la memoria

- RAM) en los que se deja constancia de algunas condiciones que se dieron en la última operación realizada y que podrán ser utilizadas en operaciones posteriores.
- Los bits del registro de STATUS que me permiten pasar de un banco de registros a otro son:
 - RP1.**
 - RP0.**

- Además, como el registro STATUS es de propósito específico, hay algunos de sus bits que no podemos acceder, solo leer, esto viene indicado en la página 29 del datasheet para el registro STATUS específicamente, aunque también viene la misma descripción bit por bit de todos los demás registros de este tipo posteriormente, donde:
 - R = Read (leible):** Estos bits no los puedo cambiar, solo leer.
 - W = Write (escribible):** Estos bits si los puedo cambiar (o escribir que es lo mismo) pero no leer.
 - R/W = Read/Write (leible/escribible):** Estos bits los puedo leer y escribir.
 - Alado de esta primera descripción de bits hay un guion y un 0, 1 o una x, esto nos indica después de un reset del PIC con qué valor se inicializa cada uno de los bits del registro de propósito específico.
 - 0:** El bit del registro específico después de un reset adopta el valor cero.
 - 1:** El bit del registro específico después de un reset adopta el valor uno.
 - x:** El bit del registro específico después de un reset no sabemos qué valor va a adoptar.

REGISTER 2-1: STATUS: STATUS REGISTER								
R/W-0	R/W-0	R/W-0	R-1	R-1	R/W-x	R/W-x	R/W-x	
IRP	RP1	RP0	\overline{TO}	\overline{PD}	Z	DC ⁽¹⁾	C ⁽¹⁾	
bit 7								bit 0

Legend:

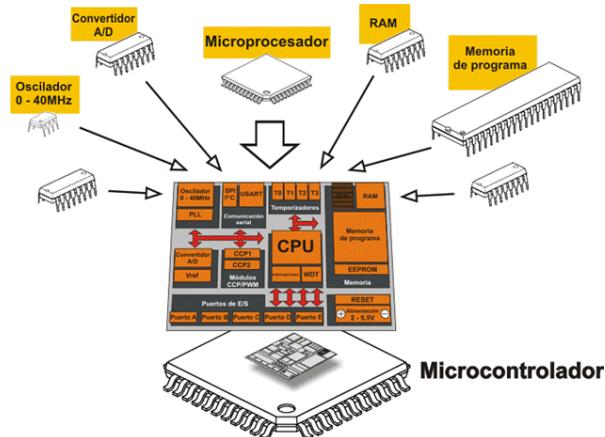
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'
-n = Value at POR	'1' = Bit is set	'0' = Bit is cleared
		x = Bit is unknown

Como siempre después de un reset o al prender el PIC RP0 = 0 y RP1 = 0, al iniciar me encuentro en el banco 0.

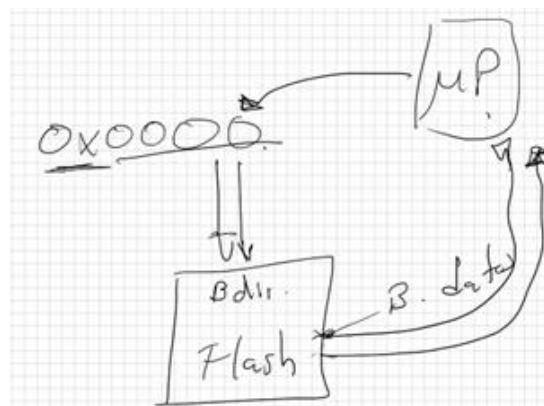
- Bandera:** Es un bit que entrega información de resultados o eventos generados por el procesamiento o por eventos acontecidos dentro o fuera del dispositivo, estas son afectadas cuando usemos algunas de las 35 instrucciones del PIC.
 - Se cuenta con 8 bits para hacer operaciones aritméticas o lógicas, los primeros 4 se consideran como un solo digito hexadecimal y los otros 4 se consideran como otro digito hexadecimal.
 - C o acarreo:** Esta bandera indica cuando el resultado total de una suma binaria tuvo un 1 de más que no cupo en los 8 bits que abarca el PIC para hacer operaciones binarias.
 - C = 1 cuando ocurre un acarreo.
 - C = 0 cuando NO ocurre un acarreo.
 - DC o acarreo decimal:** Indica cuando después de aplicar una suma o resta binaria, un acarreo pasa de los 4 primeros bits (los de la izquierda) a los segundos 4 bits

(los de la derecha) del número binario de 8 bits utilizado para hacer operaciones aritméticas.

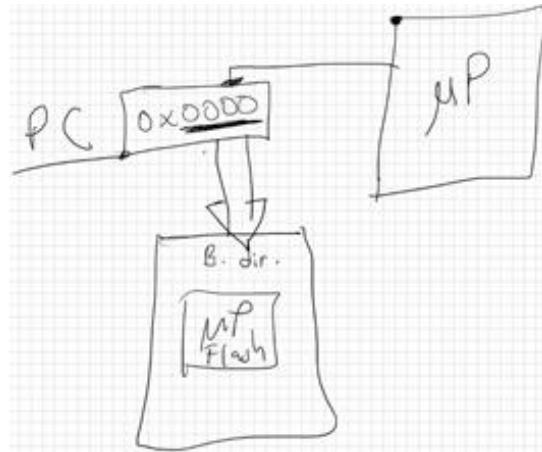
- DC = 1 cuando ocurre un acarreo decimal.
- DC = 0 cuando NO ocurre un acarreo decimal.
- **Z o ceros:** Bandera que indica cuando el resultado es totalmente cero.
 - Z = 1 cuando el resultado es totalmente cero.
 - Z = 0 cuando el resultado NO es totalmente cero.
- **Contador de programa o Program Counter (PC):** Es un registro de memoria de corto plazo que contiene la dirección de la memoria del programa (memoria FLASH) de la siguiente instrucción a ejecutar, es un contador interno que empieza desde cero después de un reset aplicado al PIC 16F887 y se incrementa de 1 en 1 siguiendo el paso de la señal de RELOJ o CLK para saber qué instrucción le toca ejecutar, es de las partes más importantes del PIC.
 - El contador de programa (en inglés Program Counter o PC), también llamado Puntero de instrucciones (Instruction Pointer), es un registro del computador que indica la posición donde está el procesador en su secuencia de instrucciones.



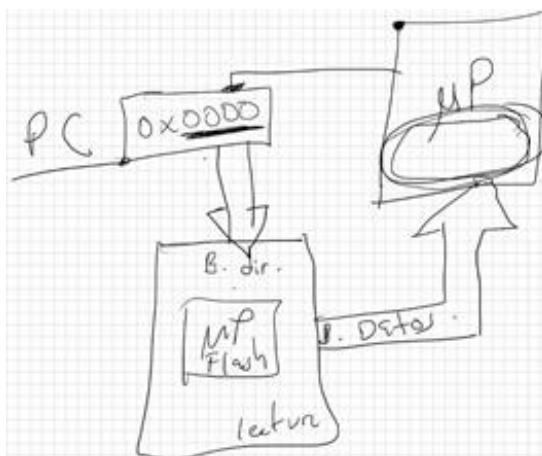
La tarea del microprocesador es buscar y ejecutar instrucciones quemadas en su memoria FLASH utilizando el contador del programa (que es un tipo de registro), como primera instancia el microcontrolador por medio del bus de direcciones buscará la primera dirección contenida en la memoria FLASH del PIC dada por la dirección cero del contador, al hacerlo el contador le entregará su dirección actual a la memoria FLASH donde se encuentra almacenada la siguiente ejecución y le sumará un 1 a su estado actual.



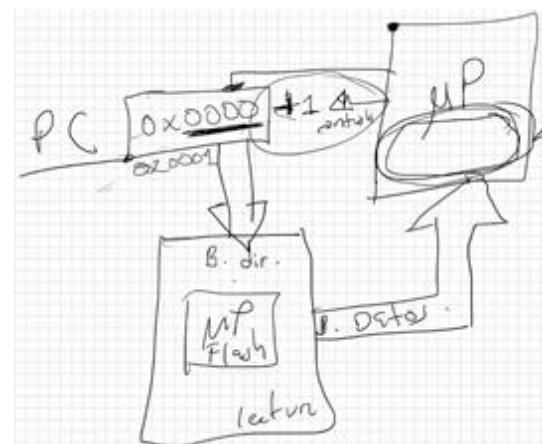
El contador de programa (PC) en un inicio estará con dirección cero, este valor lo carga a la memoria del programa (FLASH) por medio del bus de direcciones.



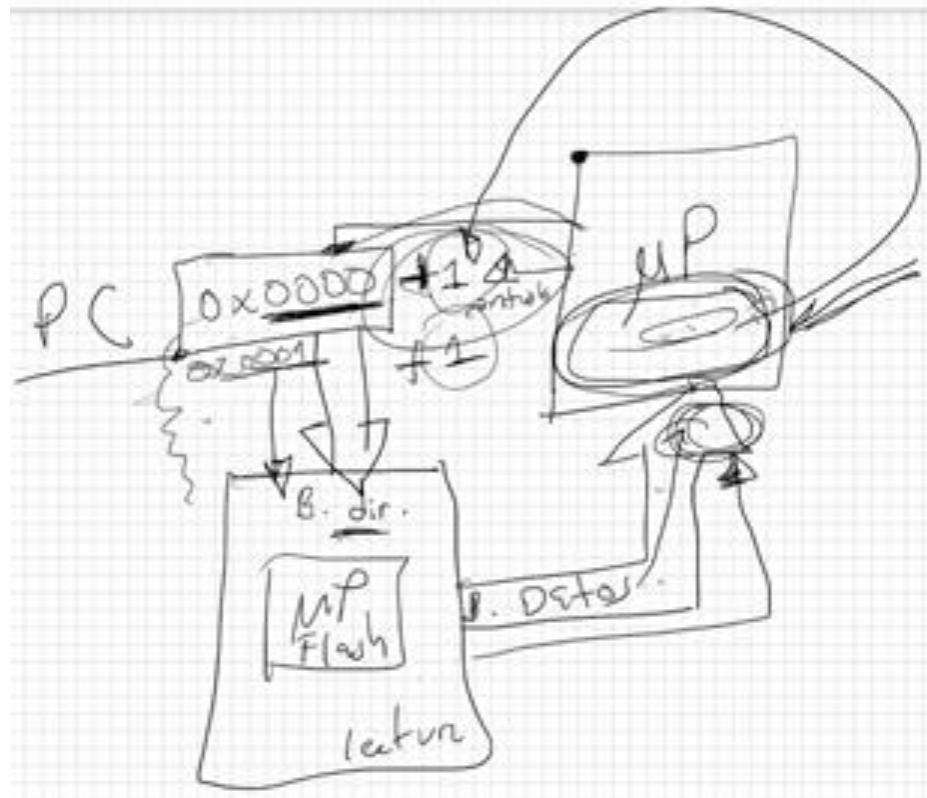
Luego si la memoria FLASH está en modo de lectura le va a dar la información que tenga contenida al microcontrolador por medio del bus de datos.



Cuando el ciclo llegue de nuevo al micro, va a tener una instrucción de +1, esto incrementa por uno el contador cuando se repite el ciclo.

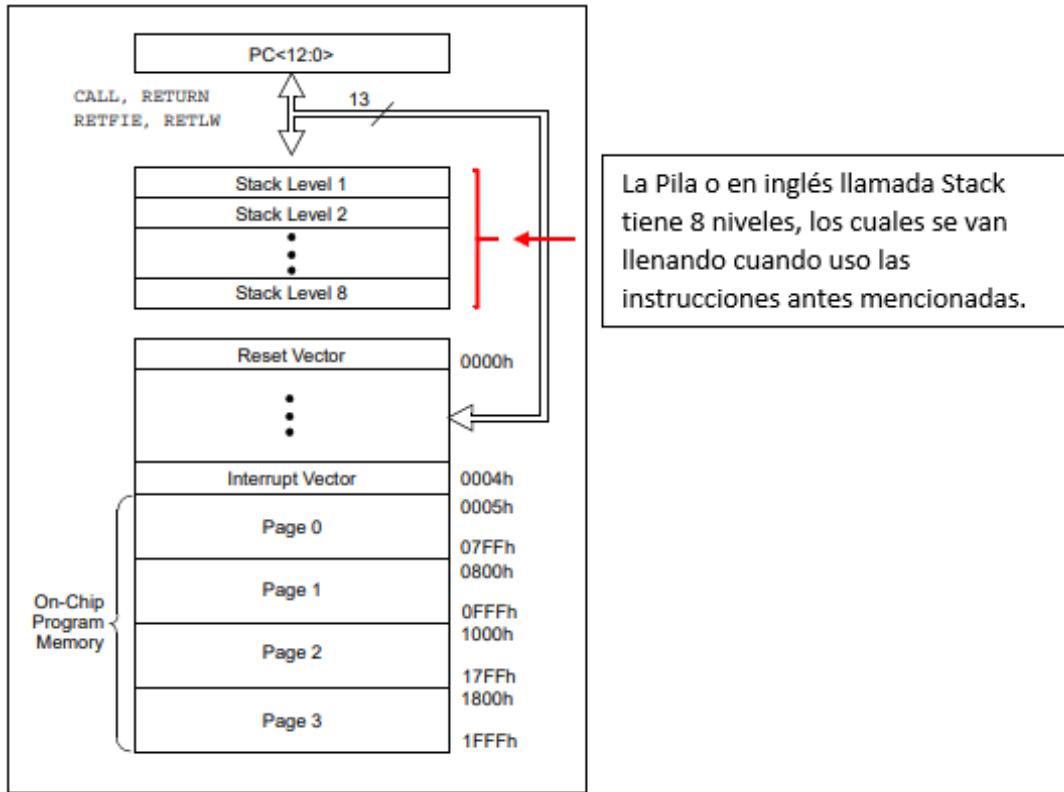


Ahora el microcontrolador consultará de nuevo al contador, que ya tendrá una dirección de 1, y volverá a seguir el ciclo, el contador nunca se debe desbordar (llegar a su límite y volver a contar desde cero), deben de estar muy delimitadas las instrucciones a leer para que el contador las abarque todas. Cuando ocurra un reset el microcontrolador forzará que vuelva a cero el contador.



- **Pila o memoria ROM (Read Only Memory):** Fragmento de la RAM diseñado para guardar direcciones del contador de programa que apuntan a direcciones de registros en la memoria FLASH del PIC16F887, osea a líneas de código (instrucciones) específicas. En el PIC16F887 la Pila no está contenida dentro de la RAM, está de forma separada y sirve para cuando se usan subrutinas o interrupciones, la Pila del PIC tiene 8 niveles contados del 0 al 7, que pueden manejar hasta 8 subrutinas anidadas, esto se puede ver en la organización de la memoria RAM que viene en la página 21 del datasheet.
 - Las únicas instrucciones del PIC que pueden modificar la Pila de la RAM son las siguientes:
 - **CALL:** Carga el valor del contador de programa (PC) que vaya después de donde se encuentre la instrucción CALL, y lo guardará en el primer nivel de la Pila que se encuentre para después subir al siguiente nivel de la Pila, a esto se le llama subir de nivel al **STACK POINTER**, que es otro tipo de contador parecido al PC pero que apunta a los niveles de la Pila.
 - **RETURN:** Esta instrucción baja 1 nivel de la pila al STACK POINTER, lo que hace que el programa cargue en el contador de programa (PC) lo que había guardado en un nivel anterior, retornando así al programa a donde se había quedado después de usar la instrucción CALL.
 - Si la Pila del microcontrolador se desborda, osea que quiere usar más niveles de la Pila que no tiene el PIC (osea más de 8), el programa fallará catastróficamente ya que estará apuntando a un valor del contador de programa que no existe.

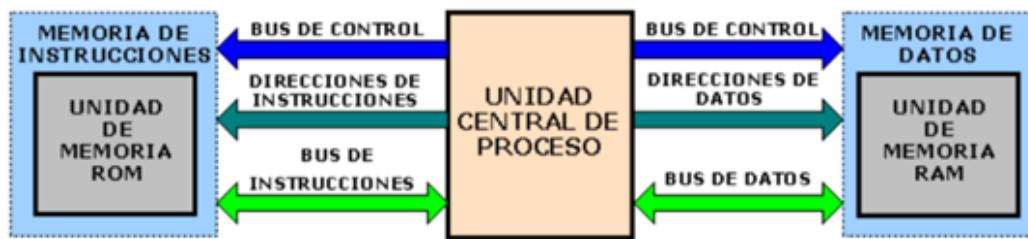
FIGURE 2-3: PROGRAM MEMORY MAP AND STACK FOR THE PIC16F886/PIC16F887



- **Watchdog:** Es un registro que contiene un temporizador donde cada cierto tiempo (marcado por el oscilador o señal de reloj CLK) se irá incrementando un contador desde 0 hasta llegar a 255 porque es de 8 bits, cuando llegue a su límite volverá a ser cero, a esto se le llama desbordamiento y lo que ocasionará es que se reinicie el programa del microcontrolador, esto se hace para evitar que ocurran errores inesperados cuando el microcontrolador esté trabajando a marchas forzadas (largos períodos de tiempo) o en entornos dañinos (con altas temperaturas u oscilaciones que pueden dañar al circuito).
 - En electrónica, un perro guardián (en inglés watchdog) es un mecanismo de seguridad que provoca un reset en el contador del sistema (PC) en caso de que éste se haya bloqueado o que sus circuitos internos hayan fallado por alguna razón. Consiste en un temporizador que irá continuamente decrementando un contador, inicialmente con un valor relativamente alto.
 - Para que el microcontrolador no se reinicie de manera accidental debo usar una instrucción en el código del PIC llamada “alimenta al perro” o watchdog que reiniciará el conteo manualmente, evitando que se desborde. Esto se aplica en dispositivos que estén en continuo funcionamiento los 365 días del año 24 horas al día o que se encuentren en zonas de mucha estática eléctrica, ya que, si llegara a fallar el microcontrolador por estas condiciones, no ejecutará correctamente su función el contador del programa (PC), por lo que no alcanzará a ejecutarse a tiempo la instrucción de “alimenta al perro” dentro del código, desbordándose así el conteo y reiniciando el programa evitando errores inesperados.

- **Ciclo de reloj:** Es el tiempo de oscilación, que se encuentra con el inverso de la frecuencia del oscilador, marcando así el tiempo de muestreo y ejecución de las instrucciones del microcontrolador.
 - También denominado ciclos por segundo o frecuencia, este término hace referencia a la velocidad del procesador incorporado en la CPU del ordenador, y se mide en megahercios (MHz). A mayor índice de frecuencia, más rápido es el procesador y, en consecuencia, el ordenador.
- **Arquitectura del PIC16F887:** Este microcontrolador utiliza una arquitectura Harvard, que accede a sus memorias por separado para ejecutar instrucciones o guardar datos siguiendo el paso de los ciclos de máquina, en específico la memoria de datos es la RAM y la de instrucciones donde se quema el código ensamblador que va a ejecutar el PIC es la memoria FLASH.

ARQUITECTURA HARVARD



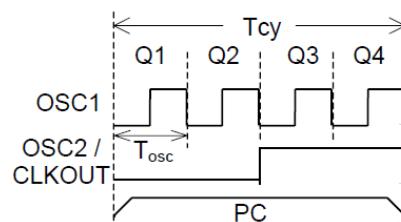
- **Ciclos de máquina:** Al proceso desde que se toma una instrucción de la memoria FLASH hasta que se toma la siguiente, se le llama ciclo de máquina, ya que solo puede hacer una tarea a la vez. Para ejecutar una instrucción se requieren 4 ciclos de reloj que conforman un solo ciclo de máquina, específicamente en el microcontrolador PIC, ya que sigue la arquitectura Harvard que lo hace funcionar más rápido:

$$1 \text{ ciclo de máquina [adimensional]} = 4 \text{ ciclos de reloj [segundos]}$$

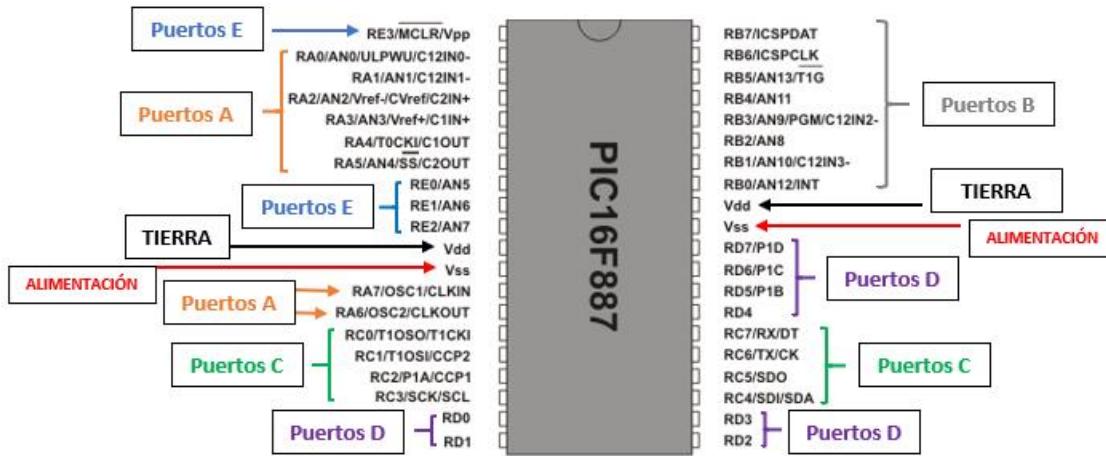
$$1 \text{ ciclo de reloj} = 1 \text{ tosc} = 1 \text{ Periodo} = 1 T = \frac{1}{\text{frecuencia oscilador}} = \frac{1}{f}$$

$$1 \text{ cm} = 4 T = \frac{4}{f}$$

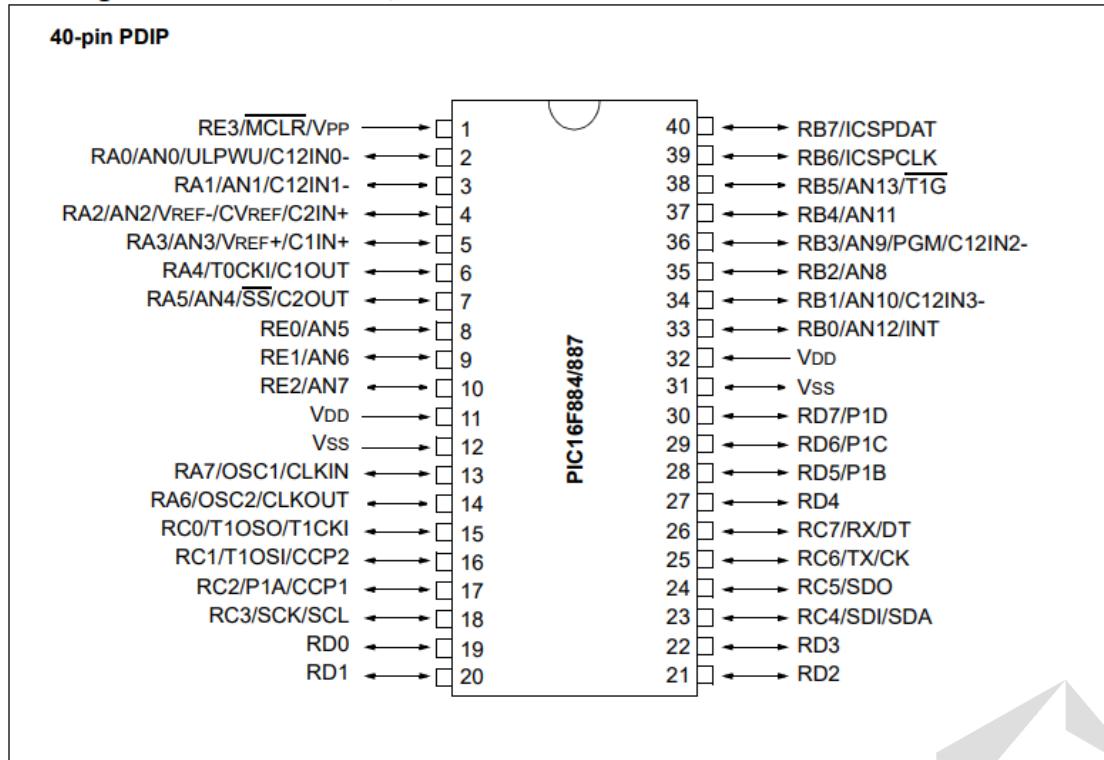
- Q1: El microprocesador obtiene del contador del programa (PC) la dirección de la siguiente instrucción a ejecutar.
- Q2: El contador del programa se incrementa en 1.
- Q3: El microprocesador extrae el contenido de la instrucción a ejecutar.
- Q4: El microprocesador decodifica y ejecuta la instrucción.



- Puertos A, B, C, D y E: Son los pines del PIC16F887 que se usan para recibir o mandar datos fuera del microcontrolador (desde por ejemplo sensores hacia dispositivos electrónicos periféricos como motores o LEDS). Los puertos del PIC entregan 5V y 15mA por pin, pero no puedo usar todos los pines a la vez ya que demandaría mucha potencia que no puede entregar el microcontrolador, ya que como son 36 pines entre los puertos A, B, C, D y E, le estaríamos demandando al microcontrolador una potencia de 0.075 [Watts] por pin, que es un total de 2.7 [Watts]. Puedo ver la ubicación de los pines de cada puerto en el datasheet.



Pin Diagrams – PIC16F884/887, 40-Pin PDIP



Puedo acceder a todos los puertos por medio de ciertos registros de propósito específico llamados PORTA, PORTB, PORTC, PORTD o PORTE, indicados en la tabla.

Además, en la figura podemos observar que no podemos acceder a los puertos del PIC 16F887 desde todos los bancos (también llamados páginas), solo desde los siguientes:

- El banco 0 contiene la dirección del registro de **todos los puertos**:
 - PORTA, PORTB, PORTC, PORTD y PORTE.
- El banco 1 **no contiene la dirección del registro de ningún puerto**.
- El banco 2 contiene la dirección del registro para acceder **solo a 1 puerto**:
 - PORTB.
- El banco 3 **no contiene la dirección del registro de ningún puerto**.

PIC16F882/883/884/886/887

FIGURE 2-6: PIC16F886/PIC16F887 SPECIAL FUNCTION REGISTERS

File Address	File Address	File Address	File Address
Indirect addr. ⁽¹⁾ 00h TMR0 01h OPTION_REG 02h PCL 03h STATUS 04h FSR 05h PORTA 06h PORTB 07h PORTC 08h PORTD ⁽²⁾ 09h PORTE 0Ah PCLATH 0Bh INTCON 0Ch PIR1 0Dh PIR2 0Eh TMR1L 0Fh TMR1H 10h T1CON 11h TMR2 12h T2CON 13h SSPBUF 14h SSPCON 15h CCPR1L 16h CCPR1H 17h CCP1CON 18h RCSTA 19h TXREG 1Ah RCREG 1Bh CCPR2L 1Ch CCPR2H 1Dh CCP2CON 1Eh ADRESH 1Fh ADCONO	Indirect addr. ⁽¹⁾ 00h TMR0 01h OPTION_REG 02h PCL 03h STATUS 04h FSR 05h TRISA 06h TRISB 07h TRISC 08h TRISD ⁽²⁾ 09h TRISE 0Ah PCLATH 0Bh INTCON 0Ch PIE1 0Dh PIE2 0Eh PCON 0Fh OSCCON 10h OSCTUNE 11h SSPCON2 12h PR2 13h SSPADD 14h SSPSTAT 15h WPUB 16h IOCB 17h VRCON 18h TXSTA 19h SPBRG 1Ah SPBRGH 1Bh PWM1CON 1Ch ECCPAS 1Dh PSTRCON 1Eh ADRESL 1Fh ADCON1	Indirect addr. ⁽¹⁾ 00h TMR0 01h OPTION_REG 02h PCL 03h STATUS 04h FSR 05h WDTCON 06h PORTB 07h CM1CON0 08h CM2CON0 09h CM2CON1 0Ah PCLATH 0Bh INTCON 0Ch EEDAT 0Dh EEADR 0Eh EEDATH 0Fh EEADRH	Indirect addr. ⁽¹⁾ 100h TMR0 101h OPTION_REG 102h PCL 103h STATUS 104h FSR 105h SRCON 106h TRISB 107h BAUDCTL 108h ANSEL 109h ANSELH 10Ah PCLATH 10Bh INTCON 10Ch EECON1 10Dh EECON2 ⁽¹⁾ 10Eh Reserved 10Fh Reserved
General Purpose Registers 96 Bytes	General Purpose Registers 80 Bytes	General Purpose Registers 16 Bytes	General Purpose Registers 16 Bytes
20h 3Fh 40h 6Fh 70h 7Fh	EFh F0h FFh	16Fh 170h 17Fh	1EFh 1F0h 1FFh
Bank 0	Bank 1	Bank 2	Bank 3
<small>■ Unimplemented data memory locations, read as '0'.</small>			
<small>Note 1: Not a physical register.</small>			
<small>2: PIC16F887 only.</small>			

Además, puedo ver una descripción más detallada de los Puertos A, B, C, D y E ya que algunos de los pines que pertenecen a cada puerto los puedo configurar para que cumplan alguna función en específico, como por ejemplo ser la entrada de osciladores externos, funcionar como botón de reset, etc. Todo esto se ve indicado en la tabla del datasheet desde la página 18, donde algunos de estos pines de los puertos se pueden convertir en:

- **RA** = Entrada o salida de propósito general.
- **AN** = Entrada o salida analógica o digital.
- **MCLR** = Master Clear o Reset que es lo mismo, al proporcionar Voltaje de 5V a este pin, se borra el código que haya en la memoria FLASH del PIC16F887.
- **OSC1/OSC2** = Entradas de la señal de un oscilador externo.
- **P1** = Salida PWM.

TABLE 1-2: PIC16F884/887 PINOUT DESCRIPTION

Name	Function	Input Type	Output Type	Description
Puerto A	RA0/AN0/ULPWU/C12IN0-	RA0	TTL	CMOS General purpose I/O.
		AN0	AN	— A/D Channel 0 input.
		ULPWU	AN	— Ultra Low-Power Wake-up input.
		C12IN0-	AN	— Comparator C1 or C2 negative input.
Puerto A	RA1/AN1/C12IN1-	RA1	TTL	CMOS General purpose I/O.
		AN1	AN	— A/D Channel 1 input.
		C12IN1-	AN	— Comparator C1 or C2 negative input.
	RA2/AN2/VREF-/CVREF/C2IN+	RA2	TTL	CMOS General purpose I/O.
		AN2	AN	— A/D Channel 2.
		VREF-	AN	— A/D Negative Voltage Reference input.
		CVREF	—	AN Comparator Voltage Reference output.
		C2IN+	AN	— Comparator C2 positive input.
	RA3/AN3/VREF+/C1IN+	RA3	TTL	CMOS General purpose I/O.
		AN3	AN	— A/D Channel 3.
		VREF+	AN	— A/D Positive Voltage Reference input.
		C1IN+	AN	— Comparator C1 positive input.
	RA4/T0CKI/C1OUT	RA4	TTL	CMOS General purpose I/O.
		T0CKI	ST	— Timer0 clock input.
		C1OUT	—	CMOS Comparator C1 output.
	RA5/AN4/SS/C2OUT	RA5	TTL	CMOS General purpose I/O.
		AN4	AN	— A/D Channel 4.
		SS	ST	— Slave Select input.
		C2OUT	—	CMOS Comparator C2 output.
	RA6/OSC2/CLKOUT	RA6	TTL	CMOS General purpose I/O.
		OSC2	—	XTAL Crystal/Resonator.
Puerto A		CLKOUT	—	CMOS Fosc/4 output.
	RA7/OSC1/CLKIN	RA7	TTL	CMOS General purpose I/O.
		OSC1	XTAL	— Crystal/Resonator.
		CLKIN	ST	— External clock input/RC oscillator connection.
RB0/AN12/INT	RB0	TTL	CMOS	General purpose I/O. Individually controlled interrupt-on-change. Individually enabled pull-up.
		AN12	AN	— A/D Channel 12.
		INT	ST	— External interrupt.
	RB1/AN10/C12IN3-	RB1	TTL	CMOS General purpose I/O. Individually controlled interrupt-on-change. Individually enabled pull-up.
		AN10	AN	— A/D Channel 10.
		C12IN3-	AN	— Comparator C1 or C2 negative input.
	RB2/AN8	RB2	TTL	CMOS General purpose I/O. Individually controlled interrupt-on-change. Individually enabled pull-up.
		AN8	AN	— A/D Channel 8.
RB3/AN9/PGM/C12IN2-	RB3	TTL	CMOS	General purpose I/O. Individually controlled interrupt-on-change. Individually enabled pull-up.
		AN9	AN	— A/D Channel 9.
		PGM	ST	— Low-voltage ICSP™ Programming enable pin.
		C12IN2-	AN	— Comparator C1 or C2 negative input.

TABLE 1-2: PIC16F884/887 PINOUT DESCRIPTION (CONTINUED)

Name	Function	Input Type	Output Type	Description
RB4/AN11	RB4	TTL	CMOS	General purpose I/O. Individually controlled interrupt-on-change. Individually enabled pull-up.
	AN11	AN	—	A/D Channel 11.
RB5/AN13/T1G	RB5	TTL	CMOS	General purpose I/O. Individually controlled interrupt-on-change. Individually enabled pull-up.
	AN13	AN	—	A/D Channel 13.
	T1G	ST	—	Timer1 Gate input.
RB6/ICSPCLK	RB6	TTL	CMOS	General purpose I/O. Individually controlled interrupt-on-change. Individually enabled pull-up.
	ICSPCLK	ST	—	Serial Programming Clock.
RB7/ICSPDAT	RB7	TTL	CMOS	General purpose I/O. Individually controlled interrupt-on-change. Individually enabled pull-up.
	ICSPDAT	ST	TTL	ICSP™ Data I/O.
RC0/T1OSO/T1CKI	RC0	ST	CMOS	General purpose I/O.
	T1OSO	—	XTAL	Timer1 oscillator output.
	T1CKI	ST	—	Timer1 clock input.
RC1/T1OSI/CCP2	RC1	ST	CMOS	General purpose I/O.
	T1OSI	XTAL	—	Timer1 oscillator input.
	CCP2	ST	CMOS	Capture/Compare/PWM2.
RC2/P1A/CCP1	RC2	ST	CMOS	General purpose I/O.
	P1A	ST	CMOS	PWM output.
	CCP1	—	CMOS	Capture/Compare/PWM1.
RC3/SCK/SCL	RC3	ST	CMOS	General purpose I/O.
	SCK	ST	CMOS	SPI clock.
	SCL	ST	OD	I ² C™ clock.
RC4/SDI/SDA	RC4	ST	CMOS	General purpose I/O.
	SDI	ST	—	SPI data input.
	SDA	ST	OD	I ² C data input/output.
RC5/SDO	RC5	ST	CMOS	General purpose I/O.
	SDO	—	CMOS	SPI data output.
RC6/TX/CK	RC6	ST	CMOS	General purpose I/O.
	TX	—	CMOS	EUSART asynchronous transmit.
	CK	ST	CMOS	EUSART synchronous clock.
RC7/RX/DT	RC7	ST	CMOS	General purpose I/O.
	RX	ST	—	EUSART asynchronous input.
	DT	ST	CMOS	EUSART synchronous data.
RD0	RD0	TTL	CMOS	General purpose I/O.
RD1	RD1	TTL	CMOS	General purpose I/O.
RD2	RD2	TTL	CMOS	General purpose I/O.
RD3	RD3	TTL	CMOS	General purpose I/O.
RD4	RD4	TTL	CMOS	General purpose I/O.
RD5/P1B	Puerto B	RD5	TTL	CMOS General purpose I/O.
	P1B	—	CMOS	PWM output.
RD6/P1C	Puerto D	RD6	TTL	CMOS General purpose I/O.
	P1C	—	CMOS	PWM output.

TABLE 1-2: PIC16F884/887 PINOUT DESCRIPTION (CONTINUED)

Name	Function	Input Type	Output Type	Description
RD7/P1D	Puerto D	RD7	TTL	CMOS General purpose I/O.
		P1D	AN	PWM output.
RE0/AN5		RE0	TTL	CMOS General purpose I/O.
		AN5	AN	A/D Channel 5.
RE1/AN6		RE1	ST	CMOS General purpose I/O.
		AN6	AN	A/D Channel 6.
RE2/AN7		RE2	TTL	CMOS General purpose I/O.
		AN7	AN	A/D Channel 7.
RE3/MCLR/VPP	Puerto E	RE3	TTL	— General purpose input.
		MCLR	ST	— Master Clear with internal pull-up.
		VPP	HV	— Programming voltage.
VSS		VSS	Power	— Ground reference.
VDD		VDD	Power	— Positive supply.

Aunado a esto, como en cada puerto existen un número determinado de pines, para acceder a cada uno debo ingresar primero al puerto y luego al bit que representa cada pin de los diferentes puertos PORTA, PORTB, PORTC, PORTD y PORTE, esto se hace indicando el puerto que quiero afectar con las 35 instrucciones del PIC e indicando la posición del bit poniendo un número decimal de 0 a 7 para los puertos A, B, C y D y entre 0 y 3 para el puerto E.

Además, como en todos los pines de los puertos no sabemos con qué valor iniciarán después de un reset (porque tienen una x), debemos limpiar sus puertos a que sean todos 0 siempre al iniciar un programa en ensamblador.

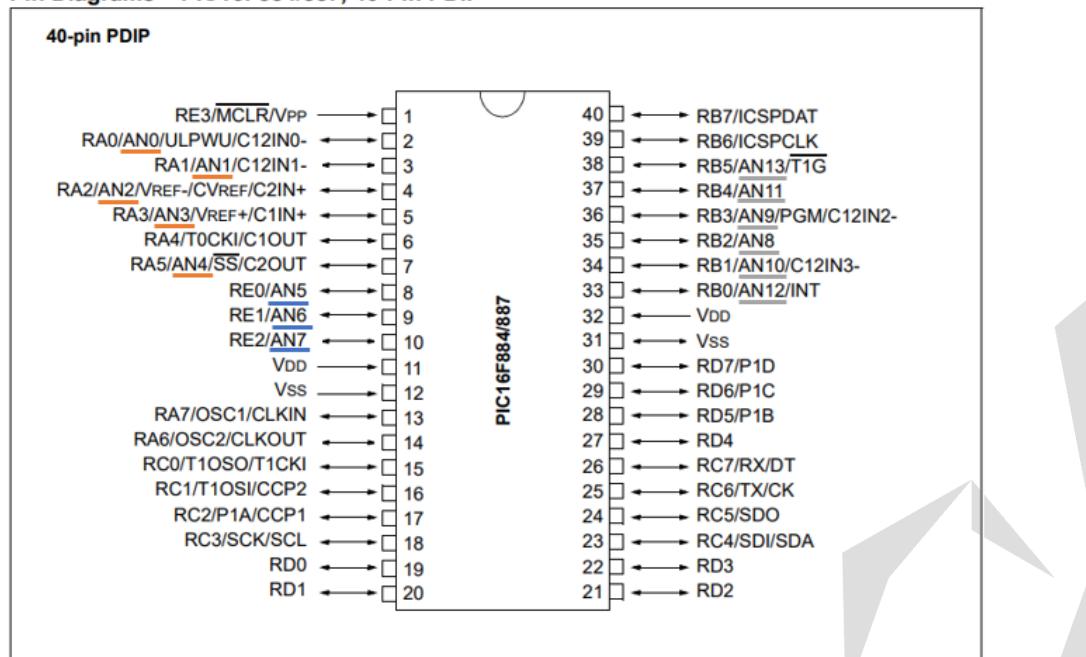
PIC16F882/883/884/886/887

TABLE 2-1: PIC16F882/883/884/886/887 SPECIAL FUNCTION REGISTERS SUMMARY BANK 0

Addr	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on POR, BOR	Page	
Bank 0												
00h	INDF	Addressing this location uses contents of FSR to address data memory (not a physical register)								xxxx xxxx	37,213	
01h	TMRO	Timer0 Module Register								xxxx xxxx	73,213	
02h	PCL	Program Counter's (PC) Least Significant Byte								0000 0000	37,213	
03h	STATUS	IRP	RP1	RP0	TO	PD	Z	DC	C	0001 1xxx	29,213	
04h	FSR	Indirect Data Memory Address Pointer								xxxx xxxx	37,213	
05h	PORTA ⁽³⁾	RA7	RA6	RA5	RA4	RA3	RA2	RA1	RA0	xxxx xxxx	39,213	
06h	PORTB ⁽³⁾	RB7	RB6	RB5	RB4	RB3	RB2	RB1	RB0	xxxx xxxx	48,213	
07h	PORTC ⁽³⁾	RC7	RC6	RC5	RC4	RC3	RC2	RC1	RC0	xxxx xxxx	53,213	
08h	PORTD ^(3,4)	RD7	RD6	RD5	RD4	RD3	RD2	RD1	RD0	xxxx xxxx	57,213	
09h	PORTE ⁽³⁾	=	=	=	=	RE3	RE2 ⁽⁴⁾	RE1 ⁽⁴⁾	RE0 ⁽⁴⁾	xxxx xxxx	59,213	
0Ah	PCLATH	—	—	—	Write Buffer for upper 5 bits of Program Counter							
0Bh	INTCON	GIE	PEIE	TOIE	INTE	RBIE	TOIF	INTF	RBIF ⁽¹⁾	0000 000x	31,213	

- Registros ANSEL y ANSELH: Los únicos puertos que se pueden comportar como entradas/salidas digitales o analógicas son los puertos A, B y E, esto se configura a través de los registros ANSEL y ANSELH, donde a diferencia de los registros TRIS, no acceden a los pines por medio de sus puertos, más bien acceden a ellos individualmente por su nomenclatura AN0, AN1, AN2, AN3, AN4, AN5, AN6, AN7, AN8, AN9, AN10, AN11, AN12 y AN13. En específico el registro ANSEL abarca solo del pin AN0 hasta el AN7 (puertos A y E) y el ANSELH abarca del pin AN8 al AN13 (puerto B).

Pin Diagrams – PIC16F884/887, 40-Pin PDIP



- **PORTA:**
 - AN0, AN1, AN2, AN3 y AN4.
- **PORTB:**
 - AN8, AN9, AN10, AN11, AN12 y AN13.
- **PORTC:**
 - Siempre son puertos digitales.
- **PORTD:**
 - Siempre son puertos digitales.
- **PORTE:**
 - AN5, AN6 y AN7.
 - Poniendo el bit de los registros como 1 o como 0 configura cada pin para que sea una entrada/salida analógica o digital:
 - **1: Entrada o salida Analógica.**
 - **0: Entrada o salida Digital.**

Estos dos registros se encuentran en el banco 3 de la memoria RAM en el PIC16F887.

FIGURE 2-6: PIC16F886/PIC16F887 SPECIAL FUNCTION REGISTERS

File Address	File Address	File Address	File Address
Indirect addr. (1) 00h	Indirect addr. (1) 80h	Indirect addr. (1) 100h	Indirect addr. (1) 180h
TMR0 01h	OPTION_REG 81h	TMR0 101h	OPTION_REG 181h
PCL 02h	PCL 82h	PCL 102h	PCL 182h
STATUS 03h	STATUS 83h	STATUS 103h	STATUS 183h
FSR 04h	FSR 84h	FSR 104h	FSR 184h
PORTA 05h	TRISA 85h	WDTCON 105h	SRCON 185h
PORTB 06h	TRISB 86h	PORTB 106h	TRISB 186h
PORTC 07h	TRISC 87h	CM1CON0 107h	BAUDCTL 187h
PORTD ⁽²⁾ 08h	TRISD ⁽²⁾ 88h	CM2CON0 108h	ANSEL 188h
PORTE 09h	TRISE 89h	CM2CON1 109h	ANSELH 189h
PCLATH 0Ah	PCLATH 8Ah	PCLATH 10Ah	PCLATH 18Ah
INTCON 0Bh	INTCON 8Bh	INTCON 10Bh	INTCON 18Bh
PIR1 0Ch	PIE1 8Ch	EEDAT 10Ch	EECON1 18Ch
PIR2 0Dh	PIE2 8Dh	EEADR 10Dh	EECON2 ⁽¹⁾ 18Dh
TMR1L 0Eh	PCON 8Eh	EEDATH 10Eh	Reserved 18Eh
TMR1H 0Fh	OSCCON 8Fh	EEADRH 10Fh	Reserved 18Fh
T1CON 10h	OSCTUNE 90h		110h
TMR2 11h	SSPCON2 91h		111h
T2CON 12h	PR2 92h		112h
SSPBUF 13h	SSPADD 93h		113h
SSPCON 14h	SSPSTAT 94h		114h
CCPR1L 15h	WPUB 95h		115h
CCPR1H 16h	IOCB 96h	General Purpose Registers	116h
CCP1ICON 17h	VRCON 97h		117h
RCSTA 18h	TXSTA 98h		118h
TXREG 19h	SPBRG 99h	16 Bytes	119h
RCREG 1Ah	SPBRGH 9Ah		11Ah
CCPR2L 18h	PWM1ICON 9Bh		11Bh
CCPR1H 16h	ECCPAS 9Ch		11Ch
CCP1ICON 17h	PSTRCON 9Dh		11Dh
ADRESH 1Eh	ADRESL 9Eh		11Eh
ADCON0 1Fh	ADCON1 9Fh		11Fh
General Purpose Registers 96 Bytes	General Purpose Registers 80 Bytes	A0h	120h
Bank 0	6Fh	EFh	General Purpose Registers 80 Bytes
	70h	F0h	16Fh
	7Fh	FFh	170h
			accesses 70h-7Fh
			accesses 70h-7Fh
			Bank 2
			Bank 3
			1EFh
			1F0h
			1FFh

- Nota: El puerto A, B y E están configurados como puertos analógicos por default, para volverlos digitales y que surta efecto la manipulación de los registros TRISA, TRISB y TRISE, previamente se deben modificar los registros ANSEL y ANSELH.
- Nota: Debido a que el registro MCLR no limpia el contenido de los puertos, a menos que se indique lo contrario, se debe escribir "0" en los bits de los registros PORTX (que serán de salida) antes de configurarlos como salida.

Después de un reset, ANSEL y ANSELH se llenan de unos, esto implica que siempre los puertos A, B y E se inicializan como puertos analógicos.

TABLE 2-4: PIC16F882/883/884/886/887 SPECIAL FUNCTION REGISTERS SUMMARY BANK 3

Addr	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on POR, BOR	Page
Bank 3											
180h	INDF									xxxxx xxxx	37,213
181h	OPTION_REG	RBPU	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0	1111 1111	30,214
182h	PCL									0000 0000	37,213
183h	STATUS	IRP	RP1	RP0	TO	PD	Z	DC	C	0001 1xxx	29,213
184h	FSR									xxxxx xxxx	37,213
185h	SRCON	SR1	SR0	C1SEN	C2REN	PULSS	PULSR	—	FVREN	0000 00-0	93,215
186h	TRISB	TRISB7	TRISB6	TRISB5	TRISB4	TRISB3	TRISB2	TRISB1	TRISB0	1111 1111	48,214
187h	BAUDCTL	ABDOVF	RCIDL	—	SCKP	BRG16	—	WUE	ABDEN	01-0 0-00	160,215
188h	ANSEL	ANS7 ⁽²⁾	ANS6 ⁽²⁾	ANS5 ⁽²⁾	ANS4	ANS3	ANS2	ANS1	ANS0	1111 1111	40,215
189h	ANSELH	—	—	ANS13	ANS12	ANS11	ANS10	ANS9	ANS8	--11 1111	99,215
18Ah	PCLATH	—	—	—	Write Buffer for the upper 5 bits of the Program Counter					---0 0000	37,213
18Bh	INTCON	GIE	PEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF ⁽¹⁾	0000 000x	31,213
18Ch	EECON1	EEPGD	—	—	—	WRERR	WREN	WR	RD	x--- x000	113,215
18Dh	EECON2									---- ----	111,215

- Cuando vaya a configurar puertos, antes de indicar si van a ser salidas o entradas debo usar los registros ANSEL y ANSELH para indicar si van a ser entradas/salidas digitales o analógicas.
- Registros TRIS: Son los registros que hacen que los puertos funcionen como entradas o salidas, son registros de propósito específico y existe uno para cada puerto (TRISA, TRISB, TRISC, TRISD y TRISCE), donde en cada bit del registro se accede a cada pin de cada puerto existente en el PIC para ponerlo como entrada o salida. **Antes de usar el registro TRIS se debe haber usado los registros ANSEL y ANSELH para indicar si los puertos A, B y E son digitales o analógicos.**
 - Si pongo en los registros TRIS:
 - **1: Configura al pin del puerto como Entrada (Input = 1).**
 - **0: Configura al pin del puerto como Salida (Output = 0).**

La mayoría de los registros TRIS se encuentran en el banco 1 aunque el del puerto B igual se encuentra en el banco 3.

- **El banco 0 no contiene ningún registro TRIS.**
- **El banco 1** contiene la dirección de todos los registros TRIS para hacer que funcione como entrada o salida **todos los puertos del PIC:**
 - **TRISA, TRISB, TRISC, TRISD y TRISE.**
- **El banco 2 no contiene ningún registro TRIS.**

- El banco 3 contiene solo la dirección del registro TRISB para hacer que funcione como entrada o salida el puerto B:

■ TRISB.

FIGURE 2-6: PIC16F886/PIC16F887 SPECIAL FUNCTION REGISTERS

File	File	File	File
Address	Address	Address	Address
Indirect addr. (1)	Indirect addr. (1)	Indirect addr. (1)	Indirect addr. (1)
00h TMR0	01h OPTION_REG	00h TMR0	100h OPTION_REG
02h PCL	02h PCL	02h PCL	101h PCL
03h STATUS	03h STATUS	03h STATUS	102h STATUS
04h FSR	04h FSR	04h FSR	103h FSR
05h TRISA	05h TRISB	05h WDTCON	104h SRCON
06h TRISB	06h PORTB	06h PORTB	105h TRISB
07h TRISC	07h PORTC	07h CM1ICON0	106h BAUDCTL
08h TRISD ⁽²⁾	08h PORTD ⁽²⁾	08h CM2ICON0	107h ANSEL
09h TRISE	09h PORTE	09h CM2ICON1	108h ANSELH
0Ah PCLATH	0Ah INTCON	0Ah PCLATH	109h PCLATH
0Bh INTCON	0Ch PIE1	0Bh INTCON	10Ah INTCON
0Ch PIE1	0Dh PIE2	0Ch EEDAT	10Bh EECON1
0Dh PIE2	0Eh PCON	0Dh EEAADR	10Ch EECON2 ⁽¹⁾
0Eh PCON	0Fh OSCCON	0Eh EEDATH	10Dh Reserved
0Fh OSCCON	10h OSCTUNE	0Fh EEADRH	10Fh Reserved
10h OSCTUNE	11h SSPCON2	10h	110h
11h SSPCON2	12h PR2	10h	111h
12h PR2	13h SSPBUF	10h	112h
13h SSPBUF	14h SSPSTAT	10h	113h
14h SSPSTAT	15h WPUB	10h	114h
15h WPUB	16h IOCB	10h	115h
16h IOCB	17h CCP1CON	10h	116h
17h CCP1CON	18h RCSTA	10h	117h
18h RCSTA	19h TXSTA	10h	118h
19h TXSTA	1Ah SPBRG	10h	119h
1Ah SPBRG	1Ah CCP2R2L	10h	11Ah
1Ah CCP2R2L	1Ah PWM1CON	10h	11Bh
1Ah PWM1CON	1Ch CCP1R1H	10h	11Ch
1Ch CCP1R1H	1Ch CCP2CON	10h	11Dh
1Ch CCP2CON	1Dh PSTRCON	10h	11Eh
1Dh PSTRCON	1Eh ADRESH	10h	11Fh
1Eh ADRESH	1Fh ADCON0	10h	120h
1Fh ADCON0	20h	10h	120h
20h	General Purpose Registers	10h	120h
3Fh	General Purpose Registers	10h	120h
40h	80 Bytes	10h	120h
6Fh	General Purpose Registers	10h	120h
70h	80 Bytes	10h	120h
7Fh	accesses 70h-7Fh	10h	120h
Bank 0	Bank 1	Bank 2	Bank 3
96 Bytes			

Por default después de ocurrir un reset en el PIC, todos los bits de los registros TRIS se pondrán como 1 (osea como entradas). Siempre que vayamos a hacer un programa debemos definir cuáles pines son los que pondremos como entrada y cuáles como salida.

PIC16F882/883/884/886/887

TABLE 2-2: PIC16F882/883/884/886/887 SPECIAL FUNCTION REGISTERS SUMMARY BANK 1

Addr	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on POR, BOR	Page
Bank 1											
80h INDF		Addressing this location uses contents of FSR to address data memory (not a physical register)							xxxx xxxx	37,213	
81h OPTION_REG	RBU ₁	INTE _{DG}	T0CS	T0SE	PSA	PS2	PS1	PS0	1111 1111	30,214	
82h PCL	Program Counter's (PC) Least Significant Byte							0000 0000	37,213		
83h STATUS	IRP	RP1	RP0	TO	PD	Z	DC	C	0001 1xxx	29,213	
84h FSR	Indirect Data Memory Address Pointer							xxxx xxxx	37,213		
85h TRISA	TRISA7	TRISA6	TRISA5	TRISA4	TRISA3	TRISA2	TRISA1	TRISA0	1111 1111	39,214	
86h TRISB	TRISB7	TRISB6	TRISB5	TRISB4	TRISB3	TRISB2	TRISB1	TRISB0	1111 1111	48,214	
87h TRISC	TRISC7	TRISC6	TRISC5	TRISC4	TRISC3	TRISC2	TRISC1	TRISCO	1111 1111	53,214	
88h TRISD ⁽³⁾	TRISD7	TRISD6	TRISD5	TRISD4	TRISD3	TRISD2	TRISD1	TRISD0	1111 1111	57,214	
89h TRISE	—	—	—	—	TRISE3	TRISE2 ⁽³⁾	TRISE1 ⁽³⁾	TRISE0 ⁽³⁾	---- 1111	59,214	
8Ah PCLATH	—	—	—	Write Buffer for the upper 5 bits of the Program Counter					---0 0000	37,213	
8Bh INTCON	GIE	PEIE	TOIE	INTE	RBIE	T0IF	INTF	RBIF ⁽¹⁾	0000 000x	31,213	
8Ch PIE1	—	ADIE	RCIE	TXIE	SSPIE	CCP1IE	TMR2IE	TMR1IE	-000 0000	32,214	
8Dh PIE2	OSFIE	C2IE	C1IE	EEIE	BCLIE	ULPWUIE	—	CCP2IE	0000 00-0	33,214	

35 instrucciones PIC:

Su descripción y ejemplos vienen en la página 225 del datasheet o en el archivo Manual de instrucciones PIC.pdf

En la parte donde dice 14-Bit Opcode podemos ver el equivalente en bits de cada instrucción del PIC ya que en esa parte es donde se aplica el lenguaje ensamblador, que transforma esos bits a una palabra para que así sea más entendible para nosotros los humanos, mientras que a la máquina, solo le manda los bits indicados. **La mayoría de las instrucciones afectan a las banderas del microcontrolador con algunas pocas excepciones.**

TABLE 15-2: PIC16F883/884/886/887 INSTRUCTION SET

Mnemonic, Operands	Description	Cycles	14-Bit Opcode		Status Affected	Notes
			MSb	Lsb		
BYTE-ORIENTED FILE REGISTER OPERATIONS						
ADDWF	f, d	Add W and f	1	00 0111 dfff ffff	C, DC, Z	1, 2
ANDWF	f, d	AND W with f	1	00 0101 dfff ffff	Z	1, 2
CLRF	f	Clear f	1	00 0001 1fff ffff	Z	2
CLRW	-	Clear W	1	00 0001 0xxx xxxx	Z	
COMF	f, d	Complement f	1	00 1001 dfff ffff	Z	1, 2
DECFSZ	f, d	Decrement f, Skip if 0	1(2)	00 1011 dfff ffff	Z	1, 2, 3
INCF	f, d	Increment f	1	00 1010 dfff ffff	Z	1, 2
INCFSZ	f, d	Increment f, Skip if 0	1(2)	00 1111 dfff ffff	Z	1, 2, 3
IORWF	f, d	Inclusive OR W with f	1	00 0100 dfff ffff	Z	1, 2
MOVF	f, d	Move f	1	00 1000 dfff ffff	Z	1, 2
MOVWF	f	Move W to f	1	00 0000 1fff ffff	Z	1, 2
NOP	-	No Operation	1	00 0000 0xx0 0000	Z	
RLF	f, d	Rotate Left f through Carry	1	00 1101 dfff ffff	C	1, 2
RRF	f, d	Rotate Right f through Carry	1	00 1100 dfff ffff	C	1, 2
SUBWF	f, d	Subtract W from f	1	00 0010 dfff ffff	C, DC, Z	1, 2
SWAPF	f, d	Swap nibbles in f	1	00 1110 dfff ffff	Z	1, 2
XORWF	f, d	Exclusive OR W with f	1	00 0110 dfff ffff	Z	1, 2
BIT-ORIENTED FILE REGISTER OPERATIONS						
BCF	f, b	Bit Clear f	1	01 00bb bfff ffff		1, 2
BSF	f, b	Bit Set f	1	01 01bb bfff ffff		1, 2
BTFSZ	f, b	Bit Test f, Skip if Clear	1(2)	01 10bb bfff ffff		3
BTFSZ	f, b	Bit Test f, Skip if Set	1(2)	01 11bb bfff ffff		3
LITERAL AND CONTROL OPERATIONS						
ADDLW	k	Add literal and W	1	11 111x kkkk kkkk	C, DC, Z	
ANDLW	k	AND literal with W	1	11 1001 kkkk kkkk	Z	
CALL	k	Call Subroutine	2	10 0kkk kkkk kkkk		
CLRWDT	-	Clear Watchdog Timer	1	00 0000 0110 0100	TO, PD	
GOTO	k	Go to address	2	10 1kkk kkkk kkkk		
IORLW	k	Inclusive OR literal with W	1	11 1000 kkkk kkkk	Z	
MOVLW	k	Move literal to W	1	11 00xx kkkk kkkk		
RETFIE	-	Return from interrupt	2	00 0000 0000 1001		
RETLW	k	Return with literal in W	2	11 01xx kkkk kkkk		
RETURN	-	Return from Subroutine	2	00 0000 0000 1000	TO, PD	
SLEEP	-	Go into Standby mode	1	00 0000 0110 0011		
SUBLW	k	Subtract W from literal	1	11 110x kkkk kkkk	C, DC, Z	
XORLW	k	Exclusive OR literal with W	1	11 1010 kkkk kkkk	Z	

REGLAS

- Los registros f y valores literales k que usaré para ejecutar las 35 instrucciones del PIC son de 8 bits (1 byte) y normalmente las declararé de forma hexadecimal, esto se hace poniendo: **0Xnúmero_hexadecimal**.

- También puedo declarar registros en forma decimal (aunque no es tan común de verse) poniendo un punto seguido del número decimal que quiera: `.número_decimal` tomando en cuenta que podré poner valores decimales del 0 al 255 solamente porque se guardan en números binarios de 8 bits.
- Para aplicar las instrucciones a bits o registros de propósito específico existe una igualdad entre palabras y números, que también se les llama **directivas EQU**, esto para que sea más fácil el recordar la función y dirección de RAM para cada uno, los nombres los puedo encontrar en la tabla de organización de registros para la memoria RAM del PIC16F887 en la página 25 del datasheet.

Todos los demás registros y bits que deberé usar en las instrucciones del PIC16F887 vienen indicados en la página 225 del datasheet, capítulo 15. Además, estos también se pueden ver cómo son utilizados al convertir una instrucción en lenguaje ensamblador a su código binario correspondiente, indicado en la columna 14-Bit Opcode de la tabla anterior.

TABLE 15-1: OPCODE FIELD DESCRIPTIONS

Field	Description
f	Register file address (0x00 to 0x7F)
W	Working register (accumulator)
b	Bit address within an 8-bit file register
k	Literal field, constant data or label
x	Don't care location (= 0 or 1). The assembler will generate code with x = 0. It is the recommended form of use for compatibility with all Microchip software tools.
d	Destination select; d = 0: store result in W, d = 1: store result in file register f. Default is d = 1 .
PC	Program Counter
TO	Time-out bit
C	Carry bit
DC	Digit carry bit
Z	Zero bit
PD	Power-down bit

1. ADDWF f, d (C, DC, Z):

- a. Suma el contenido del acumulador W con el contenido de un registro de la memoria RAM indicado por la dirección f y el resultado lo guarda en el acumulador W o en el mismo registro f dependiendo del valor del parámetro d:

- Si d = 0 el resultado se guarda en el acumulador W.
 - ✚ Aunque en vez de poner 0, puedo poner directamente W.
- Si d = 1 el resultado se guarda en el registro f que se utilizó en la instrucción.
 - ✚ Aunque en vez de poner 1, puedo poner directamente f.

- b. Se afecta a las banderas C = acarreo, DC= acarreo decimal y Z = ceros.
 - c. Tarda 1 ciclo de máquina en ejecutarse.
2. ANDWF f, d (Z):
- a. Realiza la función lógica AND entre lo que haya en la dirección f del registro RAM y el acumulador W.
 - Si d = 0 el resultado se guarda en el acumulador W.
 - ✚ Aunque en vez de poner 0, puedo poner directamente W.
 - Si d = 1 el resultado se guarda en el registro f que se utilizó en la instrucción.
 - ✚ Aunque en vez de poner 1, puedo poner directamente f.
- A esto se le llama máscara AND y lo que hará es obligar a ciertos bits del registro f que se vuelvan 0, dejando a los demás intactos. La máscara en este caso se colocaría previamente en el acumulador W, poniendo un número binario de 8 bits que tenga un 0 donde quiero que se obligue a los bits del registro f que se vuelvan cero y dejando un 1 en la posición de los bits del registro f que no quiero afectar.
- Funciones de los bits en la máscara AND:
- 0: obliga a los bits a volverse 0.
 - 1: No afecta en nada, deja los bits en su forma inicial.
- b. Solo se afecta a la bandera Z = ceros.
 - c. Tarda 1 ciclo de máquina en ejecutarse.

And. Obliga a ciertos bits a ser 0's sin tocar al resto

&	b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0
/	1	1	0	1	1	0	1	0
	b_7	b_6	0	b_4	b_3	0	b_1	0

3. CLRF f (Z):
- a. Llena de ceros la dirección f del registro RAM (escribe de 0x00 en el registro de RAM indicado por el parámetro f).
 - Siempre Z = 1.
 - b. Los registros de propósito general los puedo llenar de ceros sin problema.
 - c. Pero hay algunos registros de propósito específico (los que tienen nombre) que no se pueden llenar todos sus bits de ceros, aunque se lo indique con esta instrucción, puede que solo se puedan poner algunos como ceros o puede que ninguno se pueda poner como cero, esto viene indicado desde la página 29 hasta la 36 del datasheet para cada registro individualmente, dependiendo de si los bits del registro están marcados con:
 - **R = Read (leible):** Estos bits no los puedo cambiar, solo leer.
 - **W = Write (escribible):** Estos bits si los puedo cambiar (o escribir que es lo mismo).
 - **R/W = Read/Write (leible/escribible):** Estos bits los puedo leer y escribir.
 - d. Tarda 1 ciclo de máquina en ejecutarse.
4. CLRW k (Z):
- a. Llena de ceros al acumulador W (escribe de 0x00 en el registro acumulador W).
 - Siempre Z = 1.
 - b. Tarda 1 ciclo de máquina en ejecutarse.

5. COMF f, d (Z):

- a. Realiza el complemento A1 a lo que haya en la dirección f del registro de RAM indicado. El complemento A1 literal invierte todos los bits que haya en el registro (si era 1 lo hace 0 y si era 0 lo hace 1) por lo que **sirve para aplicar una compuerta NOT al contenido del registro.**
 - Si d = 0 el resultado se guarda en el acumulador W.
⊕ **Aunque en vez de poner 0, puedo poner directamente W.**
 - Si d = 1 el resultado se guarda en el registro f que se utilizó en la instrucción.
⊕ **Aunque en vez de poner 1, puedo poner directamente F.**

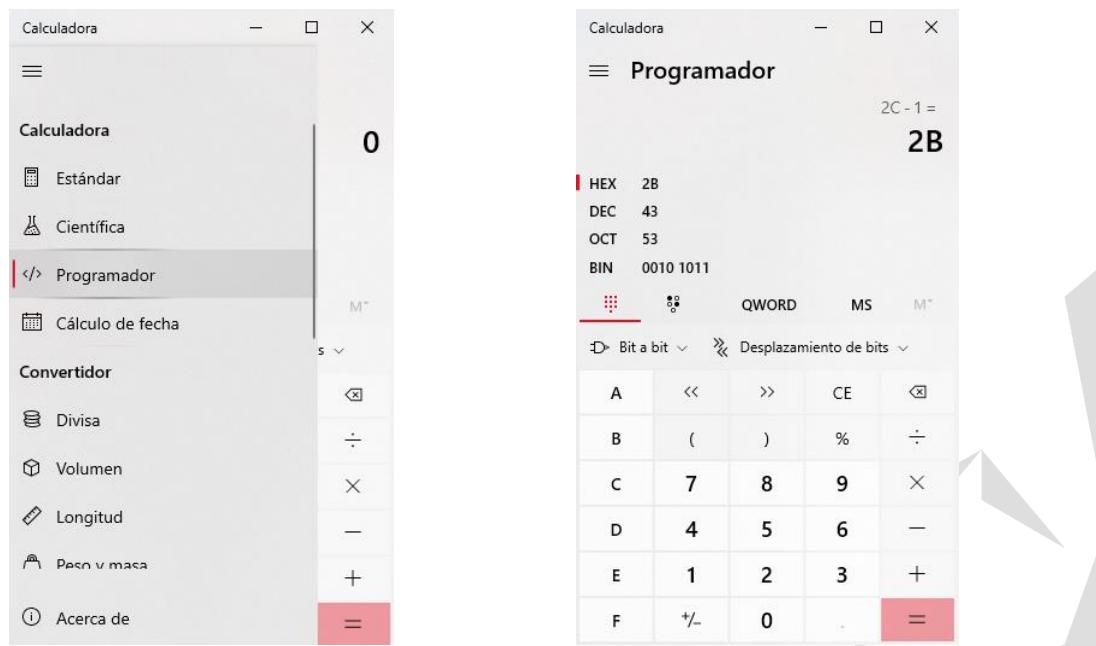
- b. Solo se afecta a la bandera Z = ceros.
- c. **Tarda 1 ciclo de máquina en ejecutarse.**

6. DECF f, d (Z):

- a. Decrementa en 1 a lo que haya en la dirección f del registro de RAM indicado.
 - Si d = 0 el resultado se guarda en el acumulador W.
⊕ **Aunque en vez de poner 0, puedo poner directamente W.**
 - Si d = 1 el resultado se guarda en el registro f que se utilizó en la instrucción.
⊕ **Aunque en vez de poner 1, puedo poner directamente f.**
- b. Solo se afecta a la bandera Z = ceros.
- c. Si decremento por 1 a un número cero, lo empajaré a que sea el máximo que pueda empujar y se repite el ciclo de la siguiente manera:
 - Si decremento por 1 el número 00, su resultado es 11.
 - Si decremento por 1 el número 11, su resultado es 10.
 - Si decremento por 1 el número 10, su resultado es 01.
 - Si decremento por 1 el número 01, su resultado es 00.

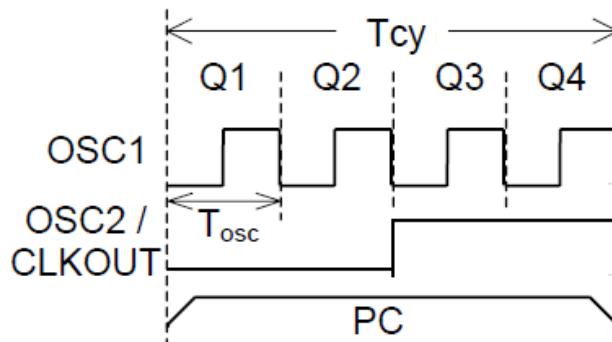
Si decremento a un registro que está en ceros en el PIC16F887 lo mandará a ser FF porque son de 8 bits.

- d. **Tarda 1 ciclo de máquina en ejecutarse.**
- e. Puedo realizar operaciones binarias con la calculadora para hacer un decrecimiento binario.



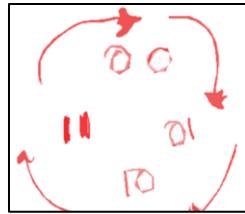
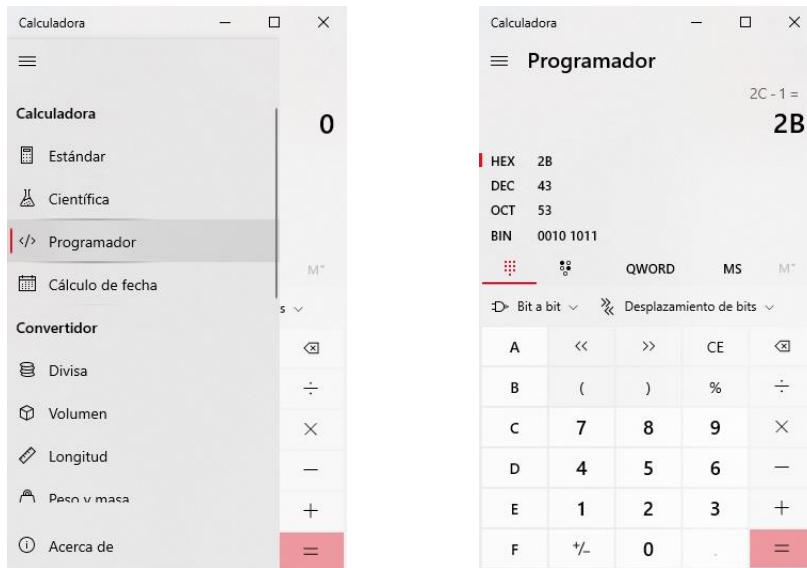
7. DECFSZ f, d:
- Esta operación es lo más parecido a un condicional if que existe en el idioma ensamblador, normalmente se usa para evitar instrucciones GOTO.
 - Esta operación decrementa (le resta 1) a lo que haya en la dirección f del registro de RAM indicado y el resultado lo guarda en el acumulador W o en el mismo registro F, además dependiendo de si el resultado es cero o no, se brincará la siguiente instrucción o la ejecutará normalmente.
 - Si el resultado es cero (00 hexadecimal):
 - Se brinca la siguiente instrucción que haya en el código (aumenta en 1 al contador de programa).
 - Tarda 2 ciclos de máquina en ejecutarse.
 - Si el resultado NO es cero:
 - Sigue la ejecución normal del código.
 - Tarda 1 ciclo de máquina en ejecutarse.
 - No afecta ninguna bandera.
 - Si d = 0 el resultado se guarda en el acumulador W.
 - ✚ Aunque en vez de poner 0, puedo poner directamente W.
 - Si d = 1 el resultado se guarda en el registro f que se utilizó en la instrucción.
 - ✚ Aunque en vez de poner 1, puedo poner directamente f.
 - Si decremento por 1 a un número cero, lo empujaré a que sea el máximo que pueda empujar y se repite el ciclo de la siguiente manera:
 - Si decremento por 1 el número 00, su resultado es 11.
 - Si decremento por 1 el número 11, su resultado es 10.
 - Si decremento por 1 el número 10, su resultado es 01.
 - Si decremento por 1 el número 01, su resultado es 00.

Si decremento a un registro que está en ceros (00) en el PIC16F887 lo mandará a ser FF porque son de 8 bits.



8. INCF f, d (Z):
- Le suma 1 (incrementa) a lo que haya almacenado en la dirección f del registro de RAM indicada.
 - Si d = 0 el resultado se guarda en el acumulador W.
 - ✚ Aunque en vez de poner 0, puedo poner directamente W.
 - Si d = 1 el resultado se guarda en el registro f que se utilizó en la instrucción.
 - ✚ Aunque en vez de poner 1, puedo poner directamente f.
 - Solo se afecta a la bandera Z = ceros.

- c. Si incremento por 1 a un número máximo, lo empujaré a que vuelva cero y se repite el ciclo de la siguiente manera:
- Si incremento por 1 el número 11, su resultado es 00.
 - Si incremento por 1 el número 00, su resultado es 01.
 - Si incremento por 1 el número 01, su resultado es 10.
 - Si incremento por 1 el número 10, su resultado es 11.
- Si incremento a un registro que está en su punto máximo que es FF en el PIC16F887 lo mandará a ser ceros porque son de 8 bits.
- d. Tarda 1 ciclo de máquina en ejecutarse.
- e. Puedo realizar operaciones binarias con la calculadora para hacer un incremento binario.



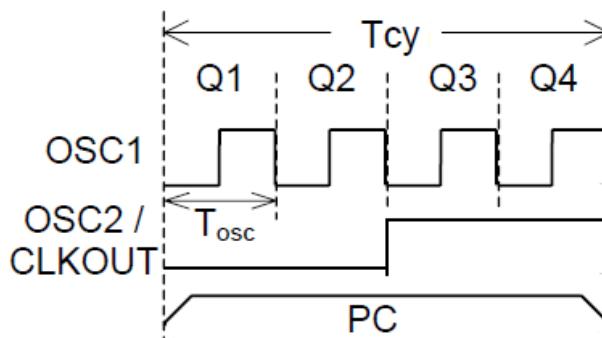
9. INCFSZ f, d:

- Esta operación es lo más parecido a un condicional if que existe en el idioma ensamblador, normalmente se usa para evitar instrucciones GOTO.
- Esta operación incrementa (le suma 1) a lo que haya en la dirección f del registro de RAM indicado y dependiendo de si el resultado es cero o no, se brincará la siguiente instrucción o si la ejecutará normalmente.
 - Si el resultado es cero (00 hexadecimal):
 - Se brinca la siguiente instrucción que haya en el código (aumenta en 1 al contador de programa).
 - Tarda 2 ciclos de máquina en ejecutarse.
 - Si el resultado NO es cero:
 - Sigue la ejecución normal del código.
 - Tarda 1 ciclo de máquina en ejecutarse.



- c. No afecta ninguna bandera.
 - Si $d = 0$ el resultado se guarda en el acumulador W.
 - ✚ Aunque en vez de poner 0, puedo poner directamente W.
 - Si $d = 1$ el resultado se guarda en el registro f que se utilizó en la instrucción.
 - ✚ Aunque en vez de poner 1, puedo poner directamente f.
- d. Si incremento por 1 a un número máximo, lo empujaré a que vuelva cero y se repite el ciclo de la siguiente manera:
 - Si incremento por 1 el número 11, su resultado es 00.
 - Si incremento por 1 el número 00, su resultado es 01.
 - Si incremento por 1 el número 01, su resultado es 10.
 - Si incremento por 1 el número 10, su resultado es 11.

Si incremento a un registro que está en su punto máximo que es FF en el PIC16F887 lo mandará a ser ceros (00) porque son de 8 bits.



10. IORWF f, d (Z):

- a. Realiza la función lógica OR ("O" inclusiva) entre lo que haya en la dirección f del registro RAM y el acumulador W.
 - Si $d = 0$ el resultado se guarda en el acumulador W.
 - ✚ Aunque en vez de poner 0, puedo poner directamente W.
 - Si $d = 1$ el resultado se guarda en el registro f que se utilizó en la instrucción.
 - ✚ Aunque en vez de poner 1, puedo poner directamente f.

A esto se le llama máscara OR y lo que hará es obligar a ciertos bits del registro f que se vuelvan 1, dejando a los demás intactos. La máscara en este caso se colocaría previamente en el acumulador W, poniendo un número binario de 8 bits que tenga un 1 donde quiero que se obligue a los bits del registro f que se vuelvan uno y dejando un 0 en la posición de los bits del registro f que no quiero afectar.

Funciones de los bits en la máscara OR:

- 0: No afecta en nada, deja los bits en su forma inicial.
- 1: Obliga a los bits a volverse 1.

- b. Solo se afecta a la bandera Z = ceros.
- c. Tarda 1 ciclo de máquina en ejecutarse.

DR Obliga a ciertos bits a ser 1's sin tocar al resto

$$\begin{array}{r}
 b_7 \ b_6 \ b_5 \ b_4 \\
 + \quad 1 \quad 1 \quad 0 \quad 0 \\
 \hline
 1 \quad 1 \quad b_5 \ b_4
 \end{array}$$

$$\begin{array}{r}
 b_3 \ b_2 \ b_1 \ b_0 \\
 + \quad 1 \quad 1 \quad 0 \quad 0 \\
 \hline
 1 \quad 1 \quad b_1 \ b_0
 \end{array}$$

11. MOVF f, d (Z):

- Lee el contenido de un registro de la RAM indicado por la dirección f y lo coloca en el mismo registro f o en el acumulador W.
 - Si d = 0 el resultado se guarda en el acumulador W.
 - Aunque en vez de poner 0, puedo poner directamente W.
 - Si d = 1 el resultado se guarda en el registro f que se utilizó en la instrucción.
 - Aunque en vez de poner 1, puedo poner directamente f.
- El uso más común para esta instrucción es mover un valor del registro f al acumulador W para guardarlo temporalmente
- Si d = 1, la instrucción para que no esté haciendo nada, ya que extrae el contenido del registro f y lo vuelve a colocar en el mismo registro f.
 - Aunque puede servir para extraer el valor del registro f y luego volver a colocarlo en f cuando se quiere comprobar si el registro f está en ceros (00), de esta manera no hay necesidad de tocar el contenido del acumulador o recurrir a otras instrucciones.
- Tarda 1 ciclo de máquina en ejecutarse.

12. MOVWF f:

- Lee el contenido del acumulador W y lo coloca en un registro de la RAM indicado por la dirección f.
 - Esto sirve para pasar el contenido de un registro de RAM a otro, aunque para ello el contenido del registro de entrada previamente se debe haber guardado en el acumulador W con la instrucción MOVF.
- Tarda 1 ciclo de máquina en ejecutarse.

13. NOP:

- Deja pasar un ciclo de máquina sin hacer nada.
 - Su uso es fundamental para el manejo de tiempos, osea para que cada instrucción del microcontrolador se ejecute en el momento deseado.
- No afecta ninguna bandera.
- Tarda 1 ciclo de máquina en ejecutarse.

14. RLF f, d (C):

- Lo que hará esta instrucción es añadir lo que haya en la bandera de acarreo C al bit menos significativo (el que vale menos, osea el de hasta la derecha) del número binario que haya en el registro f indicado de la RAM, y recorrer todos los bits originales a la izquierda para crear uno nuevo. El bit que haya sido sacado del número binario de 8 bits original será el nuevo acarreo, se puede decir que ejecuta una rotación a la izquierda.
 - Si d = 0 el resultado se guarda en el acumulador W.
 - Aunque en vez de poner 0, puedo poner directamente W.
 - Si d = 1 el resultado se guarda en el registro f que se utilizó en la instrucción.
 - Aunque en vez de poner 1, puedo poner directamente f.

- Esto se usa por ejemplo para el control de motores a pasos:

- C = 0, Contenido del registro f = 0001.
- C = 0, Contenido del registro f = 0010.
- C = 0, Contenido del registro f = 0100.
- C = 0, Contenido del registro f = 1000.
- C = 1, Contenido del registro f = 0000.

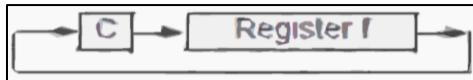


- a. El acarreo lo pasa hasta la derecha del registro f y toma el bit más significativo (el de la izquierda) como nuevo acarreo.
- b. Y se vuelve a iniciar el ciclo.
- c. Tarda 1 ciclo de máquina en ejecutarse.



15. RRF f, d (C):

- a. Lo que hará esta instrucción es añadir lo que haya en la bandera de acarreo C al bit más significativo (el que vale más, osea el de hasta la izquierda) del número binario que haya en el registro f indicado de la RAM y recorrer todos los bits originales a la derecha para crear uno nuevo. El bit que haya sido sacado del número binario de 8 bits original será el nuevo acarreo, se puede decir que ejecuta una rotación a la derecha.
 - Si d = 0 el resultado se guarda en el acumulador W.
 - ⊕ Aunque en vez de poner 0, puedo poner directamente W.
 - Si d = 1 el resultado se guarda en el registro f que se utilizó en la instrucción.
 - ⊕ Aunque en vez de poner 1, puedo poner directamente f.
- b. Esto se usa por ejemplo para el control de motores a pasos:
 - ⊕ C = 0, Contenido del registro f = 0001.
 - ⊕ C = 1, Contenido del registro f = 0000.
 - ⊕ C = 0, Contenido del registro f = 1000.
 - ⊕ C = 0, Contenido del registro f = 0100.
 - ⊕ C = 0, Contenido del registro f = 0010.
 - a. El acarreo lo pasa hasta la izquierda del registro f y toma el bit menos significativo (el de la derecha) como nuevo acarreo.
 - b. Y se vuelve a iniciar el ciclo.
- c. Tarda 1 ciclo de máquina en ejecutarse.



16. SUBWF f, d (C, DC, Z):

- a. Se resta el contenido del registro f de RAM menos lo que haya en el acumulador W, la resta se lleva a cabo haciendo una suma entre el registro f y el complemento A2 de lo que haya en el acumulador W y después:
 - Si d = 0 el resultado se guarda en el acumulador W.
 - ⊕ Aunque en vez de poner 0, puedo poner directamente W.
 - Si d = 1 el resultado se guarda en el registro f que se utilizó en la instrucción.
 - ⊕ Aunque en vez de poner 1, puedo poner directamente f.
- b. Se afecta a las banderas C = acarreo, DC= acarreo decimal y Z = ceros.
 - Si C = 1 el resultado de la resta fue positivo.
 - Si C = 0 el resultado de la resta fue negativo.
 - Si Z =1 el resultado fue cero, 0000 0000 en binario o 00 en hexadecimal.

Handwritten notes showing two subtraction examples:

- Example 1: $4 - 3 = 1$. A box contains "ocurro = 1" and "positivo".
- Example 2: $-3 - (-4) = 9999$. A box contains "ocurro = 0" and "negativo".

- c. Se usa el complemento A2 para saber la magnitud correcta de un número binario negativo.
 - Este se obtiene buscando de izquierda a derecha el primer 1 y a partir de este cambiar los 0 por 1 y los 1 por 0, sin cambiar el primer 1 encontrado.
- d. Tarda 1 ciclo de máquina en ejecutarse.

$$\text{Resultado}_{SUBWF} = (\text{Registro}_f) - W$$

$$\text{Resultado}_{SUBWF} = (\text{Registro}_f) + C_2^2 W = (\text{Registro}_f) + C_{A2} W$$

17. SWAPF f, d:

- a. Lee el contenido de un registro de RAM f dado e intercambia entre sí los 2 grupos de 4 bits (llamados nibbles) que haya en el número binario de 8 bits, viéndolo desde un punto de vista hexadecimal simplemente cambia de lugar sus valores.
 - Si d = 0 el resultado se guarda en el acumulador W.
 - ⊕ Aunque en vez de poner 0, puedo poner directamente W.
 - Si d = 1 el resultado se guarda en el registro f que se utilizó en la instrucción.
 - ⊕ Aunque en vez de poner 1, puedo poner directamente f.
- b. No afecta ninguna bandera.
- c. Tarda 1 ciclo de máquina en ejecutarse.
 - Ejemplo: Si en el registro f está el valor hexadecimal **6A**, al aplicar la instrucción ahora tendremos el valor **A6**.

18. XORWF f, d (Z):

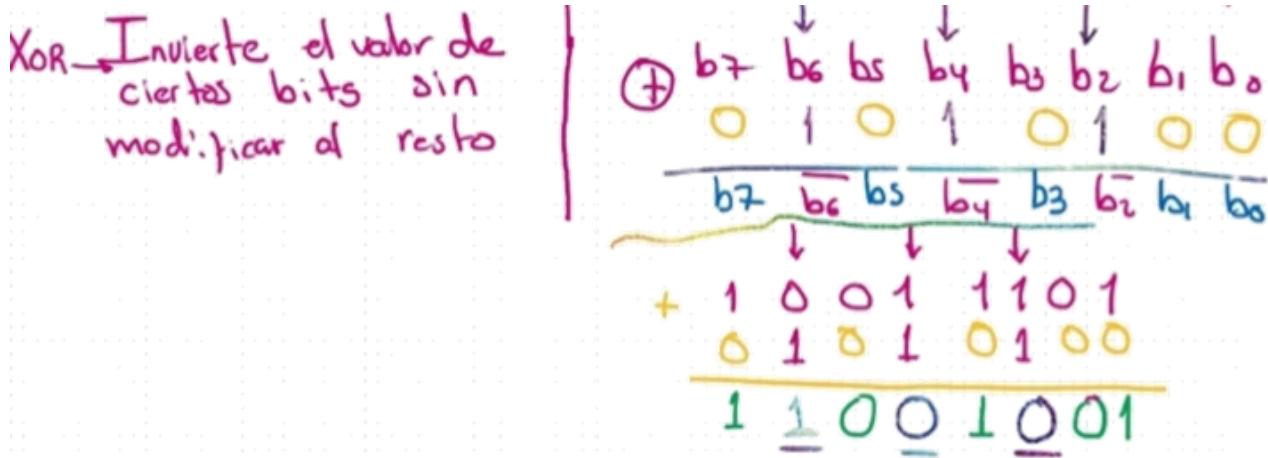
- a. Realiza la función lógica XOR ("O" exclusiva) entre lo que haya en la dirección f del registro en la RAM y el acumulador W.
 - Si d = 0 el resultado se guarda en el acumulador W.
 - ⊕ Aunque en vez de poner 0, puedo poner directamente W.
 - Si d = 1 el resultado se guarda en el registro f que se utilizó en la instrucción.
 - ⊕ Aunque en vez de poner 1, puedo poner directamente f.

A esto se le llama máscara XOR y lo que hará es obligar a ciertos bits del registro f que inviertan su valor, dejando a los demás intactos. La máscara en este caso se colocaría previamente en el acumulador W, poniendo un número binario de 8 bits que tenga un 1 donde quiero que se obligue a los bits del registro f que se inviertan (si eran 0 que se hagan 1 y si eran 1 que se hagan 0) y dejando un 0 en la posición de los bits del registro f que no quiero afectar.

Funciones de los bits en la máscara XOR:

- 0: No afecta en nada, deja los bits en su forma inicial.
- 1: Obliga a los bits a invertirse (si era 1 se vuelve 0 y si era 0 se vuelve 1).

- b. Solo se afecta a la bandera Z = ceros.
- c. Tarda 1 ciclo de máquina en ejecutarse.



19. BCF f, b:

- a. Escribe un 0 en el bit b de la dirección f del registro RAM.
 - La posición b del bit en el registro f se indica contando desde cero, osea poniendo 0, 1, 2, 3, 4, 5, 6 o 7.
 - La posición b se indica en la instrucción con un número decimal, se debe hacer sin usar el punto como se hace para declarar registros en forma decimal (.número_decimal), con poner el simple número basta.
 - Si conozco el nombre (directiva EQU) del bit que quiero afectar en mi registro de propósito específico como por ejemplo los bits RP1 y RP0 del registro STATUS o el nombre de la bandera C, DC o Z puedo ponerlo directamente.
- b. No afecta ninguna bandera.
- c. Si quiero cambiar más de 1 bit a cero, es mejor utilizar una máscara a usar esta instrucción.
- d. Cualquier bit b de los registros de propósito general lo puedo llenar de ceros sin problema.
- e. Pero hay algunos registros de propósito específico (los que tienen nombre) que no se pueden llenar todos sus bits de ceros, aunque se lo indique con esta instrucción, puede que solo se puedan poner algunos como 0 o puede que ninguno se pueda poner como cero, esto viene indicado desde la página 29 hasta la 36 del datasheet para cada registro individualmente, dependiendo de si los bits del registro están marcados como:
 - R = Read (leíble): Estos bits no los puedo cambiar, solo leer.
 - W = Write (escribible): Estos bits si los puedo cambiar (o escribir que es lo mismo).
 - R/W = Read/Write (leíble/escribible): Estos bits los puedo leer y escribir.
- f. Tarda 1 ciclo de máquina en ejecutarse.

1 byte = 8 bits							
Nibble alto				Nibble bajo			
0	1	1	0	0	1	1	1
↑	↑	↑	↑	↑	↑		
0	1	0	1	0			
BCF	BSF	BCF	BSF	BCF			
Posición de los bits		6	5	4	3	2	

- g. Es muy común usar esta instrucción para alcanzar registros que se encuentren en otro banco, cambiando los valores de los bits **RP1** y **RPO** del registro STATUS de la siguiente manera dependiendo del banco (también llamado página) a donde me quiera dirigir.

RP1 RP0

0 0 →Bank 0 is selected

0 1 →Bank 1 is selected

1 0 →Bank 2 is selected

1 1 →Bank 3 is selected

20. **BSF** f, b:

- a. Pone un 1 en el bit b de la dirección f del registro RAM.
 - La posición b del bit en el registro f se indica contando desde cero, osea poniendo 0, 1, 2, 3, 4, 5, 6 o 7.
 - ◆ La posición b se indica en la instrucción con un número decimal, se debe hacer sin usar el punto como se hace para declarar registros en forma decimal (.número_decimal), con poner el simple número basta.
 - Si conozco el nombre (directiva EQU) del bit que quiero afectar en mi registro de propósito específico como por ejemplo los bits **RP1** y **RPO** del registro STATUS o el nombre de la bandera C, DC o Z puedo ponerlo directamente.
- b. No afecta ninguna bandera.
- c. Si quiero cambiar más de 1 bit a uno, es mejor utilizar una máscara a usar esta instrucción.
- d. Cualquier bit b de los registros de propósito general lo puedo llenar de unos sin problema.
- e. Pero hay algunos registros de propósito específico (los que tienen nombre) que no se pueden llenar todos sus bits de unos, aunque se lo indique con esta instrucción, puede que solo se puedan poner algunos como 1 o puede que ninguno se pueda poner como 1, esto viene indicado desde la página 29 hasta la 36 del datasheet para cada registro individualmente, dependiendo de si los bits del registro están marcados como:
 - **R** = Read (leíble): Estos bits no los puedo cambiar, solo leer.
 - **W** = Write (escribible): Estos bits si los puedo cambiar (o escribir que es lo mismo).
 - **R/W** = Read/Write (leíble/escribible): Estos bits los puedo leer y escribir.

- f. Tarda 1 ciclo de máquina en ejecutarse.

1 byte = 8 bits							
Nibble alto				Nibble bajo			
Hex: 0x2B	00101011	←	0	1	1	0	0
			↑	↑	↑	↑	↑
			0	1	0	1	0
			BCF	BSF	BCF	BSF	BCF
Posición de los bits		6	5	4	3	2	

- g. Es muy común usar esta instrucción para alcanzar registros que se encuentren en otro banco, cambiando los valores de los bits **RP1** y **RP0** del registro STATUS de la siguiente manera dependiendo del banco (también llamado página) a donde me quiera dirigir.

RP1 RP0

0 0 →Bank 0 is selected

0 1 →Bank 1 is selected

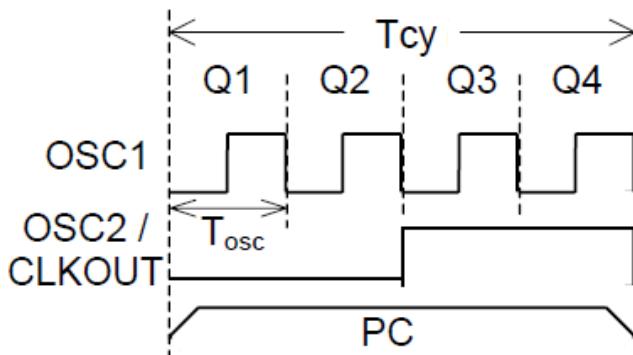
1 0 →Bank 2 is selected

1 1 →Bank 3 is selected

21. BTFSC f, b:

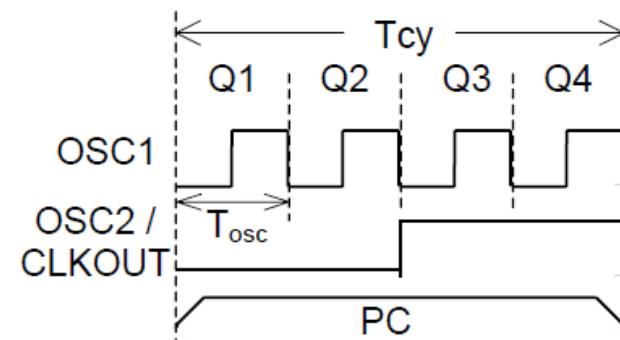
- a. Esta operación es lo más parecido a un condicional if que existe en el idioma ensamblador, normalmente se usa para evitar instrucciones GOTO o para evaluar el estado de banderas.
- b. Su condición la ejecuta evaluando si el bit b del registro f es uno o cero y si es 0 se brinca la siguiente instrucción, sino sigue la ejecución normal del programa.
 - La posición b del bit en el registro f se indica contando desde cero, osea poniendo 0, 1, 2, 3, 4, 5, 6 o 7.
 - La posición b se indica en la instrucción con un número decimal, se debe hacer sin usar el punto como se hace para declarar registros en forma decimal (.número_decimal), con poner el simple número basta.
 - Si conozco el nombre (directiva EQU) del bit que quiero afectar en mi registro de propósito específico como por ejemplo los bits **RP1** y **RP0** del registro STATUS o el nombre de la bandera C, DC o Z puedo ponerlo directamente.
- c. Analiza el bit que haya en la posición b del registro f de RAM y dependiendo si el bit es cero o no, se brincará la siguiente instrucción o si la ejecutará normalmente.
 - Si el bit b es cero (0):
 - Se brinca la siguiente instrucción que haya en el código (aumenta en 1 al contador de programa).
 - Tarda 2 ciclos de máquina en ejecutarse.
 - Si el bit b es uno (1):
 - Sigue la ejecución normal del código.
 - Tarda 1 ciclo de máquina en ejecutarse.

- d. No afecta ninguna bandera.



22. BTFSS f, b:

- a. Esta operación es lo más parecido a un condicional if que existe en el idioma ensamblador, normalmente se usa para evitar instrucciones GOTO o para evaluar el estado de banderas.
- b. Su condición la ejecuta evaluando si el bit b del registro f es uno o cero y si es 1 se brinca la siguiente instrucción, sino sigue la ejecución normal del programa.
 - La posición b del bit en el registro f se indica contando desde cero, osea poniendo 0, 1, 2, 3, 4, 5, 6 o 7.
 - La posición b se indica en la instrucción con un número decimal, se debe hacer sin usar el punto como se hace para declarar registros en forma decimal (.número_decimal), con poner el simple número basta.
 - Si conozco el nombre (directiva EQU) del bit que quiero afectar en mi registro de propósito específico como por ejemplo los bits RP1 y RP0 del registro STATUS o el nombre de la bandera C, DC o Z puedo ponerlo directamente.
- c. Analiza el bit que haya en la posición b del registro f de RAM y dependiendo si el bit es cero o no, se brincará la siguiente instrucción o si la ejecutará normalmente.
 - Si el bit b es cero (0):
 - Sigue la ejecución normal del código.
 - Tarda 1 ciclo de máquina en ejecutarse.
 - Si el bit b es uno (1):
 - Se brinca la siguiente instrucción que haya en el código (aumenta en 1 al contador de programa).
 - Tarda 2 ciclos de máquina en ejecutarse.
- d. No afecta ninguna bandera.



23. ADDLW k (C, DC, Z):

- a. Suma el contenido del acumulador W con un valor cualquiera de 8 bits indicado por el valor literal k y el resultado lo guarda en el acumulador W.
 - El resultado siempre se guarda en el acumulador, esta es una condición de las instrucciones literales (L).
- b. Se afecta a las banderas C = acarreo, DC= acarreo decimal y Z = ceros.
- c. Tarda 1 ciclo de máquina en ejecutarse.

24. ANDLW k (Z):

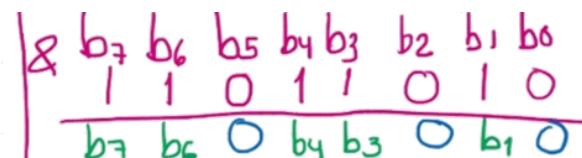
- a. Realiza la función lógica AND entre un valor literal k (número binario cualquiera de 8 bits) y lo que haya en el acumulador W.
 - El resultado siempre se guarda en el acumulador, esta es una condición de las instrucciones literales (L).

A esto se le llama **máscara AND** y lo que hará es obligar a ciertos bits del valor literal k que se vuelvan 0, dejando a los demás intactos. La máscara en este caso se colocaría previamente en el acumulador W o en el valor literal k, poniendo un número binario de 8 bits que tenga un 0 donde quiero que se obligue a los bits del otro número binario que se vuelvan 0 y dejando un 1 en la posición de los bits que no quiero afectar.

Funciones de los bits en la máscara AND:

- 0: obliga a los bits a volverse 0.
- 1: No afecta en nada, deja los bits en su forma inicial.

- b. Solo se afecta a la bandera Z = ceros.
- c. Tarda 1 ciclo de máquina en ejecutarse.

And. Obliga a ciertos bits a ser 0's sin tocar al resto | 

25. CALL k:

- a. Sirve para ejecutar una subrutina, una subrutina es el equivalente a una función en cualquier otro lenguaje de programación.
- b. Las subrutinas se usan para ejecutar una acción que se necesita recurrentemente en el código, siempre se ejecutan con la instrucción CALL y terminan con la instrucción RETURN.
- c. La forma en la que se llama una subrutina es la siguiente:
 - Primero debemos haber asignado un nombre a cierto nivel del código donde empieza la subrutina.
 - Luego ya podremos llamarla usando la instrucción CALL junto con el nombre que anteriormente asignamos a la línea de código donde empieza la subrutina.
- d. El valor literal k solamente se usa en las instrucciones CALL, GOTO, RETFIE y RETLW representa una dirección de FLASH y es de 11 bits (ya que en todas las demás solo soporta 8 bits), el número de bits cambia en estas instrucciones porque las direcciones de FLASH requieren 13 bits, pero como k solo soporta 11 bits, los 2 faltantes los obtiene de los bits 3 y 4 del registro PCLATH (que inicialmente valen 00), esto solo es necesario modificarlo cuando el salto que vamos a hacer con la instrucción GOTO es muy grande (alrededor de 2 mil instrucciones atrás, osea 2k registros de FLASH), los registros PCLATH se encuentran

en los siguientes registros dependiendo del banco o página donde nos encontremos de la RAM y en pocas palabras se usan simplemente para brincar de un banco a otro:

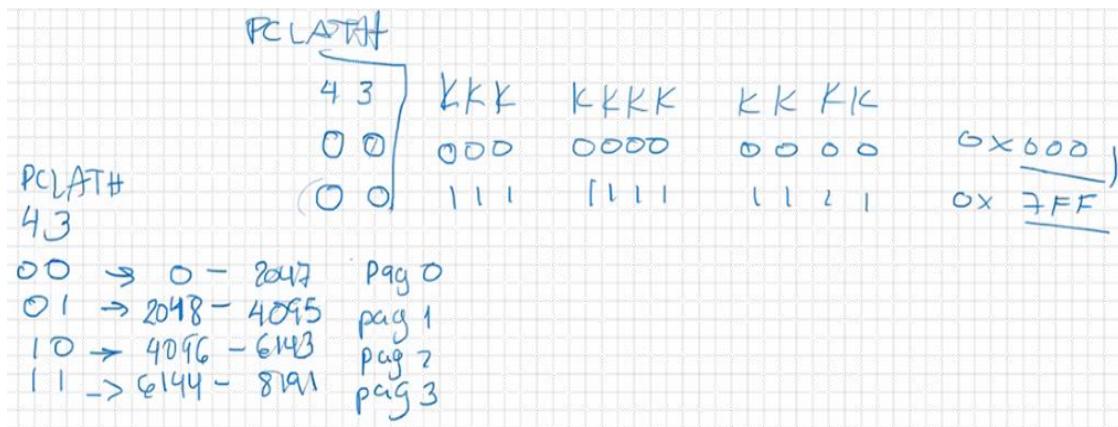
- **Registro PCLATH Banco 0:** Dirección de memoria RAM **0Ah**, 0X0A o 0A hexadecimal que es lo mismo a decir 0000 1010 binario.
- **Registro PCLATH Banco 1:** Dirección de memoria RAM **8Ah**, 0X8A u 8A hexadecimal que es lo mismo a decir 1000 1010 binario.
- **Registro PCLATH Banco 2:** Dirección de memoria RAM **10Ah**, 0X10A o 10A hexadecimal que es lo mismo a decir 0001 0000 1010 binario.
- **Registro PCLATH Banco 3:** Dirección de memoria RAM **18Ah**, 0X18A o 18A hexadecimal que es lo mismo a decir 0001 1000 1010 binario.

- e. **Tarda 2 ciclos de máquina en ejecutarse.**
- f. **Tiene acceso a la Pila de la memoria RAM.**

Los bits 3 y 4 del registro PCLATH son los mismos que los bits RP1 y RP2 del registro STATUS y lo que hacen es permitirme brincar de un banco (o página) a otro, a esto se le llama paginación y en cada página se cuenta con un número de registros específicos que puedo alcanzar sin salirme de esa página o banco.

Si pongo un número binario que no es soportado por los bits PCLATH del banco en donde estoy (osea que la dirección de registro es más grande de las que hay en el banco), el último bit el programa hará como que no existe y me mandará a la correspondiente dirección de registro que si existe en el banco donde me encuentro sin tomar en cuenta ese último bit.

- Por ejemplo, si me encuentro en el banco 1 y me quiero ir a un registro mayor a 7Fh, como por ejemplo el 10Ah, el programa me mandará al registro 0Ah, porque los últimos 4 bits del número hexadecimal que puse como dirección de registro no son soportados por el banco 1.



- g. En el programa se llama a la subrutina siguiendo los pasos:
 - El valor del **contador del programa (PC o Program Counter)** actual se guarda en una dirección de la **Pila**, indicado por el **Apuntador de la Pila (o STACK POINTER en inglés)**.
 - El apuntador de la pila se incrementa en 1.
 - El valor del **PC** se carga con la dirección del valor literal k en donde empieza la subrutina, saltando así a la parte del programa que la describe.

FIGURE 2-6: PIC16F886/PIC16F887 SPECIAL FUNCTION REGISTERS

	File	File	File	File
	Address	Address	Address	Address
Indirect addr. (1)	00h	Indirect addr. (1)	80h	Indirect addr. (1)
TMR0	01h	OPTION_REG	81h	TMR0
PCL	02h	PCL	82h	PCL
STATUS	03h	STATUS	83h	STATUS
FSR	04h	FSR	84h	FSR
PORTA	05h	TRISA	85h	WDTCON
PORTB	06h	TRISB	86h	PORTB
PORTC	07h	TRISC	87h	CM1CON0
PORTD ⁽²⁾	08h	TRISD ⁽²⁾	88h	CM2CON0
PORTE	09h	TRISE	89h	CM2CON1
PCLATH	0Ah	PCLATH	8Ah	PCLATH
INTCON	0Bh	INTCON	8Bh	INTCON
PIR1	0Ch	PIE1	8Ch	EEDAT
PIR2	0Dh	PIE2	8Dh	EEADDR
TMR1L	0Eh	PCON	8Eh	EEDATH
TMR1H	0Fh	OSCCON	8Fh	EEADRH
T1CON	10h	OSCTUNE	90h	
TMR2	11h	SSPCON2	91h	
T2CON	12h	PR2	92h	
SSPBUF	13h	SSPADD	93h	
SSPCON	14h	SSPSTAT	94h	
CCPR1L	15h	WPUB	95h	
CCPR1H	16h	IOCB	96h	General
CCP1CON	17h	VRCON	97h	Purpose
RCSTA	18h	TXSTA	98h	Registers
TXREG	19h	SPBRG	99h	16 Bytes
RCREG	1Ah	SPBRGH	9Ah	
CCPR2L	1Bh	PWM1CON	9Bh	
CCPR2H	1Ch	ECCPAS	9Ch	
CCP2CON	1Dh	PSTRCON	9Dh	
ADRESH	1Eh	ADRESL	9Eh	
ADCON0	1Fh	ADCON1	9Fh	
General Purpose Registers	20h	General Purpose Registers	A0h	General Purpose Registers
96 Bytes	3Fh	80 Bytes		80 Bytes
	40h			
	6Fh			1EFh
	70h	accesses 70h-7Fh		1F0h
	7Fh			1FFh
Bank 0	Bank 1		Bank 2	Bank 3

26. CLRWDT :

- a. Sirve para limpiar el perro guardián (le escribe el valor 0x00), pero como en el curso lo vamos a apagar al watchdog porque no vamos a tener funciones tan robustas, osea de largos periodos de trabajo 24/7 o en condiciones ambientales o eléctricas extremas, no lo vamos a utilizar.

27. GOTO k:

- a. Sirve para hacer que el programa brinque a otra parte del programa indicado por el parámetro literal k, la parte a donde quiero que brinque el programa se puede indicar por nombre, por número del contador de programa (PC) o por número de instrucciones hacia atrás de la instrucción GOTO.
- b. Esto se debe utilizar siempre en los programas hechos en ensamblador ya que como la función del microcontrolador es buscar y ejecutar instrucciones indefinidamente, su código se debe repetir una y otra vez, pero debemos pensar bien a qué parte del código queremos saltar para que no se realicen acciones innecesarias durante la ejecución del microcontrolador, podemos saltar a cualquier parte del código de la siguiente manera:
 - Poniéndole un nombre a una parte del código, desde esa parte del código se estará ejecutando una acción indefinidamente.
 - Indicando el valor del contador del programa que corresponde a la instrucción a la cual queremos saltar.
 - Poniendo el símbolo de pesos seguido del signo menos y el número de instrucciones hacia atrás de la instrucción GOTO que quiero que se ejecute indefinidamente.
- c. El valor literal k solamente en las instrucciones CALL, GOTO, RETFIE y RETLW representa una dirección de FLASH y es de 11 bits (ya que en todas las demás solo soporta 8 bits), el número de bits cambia en estas instrucciones porque las direcciones de FLASH requieren 13 bits, pero como k solo soporta 11 bits, los 2 faltantes los obtiene de los bits 3 y 4 del registro PCLATH (que inicialmente valen 00), esto solo es necesario modificarlo cuando el salto que vamos a hacer con la instrucción GOTO es muy grande (alrededor de 2 mil instrucciones atrás, osea 2k registros de FLASH), los registros PCLATH se encuentran en los siguientes registros dependiendo del banco o página donde nos encontramos de la RAM y en pocas palabras se usan simplemente para saltar de un banco a otro:
 - Registro PCLATH Banco 0: Dirección de memoria RAM 0Ah, 0X0A o 0A hexadecimal que es lo mismo a decir 0000 1010 binario.
 - Registro PCLATH Banco 1: Dirección de memoria RAM 8Ah, 0X8A u 8A hexadecimal que es lo mismo a decir 1000 1010 binario.
 - Registro PCLATH Banco 2: Dirección de memoria RAM 10Ah, 0X10A o 10A hexadecimal que es lo mismo a decir 0001 0000 1010 binario.
 - Registro PCLATH Banco 3: Dirección de memoria RAM 18Ah, 0X18A o 18A hexadecimal que es lo mismo a decir 0001 1000 1010 binario.
- d. Tarda 2 ciclos de máquina en ejecutarse.



FIGURE 2-6: PIC16F886/PIC16F887 SPECIAL FUNCTION REGISTERS

File Address	File Address	File Address	File Address	File Address
Indirect addr. ⁽¹⁾ 00h	Indirect addr. ⁽¹⁾ 01h	Indirect addr. ⁽¹⁾ 00h	Indirect addr. ⁽¹⁾ 01h	Indirect addr. ⁽¹⁾ 100h
TMR0 01h	OPTION_REG 02h	PCL 03h	STATUS 04h	TMR0 101h
PCL 02h	PCL 05h	STATUS 05h	FSR 06h	PCL 102h
STATUS 03h	TRISA 07h	STATUS 07h	FSR 08h	STATUS 103h
FSR 04h	TRISB 08h	FSR 08h	WDTCON 09h	FSR 104h
PORTA 05h	TRISC 09h	PORTB 06h	PORTB 09h	SRCON 105h
PORTB 06h	TRISD ⁽²⁾ 0Ah	PORTC 07h	CM1CON0 0Ah	TRISB 106h
PORTC 07h	TRISD ⁽²⁾ 0Ah	PORTC 07h	CM2CON0 0Ah	BAUDCTL 107h
PORTD ⁽²⁾ 08h	TRISE 09h	PORTD ⁽²⁾ 08h	CM2CON1 0Ah	ANSEL 108h
PORTE 09h	PCLATH 0Ah	PORTE 09h	PCLATH 0Ah	ANSELH 109h
PCLATH 0Ah	PCLATH 0Ah	PCLATH 0Ah	PCLATH 10Ah	PCLATH 10Ah
INTCON 0Bh	INTCON 0Ch	INTCON 0Bh	INTCON 0Bh	INTCON 10Bh
PIR1 0Ch	PIE1 0Dh	PIE2 0Eh	EEDAT 0Dh	EECON1 10Ch
PIR2 0Dh	PCON 0Fh	OSCCON 10h	EEADR 0Eh	EECON2 ⁽¹⁾ 10Dh
TMR1L 0Eh	OSCTUNE 11h	SSPCON2 12h	EEDATH 0Fh	Reserved 10Eh
TMR1H 0Fh	SSPADD 13h	PR2 12h	EEADDRH 10Fh	Reserved 10Fh
T1CON 10h	SSPSTAT 14h	90h		
TMR2 11h	WPUB 15h	91h		190h
T2CON 12h	IOCB 16h	92h		191h
SSPBUF 13h	VRCON 17h	93h		192h
SSPCON 14h	TXSTA 18h	94h		193h
CCPR1L 15h	SPBRG 19h	95h		194h
CCPR1H 16h	SPBRGH 1Ah	96h	General Purpose Registers	195h
CCP1CON 17h	PWM1CON 1Bh	97h	16 Bytes	196h
RCSTA 18h	ECCPAS 1Ch	98h		197h
TXREG 19h	PSTRCON 1Dh	99h		198h
RCREG 1Ah	ADRESL 1Eh	9Ah		16 Bytes
CCPR2L 1Bh	ADCON1 1Fh	9Bh		199h
CCPR2H 1Ch	General Purpose Registers 3Fh	9Ch		19Ah
CCP2CON 1Dh	80 Bytes 40h	9Dh		19Bh
ADRESH 1Eh	80 Bytes 6Fh	9Eh		19Ch
ADCON0 1Fh	accesses 70h-7Fh 7Fh	9Fh		19Dh
		A0h	General Purpose Registers 80 Bytes	19Eh
General Purpose Registers 96 Bytes			80 Bytes	19Fh
			80 Bytes	1A0h
			accesses 70h-7Fh EFh	General Purpose Registers 80 Bytes 16Fh
			accesses 70h-7Fh F0h	80 Bytes 170h
			accesses 70h-7Fh FFh	80 Bytes 17Fh
				accesses 70h-7Fh 1EFh
				accesses 70h-7Fh 1F0h
				accesses 70h-7Fh 1FFh

Los bits 3 y 4 del registro PCLATH son los mismos que los bits RP1 y RP2 del registro STATUS y lo que hacen es permitirme brincar de un banco (o página) a otro, a esto se le llama paginación y en cada página se cuenta con un número de registros específicos que puedo alcanzar sin salirme de esa página o banco.

Si pongo un número binario que no es soportado por los bits PCLATH del banco en donde estoy (osea que la dirección de registro es más grande de las que hay en el banco), el último bit el programa hará como que no existe y me mandará a la correspondiente dirección de registro que si existe en el banco donde me encuentro sin tomar en cuenta ese último bit.

- Por ejemplo, si me encuentro en el banco 1 y me quiero ir a un registro mayor a 7Fh, como por ejemplo el 10Ah, el programa me mandará al registro 0Ah, porque los últimos 4 bits del número hexadecimal que puse como dirección de registro no son soportados por el banco 1.

PCLATH		KKK	KKKK	KK KK	$\frac{6 \times 600}{}$
43	00	KKK	KKKK	KK KK	
00	00	000	0000	00 00	0x 7FF
43	00	111	1111	11 11	
00	00	000	0000	00 00	
01	01	001	0001	00 01	
10	10	010	0010	01 00	
11	11	011	0011	01 11	
		000	0000	00 00	
		001	0001	00 01	
		010	0010	01 00	
		011	0011	01 11	
		100	0100	10 00	
		101	0101	10 01	
		110	0110	10 10	
		111	0111	10 11	

00 → 0 - 2047 pag 0
 01 → 2048 - 4095 pag 1
 10 → 4096 - 6143 pag 2
 11 → 6144 - 8191 pag 3

28. IORLW k (Z):

- Ejecuta la compuerta lógica OR entre un valor literal k (número binario cualquiera de 8 bits) y lo que haya en el acumulador W.
 - El resultado siempre se guarda en el acumulador, esta es una condición de las instrucciones literales (L).

A esto se le llama **máscara OR** y lo que hará es obligar a ciertos bits del valor literal k que se vuelvan 1, dejando a los demás intactos. La máscara en este caso se colocaría previamente en el acumulador W o en el valor literal k, poniendo un número binario de 8 bits que tenga un 0 donde quiero que se obligue a los bits del otro número binario que se vuelvan 1 y dejando un 0 en la posición de los bits que no quiero afectar.

Funciones de los bits en la máscara OR:

- 0: No afecta en nada, deja los bits en su forma inicial.
- 1: Obliga a los bits a volverse 1.

- Solo se afecta a la bandera Z = ceros.
- Tarda 1 ciclo de máquina en ejecutarse.



29. MOVLW k:

- Coloca directamente en el acumulador W un número cualquier guardado en el valor literal k de 8 bits.
- Esto se hace para utilizar al acumulador como punto intermedio de ingreso de datos al programa.

c. **Tarda 1 ciclo de máquina en ejecutarse.**

30. RETFIE :

a. Hace que el programa regrese de una interrupción siguiendo los pasos:

- Decrementa en 1 al apuntador de la Pila.
- El contador del programa (PC) se carga con el contenido de la dirección de la pila que indica el apuntador de la Pila (que previamente ha sido decrementado).
- El permiso global de interrupciones (GIE) se activa.

b. **Tarda 2 ciclos de máquina en ejecutarse.**

31. RETLW k:

a. Siempre se debe declarar dentro de una subrutina y lo que hace es cargar en el acumulador W un número de 8 bits escrito en el valor literal k y luego saca al programa de la subrutina donde se encuentre por medio de los siguientes pasos:

- El acumulador W se carga con el valor del registro k.
- Se decrementa en 1 al apuntador de la Pila.
- El contador del programa (PC) se carga con el contenido de la dirección de la Pila que indica el apuntador de la Pila (que previamente ha sido decrementado).

b. **RETLW = MOVLW + RETURN.**

c. Tiene que ver con el tema de tablas.

d. **Tarda 2 ciclos de máquina en ejecutarse.**

32. RETURN :

a. Sirve para terminar una subrutina, una subrutina es el equivalente a una función en cualquier otro lenguaje de programación. La instrucción logra regresar de una subrutina a través de los siguientes pasos:

- Se decrementa en 1 al apuntador de la Pila.
- El contador del programa (PC) se carga con el contenido de la dirección de la Pila que indica el apuntador de la pila (que previamente ha sido decrementado).

b. Las subrutinas se usan para ejecutar una acción que se necesita recurrentemente en el código y siempre se ejecutan con la instrucción CALL y se terminan con la instrucción RETURN.

- ¡SIEMPRE DEBO AGREGAR LA INSTRUCCIÓN RETURN AL FINAL DE UNA SUBRUTINA!

c. **Tarda 2 ciclos de máquina en ejecutarse.**

d. **Tiene acceso a la Pila de la memoria RAM.**

33. SLEEP :

a. Duerme al microcontrolador, lo pone en un modo de bajo consumo de energía.

- La función fundamental de un microprocesador es buscar y ejecutar instrucciones, no existe la instrucción párate o deja de funcionar para el PIC, por lo tanto, la instrucción SLEEP lo que hace es apagar el oscilador para que el PIC ya no tenga esta herramienta fundamental y no pueda funcionar, pero eso no significa que deje de buscar instrucciones, aunque no tenga el oscilador lo seguirá intentando, se quedará bloqueado buscando instrucciones en sí mismo.

b. El microcontrolador puede despertar después de haber ejecutado la instrucción SLEEP:

- Cuando se accione de forma externa el pin de reinicio MCLR.
- Cuando se despierte por medio del temporizador del Watchdog, esto solamente si el registro WDT fue activado.

- Cuando ocurra una interrupción en el pin RB0/INT, osea que se detecte un cambio en el puerto B o una interrupción externa de un pin.
 - ✚ El primer evento (MCLR) provocará un reinicio del dispositivo, mientras que los dos últimos eventos se consideran como una forma de continuar con la ejecución del programa. Los bits TO y PD del registro STATUS pueden utilizarse para determinar la causa del reinicio del dispositivo.
 - a. El bit PD, que se activa en el encendido del microcontrolador, se borra cuando se invoca la instrucción SLEEP y el bit TO se borra si se ha producido un cambio en el registro WDT.
 - ✚ También los siguientes periféricos pueden despertar el dispositivo después de haber ejecutado la instrucción SLEEP:
 - a. Interrupción TMR1, para ello el Timer1 debe estar funcionando como contador asíncrono.
 - b. Interrupción configurada en el modo de captura ECCP.
 - c. Conversión analógica digital (cuando la fuente de reloj A/D es el registro FRC).
 - d. Finalización de la operación de escritura en el EEPROM.
 - e. La salida del comparador cambia de estado.
 - f. Interrupción por cambio.
 - g. Interrupción externa en el pin INT.
 - h. Detección de ruptura EUSART, esclavo I2C.

34. SUBLW k (C, DC, Z):

- a. Realiza una resta entre el valor literal k (número binario cualquiera de 8 bits que no tiene nada que ver con la memoria RAM) y lo que sea que haya en el acumulador W, **la resta se lleva a cabo haciendo una suma entre el número binario k de 8 bits y el complemento A2 de lo que haya en el acumulador W.**
 - El resultado siempre se guarda en el acumulador, esta es una condición de las instrucciones literales (L).
- b. Se afecta a las banderas C = acarreo, DC= acarreo decimal y Z = ceros.
 - Si C = 1 el resultado de la resta fue positivo.
 - Si C = 0 el resultado de la resta fue negativo.
 - Si Z = 1 el resultado fue cero, 0000 0000 en binario o 00 en hexadecimal.
- c. Se usa el complemento A2 para saber la magnitud correcta de un número binario negativo.
 - La magnitud de la resta se obtiene buscando en el número de 8 bits de izquierda a derecha el primer 1 y a partir de este bit se cambian los 0 por 1 y los 1 por 0, sin cambiar el primer 1 encontrado.
- d. **Tarda 1 ciclo de máquina en ejecutarse.**

$$\text{Resultado}_{SUBWF} = (\text{Registro}_k) - W$$

$$\text{Resultado}_{SUBWF} = (\text{Registro}_k) + C^2_2 W = k + C_{A2} W$$

35. XORLW k (Z):

- a. Realiza la función lógica XOR entre un valor literal k (número binario cualquiera de 8 bits) y lo que haya en el acumulador W.

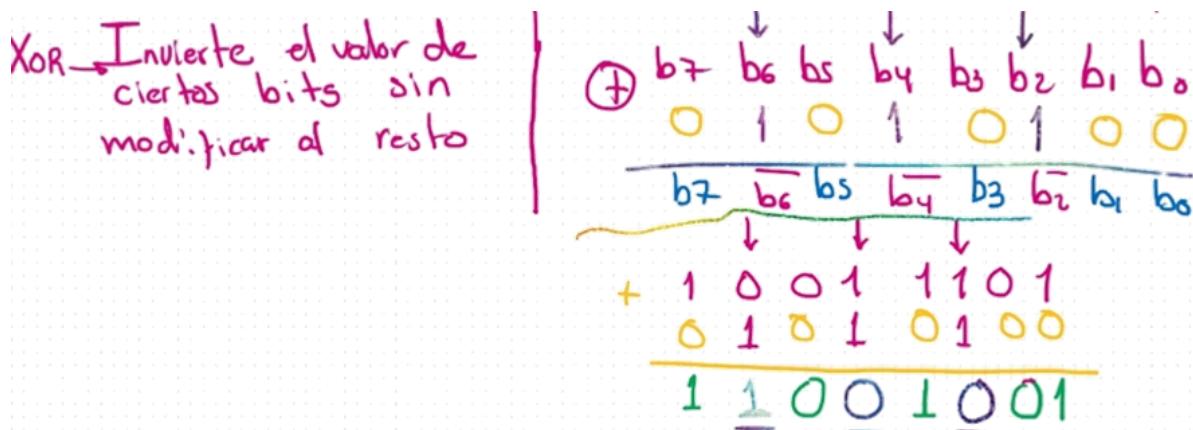
- El resultado siempre se guarda en el acumulador, esta es una condición de las instrucciones literales (L).
- Cuando el valor k y lo que haya en el acumulador W sean iguales, el resultado siempre será cero y la bandera Z se levantará, osea será igual a 1.

A esto se le llama **máscara XOR** y lo que hará es obligar a ciertos bits del valor literal k que se inviertan, dejando a los demás intactos. La máscara en este caso se colocaría previamente en el acumulador W o en el valor literal k, poniendo un número binario de 8 bits que tenga un 1 donde quiero que se obligue a los bits del otro número binario que se inviertan (si eran 0 que se hagan 1 y si eran 1 que se hagan 0) y dejando un 0 en la posición de los bits que no quiero afectar.

Funciones de los bits en la máscara XOR:

- 0: No afecta en nada, deja los bits en su forma inicial.
- 1: Obliga a los bits a invertirse (si era 1 se vuelve 0 y si era 0 se vuelve 1).

- Solo se afecta a la bandera Z = ceros.
- Tarda 1 ciclo de máquina en ejecutarse.



Directivas:

Las directivas sirven para configurar algunos aspectos del PIC16F887 y aunque ejecutan acciones, no es lo mismo a una instrucción, en total existen 35 instrucciones en el microcontrolador, todas las acciones que escribamos en el código que no pertenezcan a esas 35, se llaman directivas y no son instrucciones para el microcontrolador sino para el software que se comunica con él.

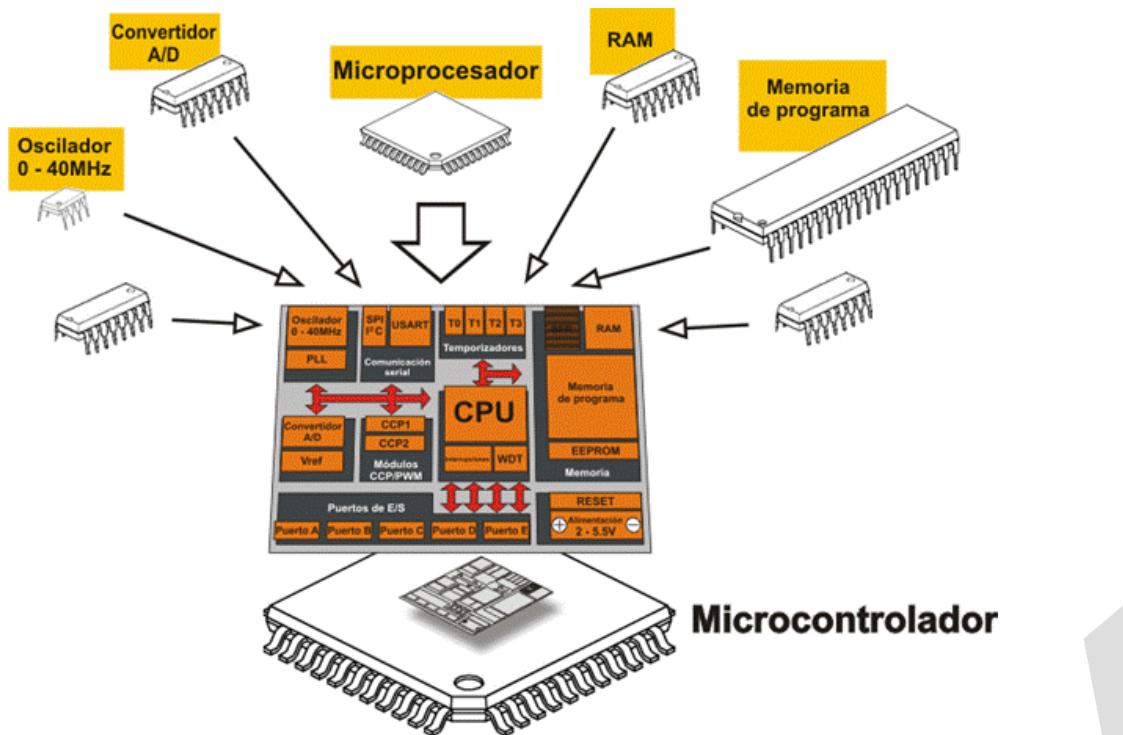
1. PROCESSOR NOMBRE_DEL_PIC:
 - Esta directiva debe ser la primera que venga en el programa y sirve para indicar qué dispositivo estoy programando.
 - En este caso se pone la directiva **PROCESSOR 16F887** porque es el PIC que estamos programando.
2. __CONFIG PALABRA DE CONFIGURACIÓN:
 - La directiva **__CONFIG** le indica al procesador qué palabras de configuración se deben quemar en la parte externa de la memoria Flash que es solo de escritura (memoria ROM), llamada **memoria de configuración**.

- **Palabra de configuración:** Es un número binario de 14 bits, donde dependiendo de lo que ponga en cada uno de sus bits, se le dará una configuración diferente a cada uno de los puertos del PIC.

- **Para configurar el PIC16F887 se deben declarar 2 palabras de configuración diferentes, que están dirigidas a las siguientes direcciones de la memoria de configuración, que es parte de la memoria ROM del PIC16F887:**

- + Dirección de la memoria FLASH **2007h**, 0X2007 o 2007 hexadecimal que proporciona los 13 bits de direccionamiento.
- + Dirección de la memoria FLASH **2008h**, 0X2008 o 2008 hexadecimal que proporciona los 13 bits de direccionamiento.

- Los registros **2008h** y **2008h** en el PIC sirven para cargar el programa y que el procesador (que está contenido dentro del microcontrolador) sepa con qué elementos de hardware cuenta y cómo usarlos, osea que reconozca todos los elementos a su alrededor, esto incluye la búsqueda en el disco duro del sistema operativo (también se le puede indicar al procesador que no existe un sistema operativo), la búsqueda de las memorias con las que cuenta, si vamos a usar o no el watchdog, el reloj u oscilador a usar, etc. a todo esto se le llama configuración y se realiza a través de **PALABRAS DE CONFIGURACIÓN** en conjunto con la directiva **_CONFIG**, que interactúa con la **memoria de configuración** que pertenece a la **memoria ROM**.



- Las palabras de configuración quemadas en las direcciones de la **memoria de configuración** son las siguientes, y a continuación, se describirá el porqué de cada una:

- Directiva en el registro **2007h** de la memoria de configuración:

- + **_CONFIG 0X2007, 0X2BE4**

- Directiva en el registro **2008h** de la memoria de configuración:

- + **_CONFIG 0X2008, 0X3FFF**

3. SIGNIFICADO DE LOS 14 BITS EN LA PALABRA DE CONFIGURACIÓN 1 (dirección **0X2007**):

- **DEBUG (bit 13 de la palabra de configuración)**: Lo que hace este bit es **activar o desactivar la depuración (debug) del programa**, esto lo que hace es checar que el programa esté bien escrito y funcionando bien, línea por línea en el código.

El problema es que al prender el debug se deshabilitan los pines de entrada y salida del puerto B, por lo que no podré usarlos como entradas/salidas.

a. Bit 13

- i. **1**: Apaga el DEBUG.
- ii. **0**: Enciende el DEBUG, pero deshabilita los pines del puerto B.

- **LVP (bit 12 de la palabra de configuración)**: El microprocesador requiere de 12 Volts para quemar instrucciones en su memoria FLASH, pero para cambiar eso **existe el modo Programación de Baja Tensión, que permite al microcontrolador quemar instrucciones en la memoria FLASH sin necesidad de estar conectado a una fuente de 12V, cuando el modo de baja tensión está activado se puede programar utilizando un nivel de tensión mínimo de 2.5V, aunque el problema es que le toma más tiempo quemar instrucciones de esta manera**. El problema es que al prender el modo de baja tensión se deshabilitan los pines de entrada y salida del puerto B, por lo que no podré usarlos como entradas/salidas.

a. Bit 12

- i. **1**: Enciende el Modo de Baja Tensión, pero deshabilita los pines del puerto B.
 - ❖ El pin RB3/PGM adopta la configuración PGM (Program), que sirve para habilitar el modo de programación de baja tensión, deshabilitando todos los pines del puerto B.
- ii. **0**: Apaga el Modo de Baja Tensión, habilita el Puerto B como entradas y salidas digitales o analógicas.

- **FCMEN (bit 11 de la palabra de configuración)**: **Activa un modo llamado Reloj a Prueba de Fallas que monitorea si el oscilador funciona correctamente o no**, esto vale la pena aplicarlo cuando no puede haber fallas de tiempo en el microprocesador o cuando trabaja largas jornadas de trabajo (24/7).

Este modo no afecta activarlo, ya que no deshabilita los pines del puerto B.

a. Bit 11

- i. **1**: Activa el modo Reloj a Prueba de Fallas.
- ii. **0**: Apaga el modo Reloj a Prueba de Fallas.

- **IESO (bit 10 de la palabra de configuración)** **Divisor de Reloj**: **Activa el modo divisor de reloj**, ya que con el oscilador interno o externo que dé la señal de reloj (con la cual se marcará el paso para el funcionamiento del microcontrolador), se puede aplicar un divisor de reloj para reducir la frecuencia del reloj.

Para poder hacer un divisor de reloj se usa el registro de propósito específico llamado **OSCCON**, donde 3 de sus bits (del 4 al 6) llamados IRCF se usan para seleccionar una frecuencia de las enlistadas en la siguiente figura, dependiendo de los bits que mandemos al registro, como viene indicado en las páginas 61 y 62 del datasheet.

Además, con el bit **SCS** del registro **OSCCON** (que es el primero) podemos elegir si queremos usar un oscilador externo o el interno.

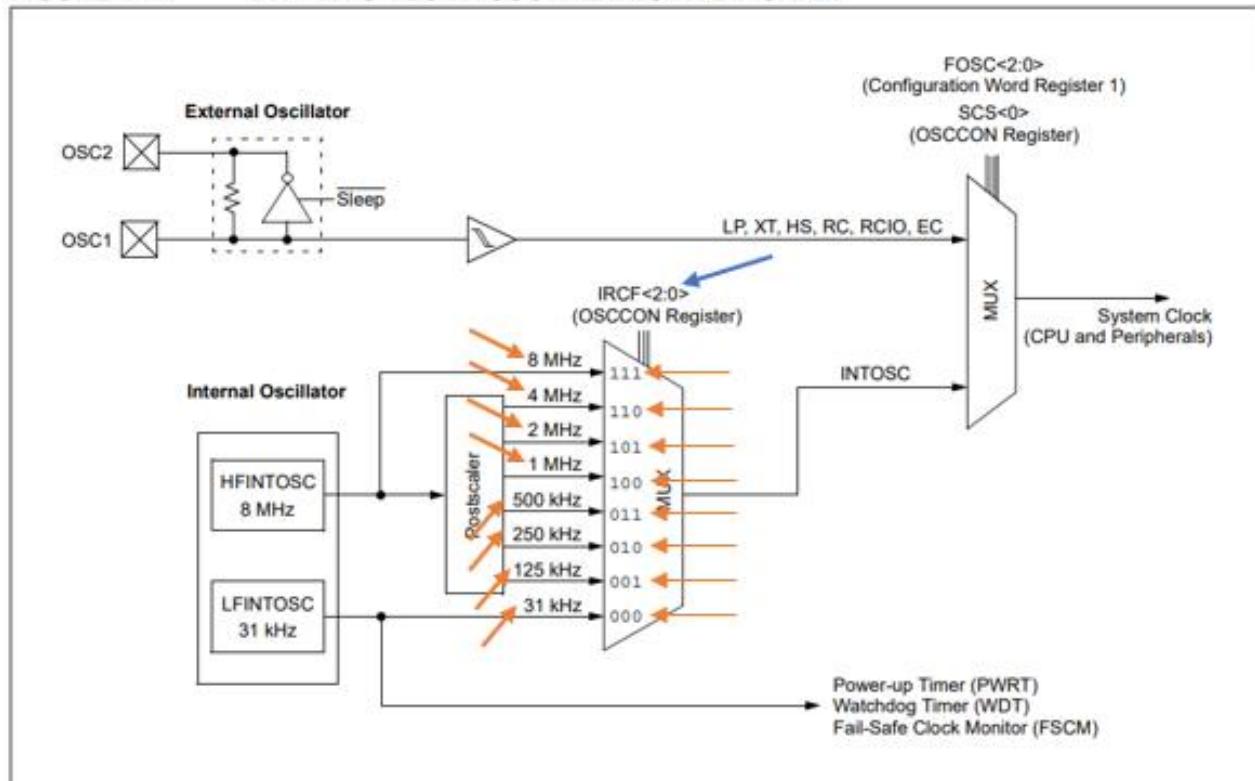
- a. **1**: Usamos el oscilador interno del PIC.
- b. **0**: Usamos un oscilador externo, el tipo del oscilador externo es dado por los bits FOSC de la palabra de configuración actual.

- Podemos observar además que por default la frecuencia de este oscilador interno es de 4MHz a menos que se active y cambie directamente con el divisor de reloj.

a. Bit 10

- 1:** Activa el modo Divisor de Reloj.
- 0:** Desactiva el modo Divisor de Reloj.

FIGURE 4-1: PIC® MCU CLOCK SOURCE BLOCK DIAGRAM



REGISTER 4-1: OSCCON: OSCILLATOR CONTROL REGISTER

U-0	R/W-1	R/W-1	R/W-0	R-1	R-0	R-0	R/W-0
—	IRCF2	IRCF1	IRCF0	OSTS ⁽¹⁾	HTS	LTS	SCS
bit 7							bit 0

Legend:

R = Readable bit

W = Writable bit

U = Unimplemented bit, read as '0'

-n = Value at POR

'1' = Bit is set

'0' = Bit is cleared

x = Bit is unknown

bit 7 **Unimplemented:** Read as '0'

bit 6-4 **IRCF<2:0>:** Internal Oscillator Frequency Select bits

- 111 = 8 MHz
- 110 = 4 MHz (default)
- 101 = 2 MHz
- 100 = 1 MHz
- 011 = 500 kHz
- 010 = 250 kHz
- 001 = 125 kHz
- 000 = 31 kHz (LFINTOSC)

REGISTER 4-1: OSCCON: OSCILLATOR CONTROL REGISTER

U-0	R/W-1	R/W-1	R/W-0	R-1	R-0	R-0	R/W-0
—	IRCF2	IRCF1	IRCF0	OSTS ⁽¹⁾	HTS	LTS	SCS
bit 7	bit 0						

Legend:

R = Readable bit
-n = Value at POR

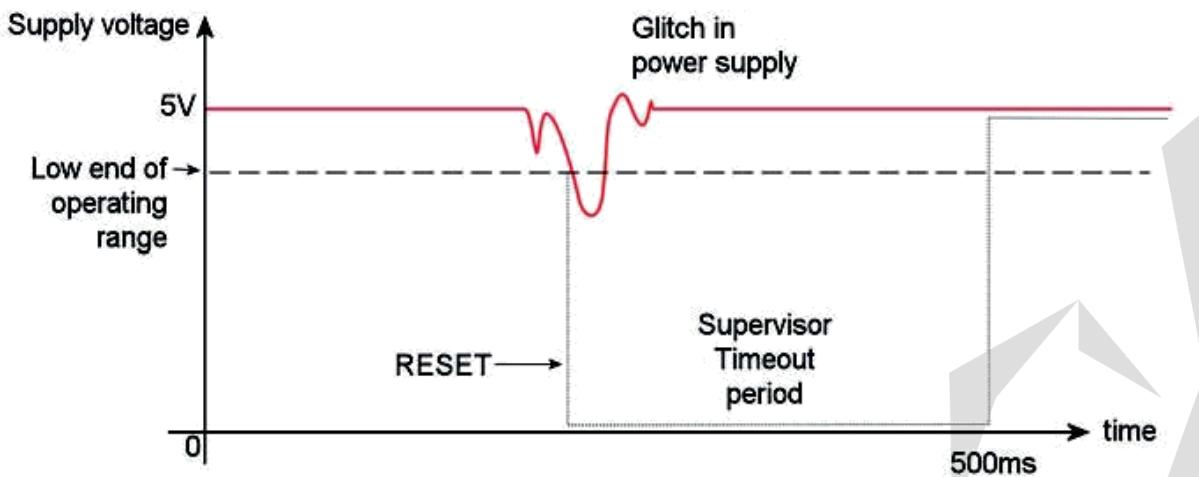
W = Writable bit
'1' = Bit is set

U = Unimplemented bit, read as '0'
'0' = Bit is cleared
x = Bit is unknown

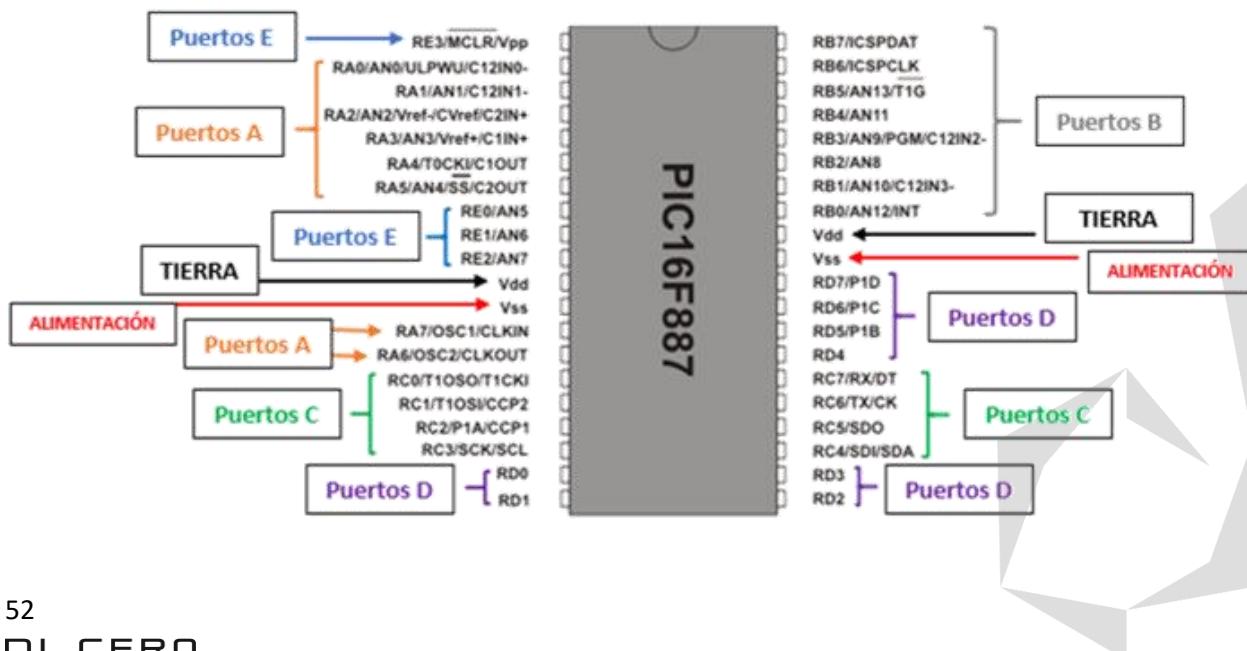
- bit 7 **Unimplemented:** Read as '0'
- bit 6-4 **IRCF<2:0>:** Internal Oscillator Frequency Select bits
 111 = 8 MHz
 110 = 4 MHz (default)
 101 = 2 MHz
 100 = 1 MHz
 011 = 500 kHz
 010 = 250 kHz
 001 = 125 kHz
 000 = 31 kHz (LFINTOSC)
- bit 3 **OSTS:** Oscillator Start-up Time-out Status bit⁽¹⁾
 1 = Device is running from the external clock defined by FOSC<2:0> of the CONFIG1 register
 0 = Device is running from the internal oscillator (HFINTOSC or LFINTOSC)
- bit 2 **HTS:** HFINTOSC Status bit (High Frequency – 8 MHz to 125 kHz)
 1 = HFINTOSC is stable
 0 = HFINTOSC is not stable
- bit 1 **LTS:** LFINTOSC Stable bit (Low Frequency – 31 kHz)
 1 = LFINTOSC is stable
 0 = LFINTOSC is not stable
- bit 0 **SCS:** System Clock Select bit
 1 = Internal oscillator is used for system clock
 0 = Clock source defined by FOSC<2:0> of the CONFIG1 register



- **BOREN o Brown Out Reset Selection (bits 9 y 8 de la palabra de configuración):** Activar el Brown Out ofrece una protección extra a la función del microcontrolador ya que se ocasiona un reset cuando el oscilador falle y baje de cierto nivel del voltaje, esto importa porque cuando ocurre esta situación se ve afectado el contador del programa y el PIC no ejecuta correctamente sus instrucciones respecto al manejo de tiempos.
 La configuración del Brown-Out se hace a través de las 2 palabras de configuración del PIC. Hay un registro de propósito específico llamado PCON (Power Control Register) que tiene un bit llamado BOR que me indica cuando en el PIC ocurrió un reset causado por el Brown Out y hay otro bit llamado POR que indica cuando el PIC se prendió por primera vez.



- a. Bits 9 y 8
 - i. **11:** Activa el Brown-Out todo el tiempo.
 - ii. **10:** Activa el Brown-Out solamente cuando el microcontrolador esté en operación y lo desactiva cuando se utilice la instrucción SLEEP.
 - iii. **10:** Hace que el Brown-Out pueda ser controlado por el bit 4 del registro PCON.
 - iv. **00: Apaga el Brown-Out.**
- **CPD (bit 7 de la palabra de configuración): Protege que la memoria de datos (RAM) no pueda ser escrita**, logrando así que no se pueda editar ni replicar mi código.
El CPD debe estar prendido siempre como buena práctica en proyectos formales.
 - a. Bit 7
 - i. **1:** Apaga el modo de protección de escritura en la memoria RAM.
 - ii. **0: Enciende el modo de protección de escritura en la memoria RAM.**
- **CP (bit 6 de la palabra de configuración): Protege que la memoria de instrucciones (FLASH) no pueda ser escrita**, logrando así que no se pueda editar ni replicar mi código.
El CP debe estar prendido siempre como buena práctica en proyectos formales.
 - a. Bit 6
 - i. **1:** Apaga el modo de protección de escritura en la memoria FLASH.
 - ii. **0: Enciende el modo de protección de escritura en la memoria FLASH.**
- **MCLRE (bit 5 de la palabra de configuración):** Configura el pin RE3 (que es el primero del puerto E y se encuentra en la esquina superior izquierda del PIC16F887) para que pueda ejecutar un reinicio (reset) en el microcontrolador, activado de forma externa cuando reciba una entrada digital o que simplemente sea un pin de entradas y salidas.
 - a. Bit 5
 - i. **1:** Hace que el pin RE3 del puerto E funcione como reset cuando le entre una señal digital.
 - ❖ El pin RE3/MCLR/Vpp adopta la configuración MCLR (Master CLeaR), que sirve para habilitar el modo de reset externo.
 - ii. **0: Hace que el pin RE3 del puerto E funcione como entrada/salida digital o analógica.**

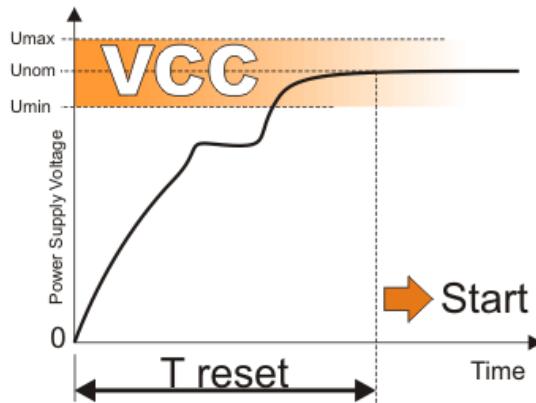


- **PWRTE (bit 4 de la palabra de configuración):** Hace que el microcontrolador se tarde 75 milisegundos al encenderse, esto se hace para proteger al PIC de las oscilaciones provenientes de la señal de alimentación de 5V.

Cuando se enciende el PIC, puede ser que en la alimentación (pila o fuente de 5V) se presente una fase semiamortiguada, amortiguada o críticamente amortiguada al inicio. Estas variaciones de voltaje pueden dañar al procesador por lo que se declara un tiempo de estabilización de 75 milisegundos donde el microcontrolador no recibe nada y se espera a que estas oscilaciones de la fuente pasen, a este modo se le llama Power-up Timer. Este tiempo de estabilización lo puedo apagar y hacer que mi microprocesador encienda más rápidamente, pero me arriesgo a que las oscilaciones de la alimentación lo dañen.

a. Bit 4

- 1:** Apaga el Power-up Timer, fijando un tiempo de estabilización de 0 milisegundos.
- 0:** Enciende el Power-up Timer, fijando un tiempo de estabilización de 75 milisegundos para proteger al microcontrolador.



- **WDTEN (bit 3 de la palabra de configuración):** Activa o apaga el perro guardián o watchdog. El watchdog es de mucha utilidad cuando el programa va a funcionar por largas jornadas de trabajo (24/7) o va a estar expuesto a entornos dañinos en temperatura o de gran estática eléctrica que puedan dañar a los elementos internos del microcontrolador, en el curso lo vamos a inhabilitar porque no es necesario.

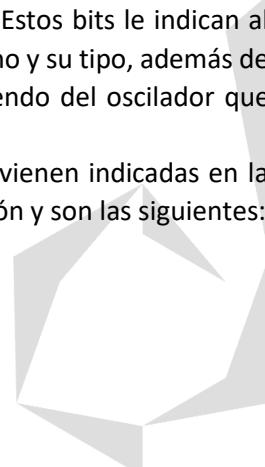
a. Bit 3

- 1:** Enciende el Watchdog.
- 0:** Apaga el Watchdog.

- **FOSC (bits 2, 1 y 0 de la palabra de configuración) Tipo de Oscilador:** Estos bits le indican al microprocesador qué oscilador estamos usando, ya sea interno, externo y su tipo, además de configurar la función de los pines RA6 y RA7 del puerto A, dependiendo del oscilador que hayamos elegido.

Podemos elegir 4 de las 8 configuraciones de oscilador en total que vienen indicadas en la página 61 del datasheet con los bits FOSC de la palabra de configuración y son las siguientes:

- LP (Low Power Crystal).**
- XT (Crystal/Resonator).**
- HS (High-Speed Crystal/Resonator).**
- RC (Resistor/Capacitor).**



The Oscillator module can be configured in one of eight clock modes.

1. EC – External clock with I/O on OSC2/CLKOUT.
2. LP – 32 kHz Low-Power Crystal mode.
3. XT – Medium Gain Crystal or Ceramic Resonator Oscillator mode.
4. HS – High Gain Crystal or Ceramic Resonator mode.
5. RC – External Resistor-Capacitor (RC) with Fosc/4 output on OSC2/CLKOUT.
6. RCIO – External Resistor-Capacitor (RC) with I/O on OSC2/CLKOUT.
7. INTOSC – Internal oscillator with Fosc/4 output on OSC2 and I/O on OSC1/CLKIN.
8. INTOSCIOS – Internal oscillator with I/O on OSC1/CLKIN and OSC2/CLKOUT.

Aquí podemos ver la función de la **memoria de configuración** a la que estamos accediendo con la palabra de configuración, que le dicen al procesador dentro del microcontrolador los elementos de hardware con los que cuenta.

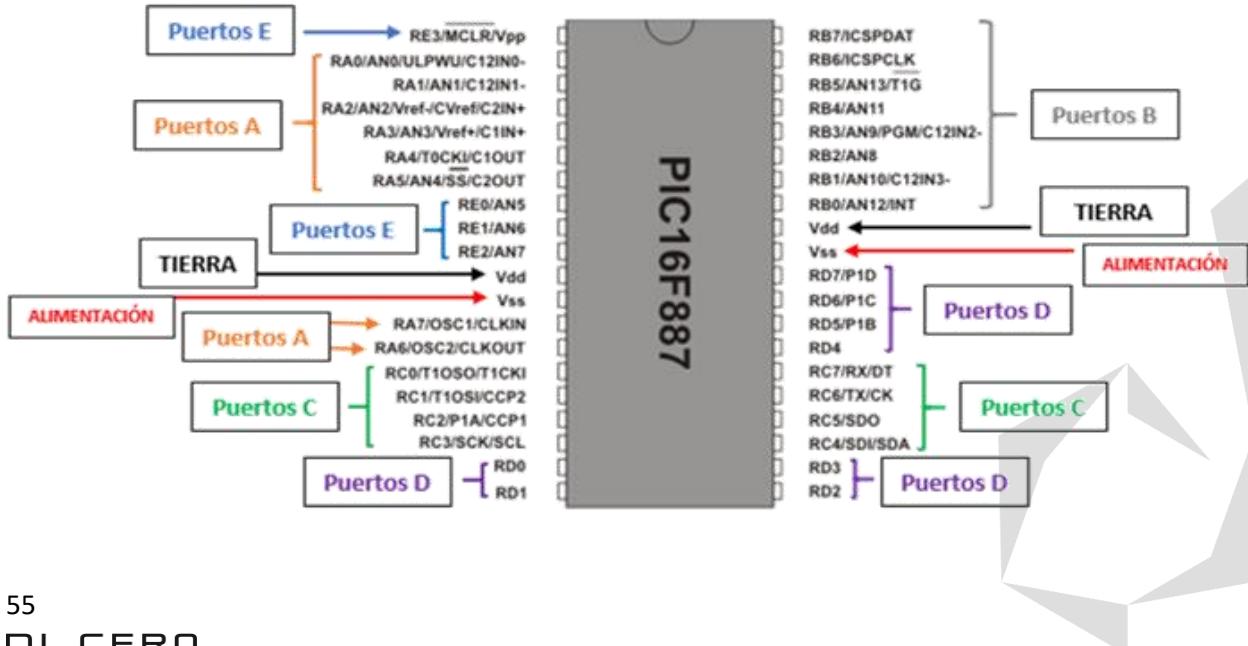
e. Bits 2, 1 y 0

- i. **111:** El oscilador es de tipo RC externo.
 - ❖ Pin RA6 del puerto A: Funciona como salida del oscilador, se configura como CLKOUT.
 - ❖ Pin RA7 del puerto A: Funciona como entrada para un oscilador tipo RC, se configura como RC.
- ii. **110:** El oscilador es de tipo RCIO externo, que es también un oscilador RC, pero configura uno de sus pines para que sea entrada/salida digital o analógica y que el otro sea la entrada del oscilador externo RC.
 - ❖ Pin RA6 del puerto A: Funciona como entrada/salida digital o analógica, se configura como I/O.
 - ❖ Pin RA7 del puerto A: Funciona como entrada para un oscilador tipo RC, se configura como RC.
- iii. **101:** El oscilador es de tipo INTOSC, que es el interno que viene incluido en el PIC, donde se configura uno de los pines del puerto A para que sea la salida del oscilador interno y al otro que sea entrada/salida digital o analógica.
 - ❖ Pin RA6 del puerto A: Funciona como una salida del oscilador, se configura como CLKOUT.
 - ❖ Pin RA7 del puerto A: Funciona como entrada/salida digital o analógica, se configura como I/O.

- iv. **100:** El oscilador es de tipo INTOSCO, que es el interno que viene incluido en el PIC, se le llama de forma diferente al anterior porque configura los pines RA6 y RA7 para que ambos sean entradas/salidas analógicas o digitales.
 - ❖ **Pin RA6 del puerto A:** Funciona como entrada/salida digital o analógica, se configura como I/O.
 - ❖ **Pin RA7 del puerto A:** Funciona como entrada/salida digital o analógica, se configura como I/O.
- v. **011:** El oscilador es de tipo EC, que significa cualquier oscilador externo que no sea RC y configura uno de los pines RA6 o RA7 del puerto A como entrada/salida digital y al otro como entrada del oscilador.
 - ❖ **Pin RA6 del puerto A:** Funciona como entrada/salida digital o analógica, se configura como I/O.
 - ❖ **Pin RA7 del puerto A:** Funciona como entrada del oscilador, se configura como CLKIN.
- vi. **010:** El oscilador es de tipo HS, que significa cualquier oscilador externo de alta velocidad que no sea RC y configura ambos pines del puerto A como entradas del oscilador.
 - ❖ **Pin RA6 del puerto A:** Funciona como entrada del oscilador no referenciada a tierra, se configura como OSC1.
 - ❖ **Pin RA7 del puerto A:** Funciona como entrada del oscilador no referenciada a tierra, se configura como OSC2.

Como primera instancia vamos a usar la configuración INTOSCO que usa al oscilador interno y configura los pines RA6 y RA7 del puerto A para que funcionen como entradas/salidas digitales o analógicas, por lo que pondremos los bits 100 en los bits FOSC que abarcan los bits 2, 1 y 0 de la palabra de configuración.

Cuando quiera utilizar un oscilador externo la opción más viable es usar la configuración EC que configura uno de los pines RA6 o RA7 del puerto A como entrada/salida digital y al otro como entrada del oscilador, por lo que por lo que pondremos los bits 011 en los bits FOSC que abarcan los bits 2, 1 y 0 de la palabra de configuración.



Por lo tanto, elegimos los siguientes bits en la **palabra de configuración 1** para nuestros primeros programas:

- **DEBUG (bit 13 de la palabra de configuración)**: Lo que hace este bit es **activar o desactivar la depuración (debug) del programa**, esto lo que hace es checar que el programa esté bien escrito y funcionando bien, línea por línea en el código.
 - a. **Bit 13**
 - i. **1**: Apaga el DEBUG.
- **LVP (bit 12 de la palabra de configuración)**: El microprocesador requiere de 12 Volts para quemar instrucciones en su memoria FLASH, pero para cambiar eso **existe el modo Programación de Baja Tensión, que permite al microcontrolador quemar instrucciones en la memoria FLASH sin necesidad de estar conectado a una fuente de 12V, cuando el modo de baja tensión está activado se puede programar utilizando un nivel de tensión mínimo de 2.5V, aunque el problema es que le toma más tiempo quemar instrucciones de esta manera**. El problema es que al prender el modo de baja tensión se deshabilitan los pines de entrada y salida del puerto B, por lo que no podré usarlos como entradas/salidas.
 - a. **Bit 12**
 - i. **0**: Apaga el Modo de Baja Tensión, habilita el Puerto B como entradas y salidas digitales o analógicas.
- **FCMEN (bit 11 de la palabra de configuración)**: **Activa un modo llamado Reloj a Prueba de Fallas que monitorea si el oscilador funciona correctamente o no**, esto vale la pena aplicarlo cuando no puede haber fallas de tiempo en el dispositivo donde está integrado el microprocesador o que trabaja largas jornadas de trabajo (24/7).
 - a. **Bit 11**
 - i. **1**: Activa el modo Reloj a Prueba de Fallas.
- **IESO (bit 10 de la palabra de configuración)**: **Activa el modo divisor de reloj**, ya que con el oscilador interno o externo que dé la señal de reloj con la cual se marcará el paso para el funcionamiento del microcontrolador, se puede aplicar un divisor de reloj para reducir su frecuencia y así reducir la frecuencia del reloj u obtener varias frecuencias para alguna aplicación.
 - a. **Bit 10**
 - i. **0**: Desactiva el modo Divisor de Reloj.
- **BOREN o Brown Out Reset Selection (bits 9 y 8 de la palabra de configuración)**: Activar el Brown Out ofrece una protección extra a la función del microcontrolador ya que **se ocasiona un reset cuando el oscilador falle y baje de cierto nivel del voltaje**, esto importa porque **cuando ocurre esta situación se ve afectado el contador del programa y el PIC no ejecuta correctamente sus instrucciones respecto al tiempo**.
 - a. **Bits 9 y 8**
 - i. **11**: Activa el Brown-Out todo el tiempo.
- **CPD (bit 7 de la palabra de configuración)**: **Protege que la memoria de datos (RAM) no pueda ser escrita**, logrando así que no se pueda editar ni replicar mi código.
 - a. **Bit 7**
 - i. **1**: Apaga el modo de protección de escritura en la memoria RAM.

- **CP (bit 6 de la palabra de configuración):** Protege que la memoria de instrucciones (**FLASH**) no pueda ser escrita, logrando así que no se pueda editar ni replicar mi código.
 - a. **Bit 6**
 - i. **1:** Apaga el modo de protección de escritura en la memoria FLASH.
- **MCLRE (bit 5 de la palabra de configuración):** Hace que el pin RE3 (que es el primero del puerto E y se encuentra en la esquina superior izquierda del PIC16F887) pueda crear un reinicio (reset) en el microcontrolador cuando reciba una entrada digital o que simplemente sea un pin de entradas y salidas.
 - a. **Bit 5**
 - i. **1:** Hace que el pin RE3 del puerto E funcione como reset cuando le entre una señal digital.
- **PWRTE (bit 4 de la palabra de configuración):** Hace que el microcontrolador se tarde 75 milisegundos para encenderse, esto se hace para proteger el PIC de las oscilaciones provenientes de la señal de alimentación de 5V.
 - a. **Bit 4**
 - i. **0:** Enciende el Power-up Timer, fijando un tiempo de estabilización de 75 milisegundos de retardo al encender para proteger al microcontrolador.
- **WDTEN (bit 3 de la palabra de configuración):** Activa o apaga el perro guardián o watchdog.
 - a. **Bit 3**
 - i. **0:** Apaga el Watchdog.
- **FOSC (bits 2, 1 y 0 de la palabra de configuración):** Estos bits le indican al microprocesador qué oscilador estamos usando, ya sea interno, externo y su tipo, además de configurar la función de los pines RA6 y RA7 del puerto A dependiendo del oscilador que hayamos elegido.
 - a. **Bits 2, 1 y 0**
 - i. **100:** El oscilador es de tipo **INTOSCO**, que es el interno que viene incluido en el PIC, se le llama de forma diferente al anterior porque configura los pines RA6 y RA7 para que ambos sean entradas/salidas analógicas o digitales.
 - ❖ **Pin RA6 del puerto A:** Funciona como entrada/salida digital o analógica, se configura como I/O.
 - ❖ **Pin RA7 del puerto A:** Funciona como entrada/salida digital o analógica, se configura como I/O.

Por lo tanto, los 14 bits de la **palabra de configuración 1** son:

10 1011 1110 0100

Que se debe poner en el registro **0X2007** en hexadecimal y es igual a:

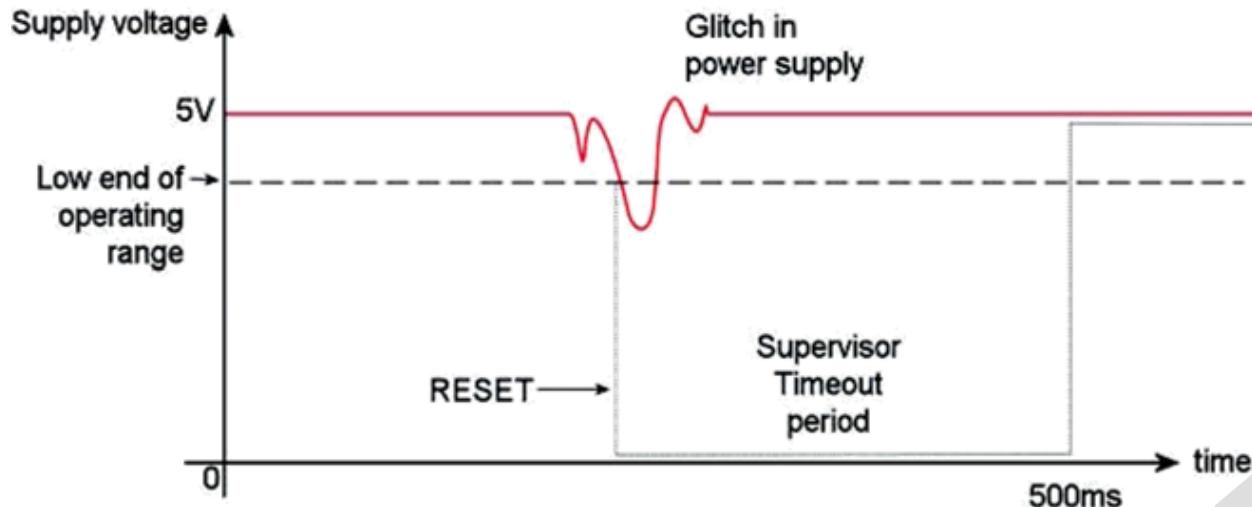
0X2BE4

4. **SIGNIFICADO DE LOS 14 BITS EN LA PALABRA DE CONFIGURACIÓN 2 (dirección **0X2008**):**
 - **UNIMPLEMENTED (bits 13, 12 y 11 de la palabra de configuración):** Siempre estarán como 1, no se les debe cambiar.
 - a. **Bits 13, 12 y 11**

- i. **111**: Siempre estarán de esta manera, no se les debe cambiar.
- **WRT (bits 10 y 9 de la palabra de configuración)**: Activan un modo de protección de la memoria FLASH, **protege que la memoria de instrucciones (FLASH) no pueda ser escrita**, logrando así que no se pueda editar ni replicar mi código.
 - a. **Bits 10 y 9**
 - i. **00**: Protege de escritura **todos** los registros de la memoria FLASH (del 0000h al OFFFh).
 - ii. **01**: Protege de escritura **casi todos** los registros de la memoria FLASH (del 0000h al 07FFh).
 - iii. **10**: Protege de escritura **la mitad** de los registros en la memoria FLASH (del 0000h al OOFFh).
 - iv. **11**: Apaga el modo de protección de escritura en la memoria FLASH.
- **BOREN o Brown Out Reset Selection (bit 8 de la palabra de configuración)**: Activar el Brown Out ofrece una protección extra a la función del microcontrolador ya que **se ocasiona un reset cuando el oscilador falle y baje de cierto nivel del voltaje**, esto importa porque **cuando ocurre esta situación se ve afectado el contador del programa y el PIC no ejecuta correctamente sus instrucciones respecto al tiempo**.

En la primera palabra de configuración se indica cuándo y si el Brown-Out va a estar encendido y en la segunda palabra de configuración se indica el nivel de voltaje donde se activará.

Hay un registro de propósito específico llamado PCON (Power Control Register) que tiene un bit llamado BOR que me indica cuando en el PIC ocurrió un reset causado por el Brown Out y hay otro bit llamado POR que indica cuando el PIC se prendió por primera vez.



- a. **Bit 8**
 - i. **1**: Hace que el Brown-Out reinicie el PIC cuando la señal de reloj baje de 4 V.
 - ii. **0**: **Hace que el Brown-Out reinicie el PIC cuando la señal de reloj baje de 2.1 V.**
- **UNIMPLEMENTED (bits 7, 6, 5, 4, 3, 2, 1 y 0 de la palabra de configuración)**: Siempre estarán como 1, no se les debe cambiar.
 - a. **Bits 7, 6, 5, 4, 3, 2, 1 y 0**
 - i. **1111 1111**: Siempre estarán de esta manera, no se les debe cambiar.

Por lo tanto, elegimos los siguientes bits en la **palabra de configuración 2** para nuestros primeros programas:

- **UNIMPLEMENTED (bits 13, 12 y 11 de la palabra de configuración)**: Siempre estarán como 1, no se les debe cambiar.
 - a. **Bits 13, 12 y 11**
 - i. **111**: Siempre estarán de esta manera, no se les debe cambiar.
- **WRT (bits 10 y 9 de la palabra de configuración)**: Activan un modo de protección de la memoria FLASH, **protege que la memoria de instrucciones (FLASH) no pueda ser escrita**, logrando así que no se pueda editar ni replicar mi código.
 - a. **Bits 10 y 9**
 - i. **11**: Apaga el modo de protección de escritura en la memoria FLASH.
- **BOREN o Brown Out Reset Selection (bit 8 de la palabra de configuración)**: Activar el Brown Out ofrece una protección extra a la función del microcontrolador ya que **se ocasiona un reset cuando el oscilador falle y baje de cierto nivel del voltaje**, esto importa porque **cuando ocurre esta situación se ve afectado el contador del programa y el PIC no ejecuta correctamente sus instrucciones respecto al tiempo**.
 - a. **Bit 8**
 - i. **1**: Hace que el Brown-Out reinicie el PIC cuando la señal de reloj baje de 4 V.
- **UNIMPLEMENTED (bits 7, 6, 5, 4, 3, 2, 1 y 0 de la palabra de configuración)**: Siempre estarán como 1, no se les debe cambiar.
 - a. **Bits 7, 6, 5, 4, 3, 2, 1 y 0**
 - i. **1111 1111**: Siempre estarán de esta manera, no se les debe cambiar.

Por lo tanto, los 14 bits de la **palabra de configuración 2** son:

11 1111 1111 1111

Que se debe poner en el registro **0X2008** en hexadecimal y es igual a:

0X3FFF

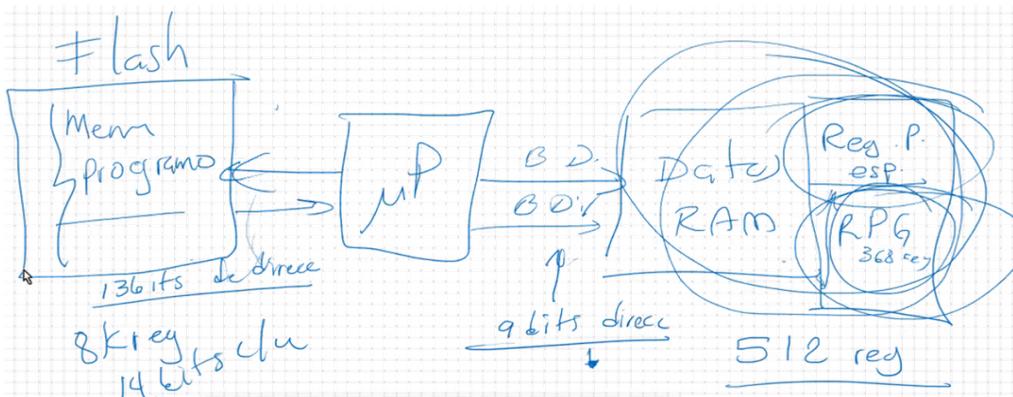
5. INCLUDE <DIRECCIÓN Y NOMBRE DEL ARCHIVO CON SU EXTENSIÓN>:

- a. La directiva INCLUDE sirve para abrir un archivo de texto plano, copiar todo su contenido y pegarlo en el programa, esto no podrá ser visible para nosotros, pero todo su código estará escrito en el programa, justo donde se encuentra la directiva INCLUDE y nos sirve para poder jalar funcionalidades o programas previamente escritos, es una forma de importar librerías o archivos en nuestro código ensamblador.
 - Se debe especificar la ruta en donde se guardó el archivo que queremos accesar, su nombre y extensión.
- b. La directiva INCLUDE siempre se debe añadir cuando programemos un PIC ya que ésta incluye sus 35 instrucciones y las directivas, adjuntando también los nombres EQU para los registros de propósito específico. La directiva para el PIC16F887 es la siguiente:
 - **INCLUDE <P16F887.INC>**

6. ORG DIRECCIÓN HEXADECIMAL DE 4 DÍGITOS:
- La directiva ORG (u origen) le indica al programa desde qué dirección de memoria FLASH debe empezar a guardar todas las instrucciones del código.
 - Alado de la directiva ORG se declaran 4 dígitos hexadecimales (donde cada uno es de 4 bits, dando 16 bits en total) porque para indicar una dirección de memoria FLASH se necesitan 13 bits de direccionamiento y con 3 dígitos hexadecimales nos quedamos cortos porque son de 12 bits.
 - Normalmente se le indica al programa que empiece a guardar todas las instrucciones partiendo desde la dirección 0 en la memoria FLASH de la siguiente manera:

 ORG 0X0000

 - En el PIC existen en total 4,369 direcciones de memoria FLASH para guardar líneas de código, esto corresponde a la dirección hexadecimal 0X1111.



7. NOMBRE_A DECLARAR EQU REGISTRO_ASIGNADO:
- Con la directiva EQU puedo asociar a algún registro de propósito general un nombre en específico, esto se hace cuando voy a utilizar mucho un registro, los nombres se suelen poner en mayúsculas por buenas prácticas.
 - No se puede aplicar esta directiva a registros de propósito específico porque esos ya tienen un nombre asignado.
 - Ejemplo: Se asignará al registro 0X04 (4 hexadecimal) del banco 1 la directiva EQU ZORRO para que de aquí en adelante se pueda llamar y utilizar de esa manera.

 ZORRO EQU 0X04
 - Con la directiva EQU también se puede simplemente asociar una palabra con un número, creando así un tipo de variable o constante.
 - Ejemplo se quiere usar varias veces el registro 0X00, por lo cual se le asigna la directiva EQU de C, al usarse en el código, ahora al registro 0X00 se referirá como C, de la siguiente manera:

	C	EQU	0X00
	BSF	STATUS	,C

8. CBLOCK REGISTRO_INICIAL PALABRA1, ..., PALABRA_N:

- La directiva CBLOCK sirve para hacer lo mismo que EQU, pero con varias palabras asignadas a más de un registro de la RAM que parten desde el registro yo le indique.

b. Otra forma de explicarlo es que iguala una serie de palabras con un grupo de números consecutivos, iniciando a partir de un número indicado como parámetro después de la directiva CBLOCK.

- Ejemplo: Se asignarán varios números consecutivos a varias variables o constantes, para ello se utilizará la directiva **CBLOCK** de la siguiente manera, donde se asignará a la PALABRA1 el número 0X00, a la PALABRA2 el número 0X01, a la PALABRA3 el número 0X02, etc. terminando todo con la instrucción **ENDC**:

```
⊕ CBLOCK      0X00  
⊕ PALABRA1, PALABRA2, PALABRA3, ..., PALABRA_N  
⊕ ENDC
```

9. END:

- c. La directiva **END** se usa para terminar el programa, indicándole hasta donde debe ensamblar el código.
- En el programa siempre se debe poner la instrucción **END** y solo se puede usar una vez.
- d. Ensamblar significa quemar cada instrucción del código en un registro de la memoria FLASH.
- Lo que siga después de la directiva **END** ya no se ensamblará, osea que no se quemará en ninguna parte de la memoria FLASH.

Subrutinas

Cuando un pedazo del código escrito en lenguaje ensamblador se va a estar reutilizando en distintas partes del mismo código, se declara como “subrutina”, que equivale a ser una “función” en los demás lenguajes de programación. Todas las subrutinas se escriben hasta el final o en el inicio del programa, se mandan a llamar con la instrucción **CALL** y se terminan con la instrucción **RETURN**.

Cuando una subrutina utiliza la instrucción **CALL** y **RETURN**, está accediendo a la **Pila (o STACK en inglés)**, que es un pedazo de la memoria RAM usado específicamente para guardar valores del **Contador del programa (o Program Counter, PC)** en 8 niveles dentro del PIC16F887 (cada nivel es un registro de 13 bits que almacena un valor del **PC**), esto se hace porque el **Apuntador de la Pila (o STACK POINTER en inglés)** guarda en estos 8 niveles los valores del **PC** que vayan después de la instrucción **CALL**, para que cuando se acabe la subrutina y se ejecute la instrucción **RETURN**, se vuelva a cargar en el **PC** el valor que se quedó almacenado en la **Pila**, logrando así que el programa siga con su ejecución normal desde donde se quedó después de haber ejecutado la subrutina.

Como se cuentan con 8 niveles de **Pila**, solamente se pueden anidar 8 subrutinas entre sí, ya que se sube un nivel en la **Pila** cuando se anidan subrutinas y si se anidan más de 8, la instrucción **CALL** no tendrá donde guardar el siguiente **PC** y la **Pila** se desbordará, haciendo que el programa falle fatalmente porque el **STACK POINTER** está intentando alcanzar un nivel de la **Pila** que no existe (aunque se pueden hacer las subrutinas que queramos dentro de un mismo programa mientras no sean más de 8 anidadas dentro de una misma).

25. CALL k:

- a. Sirve para ejecutar una subrutina, una subrutina es el equivalente a una función en cualquier otro lenguaje de programación.
- b. En el programa se llama a la subrutina siguiendo los pasos:
 - El valor del **contador del programa (PC o Program Counter)** actual se guarda en una dirección de la **Pila**, indicado por el **Apuntador de la Pila (o STACK POINTER en inglés)**.
 - El apuntador de la pila se incrementa en 1.
 - El valor del **PC** se carga con la dirección del valor literal k en donde empieza la subrutina, saltando así a la parte del programa que la describe.
- **CALL**: Lo que hace esta instrucción es guardar en un nivel de la **Pila** el valor de **PC** que le sigue al nivel de código donde se encuentra esta misma instrucción e incrementa en 1 el nivel del **STACK POINTER**, osea sube de nivel en la **Pila**.
 - Ejemplo: Si la instrucción **CALL PERRITOS** se encuentra en el PC 5 (osea que es la quinta línea de código en el programa), guardará en nivel 1 de la Pila el valor 6, hará que el **STACK POINTER** se suba al nivel 2 y cargará en el **PC** el valor que haya donde declaré la subrutina **PERRITOS**.

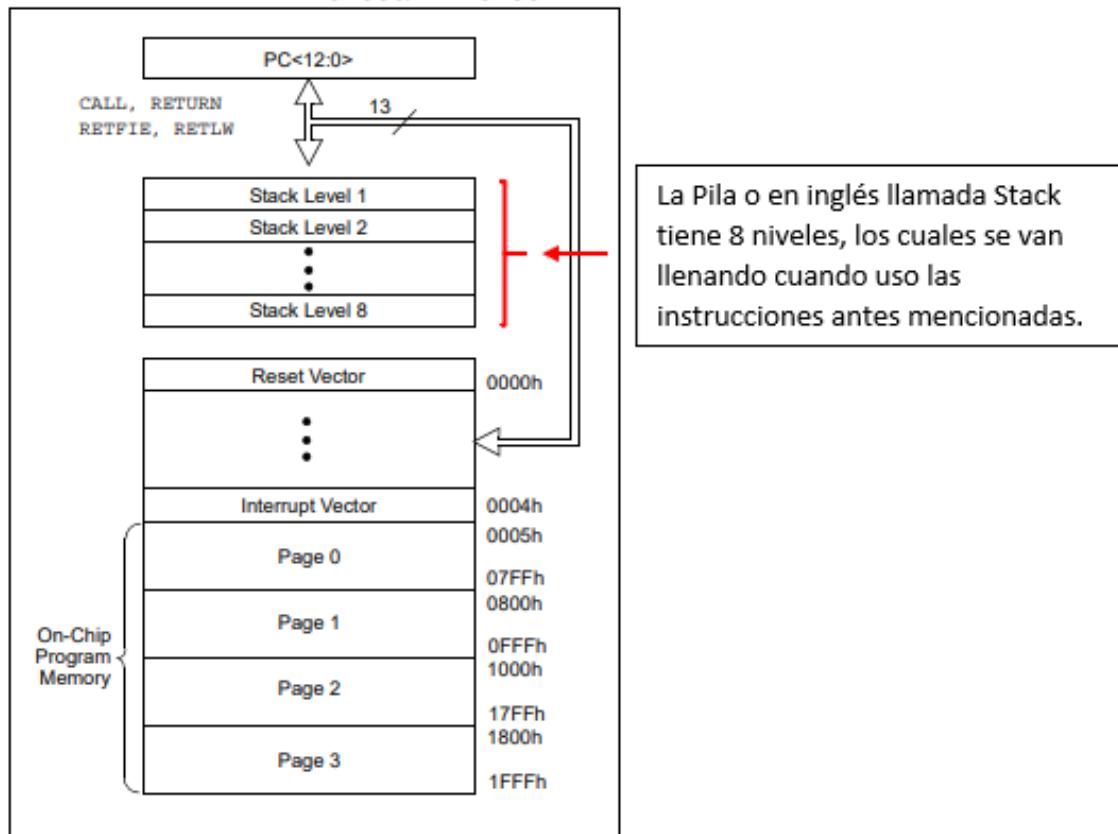
32. RETURN :

- d. Sirve para terminar una subrutina, una subrutina es el equivalente a una función en cualquier otro lenguaje de programación. La instrucción logra regresar de una subrutina a través de los siguientes pasos:
 - Se decrementa en 1 al apuntador de la **Pila**.
 - El **contador del programa (PC o Program Counter)** se carga con el contenido de la dirección de la **Pila** que indica el **Apuntador de la Pila** (que previamente ha sido decrementado).
- **RETURN**: Lo que hace esta instrucción es bajar de nivel al **STACK POINTER**, haciendo que el programa regrese a donde se había quedado antes de llamar la subrutina con la instrucción **CALL**.
 - Ejemplo: Después de usarse la última instrucción **CALL**, el **STACK POINTER** se quedó en el nivel 2 (donde por ahora no hay nada almacenado), al llegar a la instrucción **RETURN** el **STACK POINTER** se bajará al nivel 1 de la **Pila** y cargará en el **PC** el valor 6 que estaba almacenado en ese nivel, haciendo que el programa siga la ejecución normal que llevaba antes de llamar la subrutina **PERRITOS**.

Como las instrucciones **CALL** y **RETURN** modifican el estado del **contador del programa (PC o Program Counter)**, ambas tardan 2 ciclos de máquina en ejecutarse.



FIGURE 2-3: PROGRAM MEMORY MAP AND STACK FOR THE PIC16F886/PIC16F887



1. La forma en la que se llama una subrutina es la siguiente:
 - a. **Primero debemos haber asignado un nombre (directiva EQU) a cierto nivel del código donde empieza la subrutina (así como se hace con las instrucciones GOTO).**
 - b. **Luego ya podremos llamarla y usarla en cualquier punto del programa usando la instrucción CALL junto con el nombre que anteriormente asignamos a la línea de código donde empieza la subrutina.**
 - Al utilizar la instrucción **CALL**:
 - ✚ Se almacena en un nivel de la Pila el valor que le sigue al PC que está en la posición donde se usó la instrucción CALL.
 - ✚ Sube el nivel de la **Pila**.
 - ✚ Carga en el **PC** el valor de donde se encuentra la subrutina que se quiere usar.
 - c. **Siempre al terminar una subrutina debemos declarar la instrucción RETURN, que bajará de nivel la Pila, retornando al programa a donde se había quedado antes de usar la subrutina.**
 - Al utilizar la instrucción **RETURN**:
 - ✚ Se baja un nivel en la **Pila**, haciendo que se cargue en el **PC** el valor que previamente se había almacenado ahí, logrando que el programa siga con la ejecución normal que llevaba antes de haber llamado la subrutina.

La instrucción **RETURN** no limpia lo que se haya almacenado en los niveles de la Pila, simplemente lo que pasa es que cuando el programa se llegue a encontrar con otra instrucción **CALL**, sobrescribe el valor de PC que se haya almacenado en el nivel de la Pila que acceda.

Subrutinas de Tiempo

Las subrutinas de tiempo se usan para hacer que el programa no haga nada por cierto tiempo después de haber ejecutado alguna acción, a esto se le llama manejo de tiempos y es muy importante porque el PIC se caracteriza por ser muy exacto en el tiempo donde ejecuta sus funciones.

En los lenguajes de bajo nivel se tiene un gran control del tiempo, mientras que en los lenguajes de alto nivel como C, Python, JavaScript, etc. donde por ejemplo se usan funciones como `delay()`, `pause()`, etc. para indicar el valor del tiempo que el programa debe dejar pasar, no es tan exacto el manejo de tiempos.

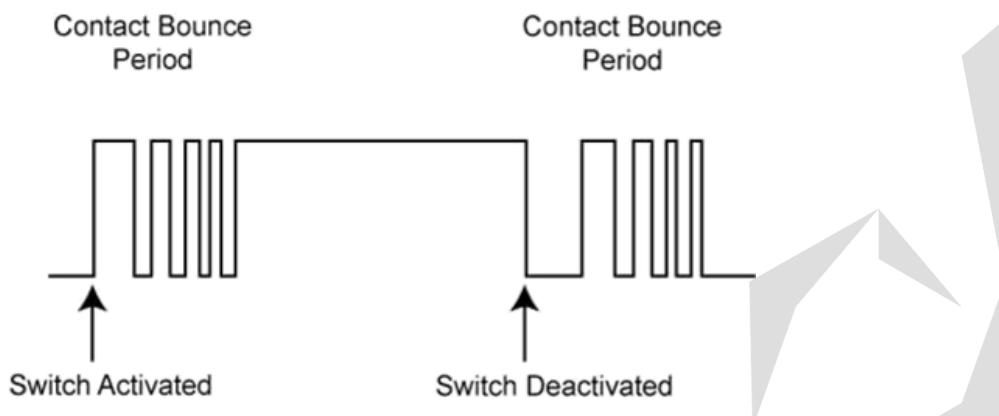
En las subrutinas de tiempo primero debemos cargar el tiempo que queremos que se tarde en ejecutarse en forma decimal poniendo `.número_decimal` y después las mandamos a llamar con la instrucción **CALL**.

Función Antirrebote

Una de las aplicaciones de las subrutinas de tiempo es el antirrebote, que evita que se capten “falsos unos” y “falsos ceros” durante el pulsamiento de botones mecánicos (particularmente este error ocurre más cuando usamos push buttons), por lo que se usan este tipo de subrutinas que dejan pasar cierto tiempo después de que ha presionado un botón, logrando así que el programa no capte estos errores.

Si no se ejecuta una función antirrebote, este error donde el botón mecánico rebota puede ser percibido por el microcontrolador como varios ceros y unos en forma de ruido, confundiendo así al programa, normalmente estos tiempos de retardo por norma son de 25 milisegundos y se deben seguir ciertos pasos para que funcione correctamente el antirrebote.

1. **Esperar a que se apriete el interruptor:** Esto pasa cuando llegue un 1 lógico al pin del puerto conectado al botón.
2. **Dejar pasar el tiempo de antirrebote de 25 milisegundos.**
3. **Esperar a que se suelte el botón:** Esto pasa cuando llegue un 0 lógico al pin del puerto conectado al botón.
4. **Dejar pasar el tiempo de antirrebote de 25 milisegundos otra vez.**
5. **Ejecutar la tarea activada por el push button.**



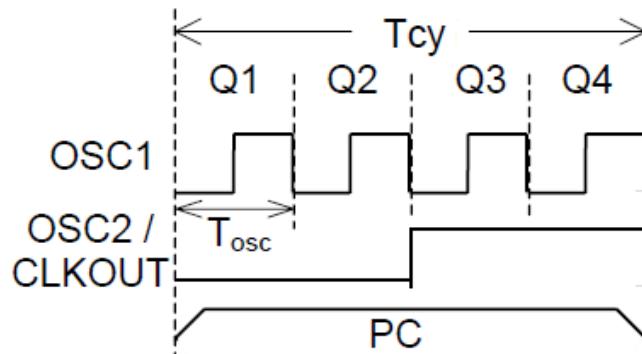
Subrutina de Tiempo: Ciclo de Máquina

Ya en el código las subrutinas de tiempo se crean por medio de un contador, que contará el tiempo de retardo deseado y al terminar saldrá de la subrutina y seguirá con la ejecución del programa.

Cada subrutina de tiempo ya sea la de 1, 2 o 3 variables cuenta con su propia ecuación y usaremos una u otra dependiendo del tiempo de retardo que se desee alcanzar, mientras mayor sea el tiempo de retardo, mayor será el número de variables. Pero antes debo considerar el concepto de ciclo de máquina para que pueda entender lo que representa.

Ciclo de máquina (cm): Es el proceso realizado por el microcontrolador desde que toma una instrucción de la memoria FLASH hasta que toma la siguiente, se le llama ciclo de máquina ya que el microcontrolador solo puede ejecutar una tarea a la vez. Para ejecutar una instrucción del código ensamblador se requieren 4 ciclos de reloj, que conforman un solo ciclo de máquina (específicamente en el microcontrolador PIC porque utiliza una arquitectura Harvard), considerando los siguientes pasos que conforman un solo ciclo de máquina o cm:

- **Q1:** El microprocesador obtiene del **contador del programa (PC o Program Counter)** la dirección de la siguiente instrucción a ejecutar.
- **Q2:** El **contador del programa** se incrementa en 1.
- **Q3:** El microprocesador extrae el contenido de la instrucción a ejecutar.
- **Q4:** El microprocesador decodifica y ejecuta la instrucción.



$$1 \text{ ciclo de máquina} = 1 \text{ cm} = 4 \text{ ciclos de reloj [segundos]}$$

$$1 \text{ ciclo de reloj} = \text{Periodo oscilador} = 1 \text{ Periodo} = 1 T = \frac{1}{\text{frecuencia oscilador}} = \frac{1}{f}$$

Cuando en el PIC16F887 se está usando el oscilador interno (sin modificar el divisor de reloj), su frecuencia es de 4MHz.

$$1 \text{ cm} = 4 T = \frac{4}{f} = \frac{4}{4 \times 10^6} = 1 \mu\text{s}$$

Además, es importante mencionar que las variables de las subrutinas de tiempo solo pueden abarcar valores de 1 a 256 porque son valores dados por números binarios de 8 bits, donde $1111\ 1111 = 255$ pero como se usa un contador para crear la subrutina de tiempo, no se puede declarar una variable como 0, siempre valor mínimo 1.

Subrutina de Tiempo de 1 Variable

Esta subrutina puede alcanzar tiempos de retardo de 11 [μs] a 1,541 [μs] = 1.541 [ms] y su ecuación se obtiene al contar los ciclos de máquina que se ejecutan desde que se llama la subrutina hasta que se termina, cabe mencionar que var1 puede tomar valores de 1 a 256 porque es un número binario de 8 bits:

- Cargar el valor de retardo a la subrutina: **El programa se tarda 2 ciclos de máquina (cm) en ejecutar esta acción.**
 - **MOVLW k:**
 - Coloca directamente en el acumulador W un número cualquier guardado en el valor literal k de 8 bits.
 - Esto se hace para utilizar al acumulador como punto intermedio de ingreso de datos al programa.
 - Tarda 1 ciclo de máquina en ejecutarse.
 - **MOVWF f:**
 - Lee el contenido del acumulador W y lo coloca en un registro de la RAM indicado por la dirección f, para las subrutinas de tiempo de 1 variable se usa el registro 0x60.
 - Esto sirve para pasar el contenido de un registro de RAM a otro, aunque para ello el contenido del registro de entrada previamente se debe haber guardado en el acumulador W con la instrucción **MOVF**.
 - Tarda 1 ciclo de máquina en ejecutarse.

$$\#cm = 1_{MOVLW} + 1_{MOVWF} = 2_{CARGA}$$

- Mandar a llamar la subrutina de tiempo de 1 variable: **El programa se tarda 2 cm en ejecutar esta acción.**
 - **CALL k:**
 - Sirve para ejecutar una subrutina, una subrutina es el equivalente a una función en cualquier otro lenguaje de programación.
 - **Tarda 2 ciclos de máquina en ejecutarse.**

$$\#cm = 2_{CARGA} + 2_{CALL}$$

```
;SUBRUTINA DE TIEMPO DE 1 VARIABLE: Abarca un tiempo de retardo de 11 us
;(microsegundos) a 1,541 us = 1.541 ms (milisegundos).
;Las variables de mi subrutina se pueden colocar en los registros de
;propósito general que quiera, pero para tener un orden, las variables 1, 2
;o 3 de las subrutinas de tiempo las guardaré en los registros de propósito
;general que vayan de la dirección 0X60 a la 0X68 del banco 0 de la memoria
;RAM en el PIC y las declararé en forma decimal de la siguiente manera:
;.numero_decimal
;Usa el siguiente registro de propósito general para guardar la variable:
;Registro 0X60 para guardar la variable 1.
;MOVLW K: Coloca directamente en el acumulador W un número binario
;cualquiera de 8 bits dado por el valor literal K. Como en este caso
;lo usaré para colocar una variable de mi subrutina de tiempo, se
;declara en forma decimal, poniendo .número_decimal.
MOVLW .25 ;W = var1 = .25 = 25 decimal
;MOVWF F: Lee el contenido del acumulador W y lo coloca en un
;registro de la RAM indicado por la dirección F. En este el
;contenido del acumulador W se coloca en un registro de propósito
;general F que sirve para guardar un valor en la memoria RAM (que se
;borrará cuando se reinicie el PIC), indicado en forma hexadecimal,
;poniendo 0Xnúmero_hexadecimal.
MOVWF 0X60 ;0X60 = W = var1 = 25 decimal
;CALL k: Sirve para llamar una subrutina por su nombre y ejecutar
```

;una acción que se pueda repetir varias dentro del mismo código, de
;esta manera esa acción no la debo escribir varias veces.
CALL ST1V ;Llama la Subrutina de Tiempo de 1 Variable (ST1V).

- Dentro de la subrutina de tiempo de 1 variable se ejecuta el retardo:
 - Se deja pasar cierto tiempo para la calibración del retardo: **El programa se tarda 1 cm en ejecutar cada iteración de esta acción, por lo que se maneja como una variable llamada #NOPS.**

▪ **NOP:**

- Deja pasar un ciclo de máquina sin hacer nada.
- Su uso es fundamental para el manejo de tiempos, osea para que cada instrucción del microcontrolador se ejecute en el momento deseado.
- **Tarda 1 ciclo de máquina en ejecutarse.**

$$\#cm = 2_{CARGA} + 2_{CALL} + \#NOPS$$

- Se crea un condicional que reste el valor del retardo ingresado a la subrutina de tiempo hasta que su valor se vuelva cero, logrando así que se ejecute un conteo regresivo: **El programa se tarda 1 cm en ejecutar esta acción si la resta de la condición no es igual a cero y tarda 2 cm cuando el resultado de la resta sea cero.**

▪ **DECFSZ f, d:**

- a. **Esta operación es lo más parecido a un condicional if que existe en el idioma ensamblador, normalmente se usa para evitar instrucciones GOTO.**
- b. Esta operación decrementa (le resta 1) a lo que haya en la dirección f del registro de RAM indicado y el resultado lo guarda en el acumulador W o en el mismo registro F, además dependiendo de si el resultado es cero o no, se brincará la siguiente instrucción o la ejecutará normalmente.
- c. La instrucción resta y evalúa el número al mismo tiempo.
 - **Si el resultado es cero (00 hexadecimal):**
 - **Se brinca la siguiente instrucción que haya en el código (aumenta en 1 al contador de programa).**
 - **Tarda 2 ciclos de máquina en ejecutarse.**

$$\#cm = 2_{CARGA} + 2_{CALL} + \#NOPS + 2_{DECFSZ=0}$$

- **Si el resultado NO es cero:**

- Sigue la ejecución normal del código.
- **Tarda 1 ciclo de máquina en ejecutarse.**

$$\#cm = 2_{CARGA} + 2_{CALL} + \#NOPS + 2_{DECFSZ=0} + 1_{DECFSZ \neq 0}$$

- Se repite el condicional hasta que el conteo del temporizador sea igual a cero: **El programa se tarda 2 cm en ejecutar esta acción.**

▪ **GOTO k:**

- d. **Sirve para hacer que el programa brinque a otra parte del programa indicado por el parámetro literal k, la parte a donde**

quiero que brinque el programa se puede indicar por nombre, por número del contador de programa (PC) o por número de instrucciones hacia atrás de la instrucción GOTO, para así lograr que el programa se ejecute indefinidamente.

e. **Tarda 2 ciclos de máquina en ejecutarse.**

$$\#cm = 2_{CARGA} + 2_{CALL} + \#NOPS + 2_{DECFSZ=0} + 1_{DECFSZ \neq 0} + 2_{GOTO}$$

- Ya que se haya llegado a cero, se ejecuta la instrucción RETURN para salir de la subrutina:

- **RETURN :**

- **Sirve para terminar una subrutina,** una subrutina es el equivalente a una función en cualquier otro lenguaje de programación.
- **Las subrutinas se usan para ejecutar una acción que se necesita recurrentemente en el código** y siempre se ejecutan con la instrucción CALL y se terminan con la instrucción RETURN.
- **Tarda 2 ciclos de máquina en ejecutarse.**

$$\#cm = 2_{CARGA} + 2_{CALL} + \#NOPS + 2_{DECFSZ=0} + 1_{DECFSZ \neq 0} + 2_{GOTO} + 2_{RETURN}$$

```
;*****
;SUBRUTINA DE TIEMPO DE 1 VARIABLE: Puede abarcar tiempos de retardo de
;11 us (microsegundos) a 1.541 ms (milisegundos) ya que varl en su ecuación
;solo puede adoptar valores de 1 a 256.
;SE USA EL REGISTRO DE PROPÓSITO GENERAL 0X60.
;Ecuación: En la ecuación, #cm es el tiempo de retardo que quiero obtener y
;lo debo introducir en microsegundos (us).
;#cm = 5 + (varl) (#NOPS + 3)
;-----Bucle con varl guardada en el registro 0X60-----
;NOP: Esta instrucción no hace nada, solamente sirve para dejar
;pasar 1 ciclo de máquina (cm). Se puede declarar un número
;cuálquiera de NOPs pero en el curso dentro de las subrutinas de
;tiempo siempre usaremos #NOPS = 3 para que sea un dato conocido.
STIV: NOP
      NOP
      NOP ;#NOPS = 3
;AHORA VAMOS A USAR UNA DE LAS 35 OPERACIONES QUE ME PERMITEN REALIZAR
;UN BUCLE PARECIDO AL CICLO FOR Y SE USA PARA CREAR UN CONTADOR.
;DECFSZ F, D: Esta operación decremente (le resta 1) a lo que haya en
;la dirección F del registro de RAM indicado y el resultado lo guarda en
;el acumulador W o en el mismo registro F, además dependiendo de si el
;resultado del decremento es cero o no, se brincará la siguiente instrucción
;o la ejecutará normalmente:
;Si el resultado del decremento NO es cero:
;Sigue la ejecución normal del código.
;Dura 1 ciclo de máquina su ejecución.

;Si el resultado del decremento es cero (0X00):
;SE BRINCA LA SIGUIENTE INSTRUCCIÓN QUE HAYA EN EL CÓDIGO (aumenta
;en 1 al contador de programa o PC).
;Dura 2 ciclos de máquina su ejecución.
;HACE QUE EL PROGRAMA SALGA DE LA SUBRUTINA DE TIEMPO DE 1 VARIABLE.
DECFSZ 0X60,F
;GOTO k: Sirve para hacer que el programa brinque a otra parte del
;código, indicado por una directiva EQU que tenga el nombre de la
;parte del código a donde quiero que brinque el programa.
GOTO STIV
;RETURN: Todas las subrutinas siempre deben terminar con la
;instrucción return, lo que hace RETURN es bajar un nivel a la pila
;y cargar en el PC el valor que esté guardado ahí, haciendo que el
;programa brinque a la parte del código después de la instrucción
;CALL que llamó esta subrutina, siguiendo así la ejecución normal
;que llevaba el programa.
;-----Bucle con varl guardada en el registro 0X60-----
RETURN
```

Ahora lo que se debe hacer es agrupar los elementos para considerar cuales tienen que ver con el valor ingresado en la variable que representa var1, al hacerlo y despejar la ecuación se llega a su expresión final.

$$\#cm = 2_{CARGA} + 2_{CALL} + \#NOPS + 2_{DECFSZ=0} + 1_{DECFSZ \neq 0} + 2_{GOTO} + 2_{RETURN}$$

$$\#cm = 2_{CARGA} + 2_{CALL} + \#NOPS + 1_{DECFSZ \neq 0} + 2_{GOTO} + 2_{DECFSZ=0} + 2_{RETURN}$$

Para llegar a la expresión final de la ecuación plantearemos un ejemplo donde la variable introducida es $var1 = 4$ y veremos como se comportan los ciclos de máquina durante la ejecución del código. Es muy importante recordar que la instrucción **DECFSZ** que ejecuta el condicional, resta y evalúa el número de la condición al mismo tiempo.

$$var1 = 4; \quad \#cm = 1_{MOVlw} + 1_{MOVwf} + 2_{CALL} + \#NOPS$$

DECFSZ = var1 = 4 - 1 = 3 \neq 0; Resta y evaluación en la misma línea de código

$$var1 = 3; \quad \#cm = 1_{DECFSZ \neq 0} + 2_{GOTO} + \#NOPS$$

DECFSZ = var1 = 3 - 1 = 2 \neq 0; Resta y evaluación en la misma línea de código

$$var1 = 2; \quad \#cm = 1_{DECFSZ \neq 0} + 2_{GOTO} + \#NOPS$$

DECFSZ = var1 = 2 - 1 = 1 \neq 0; Resta y evaluación en la misma línea de código

$$var1 = 1; \quad \#cm = 1_{DECFSZ \neq 0} + 2_{GOTO} + \#NOPS$$

DECFSZ = var1 = 1 - 1 = 0; Resta y evaluación en la misma línea de código

$$var1 = 0; \quad \#cm = 2_{DECFSZ=0} + 2_{RETURN}$$

Se reorganizan las ecuaciones para que sea más evidente como se pueden agrupar los elementos en función del valor de la variable de la subrutina de tiempo:

$$\#cm = 1_{MOVlw} + 1_{MOVwf} + 2_{CALL} + \#NOPS + 1_{DECFSZ \neq 0} + 2_{GOTO} + 2_{DECFSZ=0} + 2_{RETURN}$$

$$\#cm = \#NOPS + 1_{DECFSZ \neq 0} + 2_{GOTO}$$

$$\#cm = \#NOPS + 1_{DECFSZ \neq 0} + 2_{GOTO}$$

$$\#cm = \#NOPS + 1_{DECFSZ \neq 0} + 2_{GOTO}$$

Las 4 ecuaciones obtenidas se pueden condensar en una sola de la siguiente forma, y tomando en cuenta el valor de la variable en el ejemplo descrito se puede dejar todo en función de las variables **#NOPS** y **var1**:

$$\#cm = 1 + 1 + 2 + 4 * (\#NOPS) + 3 * (1 + 2) + 2 + 2; \quad var1 = 4$$

$$\#cm = 4 + 4 * (\#NOPS) + (3) * (3) + 4 = 4 + (var1) * (\#NOPS) + (var1 - 1) * (3) + 4$$

$$\#cm = 4 + (var1) * (\#NOPS) + (var1 - 1) * (3) + 4$$

$$\#cm = (var1) * (\#NOPS) + 3 * var1 - 3 + 8$$

$$\#cm = (var1) * (\#NOPS) + (var1) * (3) + 5 = (var1) * (\#NOPS + 3) + 5$$

Por lo tanto, la ecuación final que puede alcanzar tiempos de retardo de $11 \mu s$ a $1,541 \mu s = 1.541 ms$, considerando que su número de NOPs es de 3, es la siguiente, el número de NOPs se puede cambiar para calibrar o hasta extender el tiempo que alcanza la subrutina de tiempo de 1 variable:

$$\#cm = 5 + (var1) * (\#NOPS + 3) [\mu s]$$

Subrutina de tiempo de 2 variables

Esta subrutina se obtiene al anidar 2 subrutinas de tiempo de 1 variable y puede alcanzar tiempos de retardo de $17 \mu s$ a $394,247 \mu s = 394.247 ms$ y su ecuación se obtiene de forma muy parecida a como se obtuvo la de la subrutina de tiempo de 1 variable, realizando el mismo análisis, pero ahora con 2 variables que indiquen el tiempo de retardo, donde por convención var1 se guarda en el registro 0x61 y var2 se guarda en el registro 0x62.

Para llegar a la expresión final de la ecuación plantearemos un ejemplo donde las variables introducidas son $var1 = 5$ y $var2 = 3$, con ellas veremos el comportamiento de los ciclos de máquina durante la ejecución del código. Es muy importante recordar que la instrucción **DECFSZ** que ejecuta el condicional, resta y evalúa el número de la condición al mismo tiempo.

```
#cm = 4CARGAVar1 + 2CALL + 2CARGAVar2 + #NOPS + 1DECFSZ≠0Var1 + 2GOTDVar1 + 2DECFSZ=0Var1 + 1DECFSZ≠0Var2 + 2GOTDVar2 + 2DECFSZ=0Var2 + 2RETURN

#cm = #NOPS + 1DECFSZ≠0Var1 + 2GOTDVar1

#cm = #NOPS + 1DECFSZ≠0Var1 + 2GOTDVar1

#cm = #NOPS + 1DECFSZ≠0Var1 + 2GOTDVar1

#cm = #NOPS

#cm = 2CARGAVar2 + #NOPS + 1DECFSZ≠0Var1 + 2GOTDVar2 + 2DECFSZ=0Var1 + 1DECFSZ≠0Var2 + 2GOTDVar2

#cm = #NOPS + 1DECFSZ≠0Var1 + 2GOTDVar2

#cm = #NOPS + 1DECFSZ≠0Var1 + 2GOTDVar2

#cm = #NOPS + 1DECFSZ≠0Var1 + 2GOTDVar2

#cm = #NOPS

#cm = 2CARGAVar2 + #NOPS + 1DECFSZ≠0Var1 + 2GOTDVar2 + 2DECFSZ=0Var1

#cm = #NOPS + 1DECFSZ≠0Var1 + 2GOTDVar2

#cm = #NOPS + 1DECFSZ≠0Var1 + 2GOTDVar2

#cm = #NOPS + 1DECFSZ≠0Var1 + 2GOTDVar2

#cm = #NOPS
```

Las ecuaciones obtenidas se pueden condensar en una sola tomando en cuenta el valor de las variables en el ejemplo descrito, dejando todo en función de las variables **#NOPS**, **var1** y **var2**:

$$\#cm = 10 + 2 * (var2) + (\#NOPS) * var1 * (var2) + 3 * (var1 - 1) * (var2) + 2 * (var2) + 3 * (var2 - 1)$$

Por lo tanto, la ecuación final que puede alcanzar tiempos de retardo de $17 \mu s$ a $394,247 \mu s = 394.247 ms$, considerando que su número de NOPs es de 3, es la siguiente, el número de NOPs se puede cambiar para calibrar o hasta extender el tiempo que alcanza la subrutina de tiempo de 2 variables:

$$\#cm = 7 + (var2) * (4 + (\#NOPS + 3) * (var1)) [\mu s]$$

Subrutina de tiempo de 3 variables

Esta subrutina puede alcanzar tiempos de retardo de 23 [μs] a 100,926,473 [μs] = 100.926[s], se basa en el mismo análisis matemático de las 2 subrutinas anteriores pero esta vez anidando tres subrutinas de tiempo y su ecuación final es la siguiente:

$$\#cm = 9 + (var1) * (var3 * (var2 * (\#NOPS + 3) + 4) + 4) [\mu s]$$

Interrupciones

Hay eventos y señales que puede recibir el microcontrolador de forma interna o externa a través de sus pines; y si el programador lo permite, pueden desviar la secuencia del programa que se está ejecutando para ir a ejecutar una acción diferente llamada **Rutina de Servicio de interrupción (RSI o ISR)**, al terminar de ejecutarse, la ISR se debe regresar a una línea después de la última instrucción que se ejecutó del programa principal. A esta secuencia de desviar mediante medios externos el flujo de ejecución del programa se le llama interrupción.

Las interrupciones son parecidas a las subrutinas, pero las subrutinas deben ser convocadas, mientras las interrupciones llegan solas e interrumpen la ejecución normal del programa, aunque para que pueda hacerlo es necesario darle permiso a la ejecución de hacerlo. Aunque, si no se le da permiso al microcontrolador por medio del código, se levantará una bandera indicando que la interrupción se está tratando de activar.

Las interrupciones también pueden servir para reemplazar por la instrucción GOTO que pregunta si un botón ya se presionó o no; ya que, si esto no se ha cumplido, el PIC está gastando recursos en preguntar y con una interrupción esto se podría hacer sin gastar los recursos del PIC.

Tipos de Interrupciones

Al ocurrir una interrupción, el PIC se carga con la dirección de la memoria del programa en donde se debe escribir la ISR, esta dirección es llamada Vector de Interrupción y dependiendo del microcontrolador puede haber diferentes vectores de interrupción dependiendo del tipo de periférico que esté generando la interrupción. Existen los siguientes tipos:

- **Reset:** Para algunos microcontroladores la terminal reset al ser activada genera una interrupción que desvía la secuencia hacia un vector de interrupción en donde el programador debe escribir el código necesario para reestablecer las condiciones generales del microcontrolador.
- **Software:** Hay condiciones especiales que al cumplirse durante la ejecución de un código generan una interrupción, checando así el estado de las operaciones del código ensamblador, como por ejemplo al ejecutar una instrucción de división con valor de 0 como denominador.
- **Hardware:** Son aquellas que llegan del exterior a través de las terminales del dispositivo, checando el estado de los puertos del PIC. También se incluyen las interrupciones que se generan desde el interior a través de sus periféricos.

También existe una jerarquía de prioridad en los tipos de interrupciones:

- Interrupciones Mascarables: Pueden ser interrumpidas por otra interrupción de mayor jerarquía, entran casi al momento de ser solicitada la atención del procesador.
- Interrupciones No Mascarables: Tienen la máxima prioridad y no pueden ser interrumpidas por nada.

El único registro de interrupción en el PIC16F887, que es el 0X004 de la memoria FLASH es de tipo No Mascarable, lo que significa que es de máxima prioridad y no puede ser interrumpido, estas tardan 4 ciclos de máquina en ejecutarse y requieren de un permiso individual de interrupción, así como de la, algunas fuentes de interrupción necesitan de permisos adicionales.

Aunque sean de máxima prioridad, las interrupciones pueden tardar hasta 4 cm en ejecutarse y si no cuenta con la **Habilitación Global de Interrupciones (GIE)**, no importa que se active, esta no hará nada.

Dependiendo del tipo de interrupción, ya sea de la memoria RAM, EEPROM, del convertidor analógico digital, etc. existe un tipo de permiso que se le debe dar. Aunque el que puede mandar a todos es el denominado GIE, el cual debe estar activado para que luego específicamente se le pueda dar permiso a cada tipo de interrupción.

Hasta ahora se ha intentado hacer interrupciones cuando se utilizan los puertos, pero como no han tenido permiso, no han hecho nada en el PIC. Cada periférico tiene su propio permiso para una interrupción. Además, para regresar de una interrupción se usa la instrucción RETFIE y hacen uso de los niveles de la Pila.

Todas las fuentes de interrupción cuentan con una Bandera de Interrupción (Interrupt Flag). Para que ocurra una interrupción, es necesario que se hayan dado los permisos necesarios y que se levante la bandera de interrupción correspondiente al periférico que solicita la atención.

Los bits que habilitan a los periféricos como fuente de interrupción tienen un nombre que finaliza con las letras IE, mientras que los que corresponden a las banderas de interrupción tienen un nombre que finaliza con las letras IF (Interrupt Flag).

No existe límite en el número de interrupciones que se pueden crear, aunque si se está atendiendo a una, se debe terminar de ejecutar para que se pueda atender la segunda interrupción.

Proceso de una Interrupción en el PIC16F887

Después de proporcionar los permisos necesarios para que un periférico sea fuente de interrupción en el PIC, cuando el periférico cumple con su misión:

1. Se levanta la bandera IF (Interrupt Flag) que tiene asignada.
2. Termina la ejecución de la instrucción en curso.
3. Quita la Habilitación Global de Interrupciones (Global Interrupt Enable, GIE), y por tal motivo no puede interrumpir otra interrupción.
4. El contenido del PC se guarda en la PILA, específicamente en la dirección que señala su apuntador actualmente, de esta manera siempre regresa a una instrucción después de la última ejecutada antes de atender a la interrupción.
5. El apuntador de la Pila se incrementa en 1.
6. El PC (Program Counter) se carga con la dirección 0X004, que es el vector de interrupciones.

Rescate y recuperación de 3 Registros Importantes durante las interrupciones.

```

PROCESSOR 16F887
__CONFIG 0X2007, 0X1FE4
__CONFIG 0X2008, 0X3FFF
INCLUDE <P16F887.INC>
CBLOCK 0X70
W_R, ST_R, PC_R
ENDC
ORG 0X0000
GOTO INICIO
ORG 0X0004
MOVWF W_R
SWAPF STATUS, W
MOVWF ST_R
CLRF STATUS
MOVE PCLATH, W
MOVWF PC_R
CLRF PCLATH; EN REALIDAD PCLATH DEBE SER ACOMODADO
;DE ACUERDO A LA DIRECCIÓN A LA QUE SE SALTE
GOTO RSI
...
;AQUÍ PUEDEN IR LAS SUBRUTINAS QUE
;TENEMOS EN ARCHIVOS
...
INICIO: ...
;AQUI VA EL CÓDIGO PRINCIPAL
...
RSI: ...
;Aqui debes escribir la RSI
;NO OLVIDAR BAJAR BADERAS INT
MOVE PC_R, W
MOVWF PCLATH
SWAPF ST_R, W
MOVWF STATUS
SWAPF W_R, F

```

Interrupción por la terminal RB0/INT

La terminal RB0/INT al estar configurada como entrada, puede interrumpir al levantar su bandera INTF (INTCON, 1) cuando detecta cambios de estado, para configurar que cambio del estado es el que levanta la bandera que usa:

REGISTER 2-2: OPTION_REG: OPTION REGISTER

R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1
RBPU	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0
bit 7							bit 0

Legend:

R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'
-n = Value at POR	'1' = Bit is set	'0' = Bit is cleared x = Bit is unknown

bit 7 **RBPU:** PORTB Pull-up Enable bit

- 1 = PORTB pull-ups are disabled
- 0 = PORTB pull-ups are enabled by individual PORT latch values

bit 6 **INTEDG:** Interrupt Edge Select bit

- 1 = Interrupt on rising edge of INT pin
- 0 = Interrupt on falling edge of INT pin

Para que sea Fuente de interrupción se deben proporcionar los siguientes permisos:

- **INTE (INTCON, 4):** Se debe escribir un 1 lógico, proporcionando el permiso individual del pin.
- **GIE (INTCON, 7):** Se debe escribir un 1 lógico, proporcionando el permiso Global de Interrupciones.