

## Guia d'introducció a la utilització de microcontroladors PIC

### Objectius i àmbit

Aquest document vol ser una introducció a la programació així com a les generalitats a l'hora de dissenyar hardware extern connectat a microcontroladors. Aquesta guia es limita a parlar sobre la família anomenada intermitja dels microcontroladors, de 14 bits, la 16FXXX que inclou els dispositius més utilitzats i més flexibles donada la seva memòria flash que en permet una programació i reprogramació ràpida i sense necessitat de borrar el contingut de la memòria, això els fa una opció ideal com a sistema d'aprenentatge en aplicacions educatives degut a que escurça el temps de desenvolupament, els mitjans, i degut al seu set d'instruccions reduït RISC de només 35 instruccions també escurça la necessitat de memoritzar els mnemotècnics de les instruccions. A més disposen d'una gran varietat de perifèrics integrats en el mateix circuit integrat; USART's, A/D's, Timers, PWM's, Comptadors, Ports sèrie síncrons, EEPROM's... tots ells programables i basats en un sistema interruptiu molt aconseguit i optimitzat. També es tracta d'una família molt flexible ja que es disposa de diferents models amb diferents capacitats de memòria, perifèrics i port's. Bàsicament en aquesta guia es tractaran els PIC's 16F84A i 16F877, els dos PIC's que podríem dir que es troben en els extrems de la família, ja que un és el més limitat en memòria, en nombre de ports i en perifèrics, i l'altre és el que disposa de més memòria, més ports i més perifèrics en aquesta família. També en funció del dispositiu utilitzat es podrà treballar en un marge de tensions d'alimentació més ampli (de 3'3 a 5 V) i a més freqüència, cal tenir en compte que el consum del microcontrolador augmentarà de forma gairebé lineal en funció de la freqüència a la que treballi.

Tot el codi inclòs en aquesta guia està pensat per utilitzar els entorns de programació que proporciona el fabricant dels microcontroladors Microchip® i és totalment funcional en l'entorn MPLAB®. Tot i que originalment estan pensats per ser utilitzats en aquesta família de microcontroladors es poden extrapolar fàcilment a altres famílies sense majors problemes.

En aquesta guia hi aparèixen gràfics i traduccions, pràcticament literals, dels datasheets del mateix fabricant per tal d'aprofundir en els coneixements exposats per aquest tipus de documentació.

### Taula de selecció

El primer a tenir en compte és una bona selecció del dispositiu a utilitzar en la aplicació a desenvolupar. Normalment la tria del dispositiu ens vindrà només limitada al nombre de ports del dispositiu, però també cal tenir molt en compte l'extensió del programa a realitzar, ja que tot i que normalment ens sobra la memòria de la que disposem, en segons quina aplicació o entorn de programació (sobretot entorns d'alt nivell tipus C) ens podem trobar amb que no tinguem recursos RAM i ROM suficients. Normalment els sistemes de comunicació no acostumen a ser crítics, ja que si un dispositiu no disposa de perifèrics integrats sempre es pot realitzar *bit banging*<sup>1</sup>, tot i que acostuma a ser aconsellable que disposi d'aquests registres per poder optimitzar al

---

<sup>1</sup> S'anomena *bit banging* al procés pel qual s'implementa un protocol de comunicació basant-se en la major velocitat de processat d'un dispositiu respecte al protocol, de manera que es pot implementar sense necessitat de disposar del hardware específic. Aquest sistema consumeix més recursos que si es disposés dels registres hardware específics, però ens permet una correcte implementació.

màxim el programa i el sistema interruptiu. La freqüència de treball tampoc acostuma a ser un aspecte crític, ja que normalment 4 MHz serà una freqüència més que acceptable.

Tot seguit hi ha la taula de selecció de la família en qüestió, actualitzada a 15 de Juny del 2002.

PICmicro<sup>®</sup> MCU Family Products

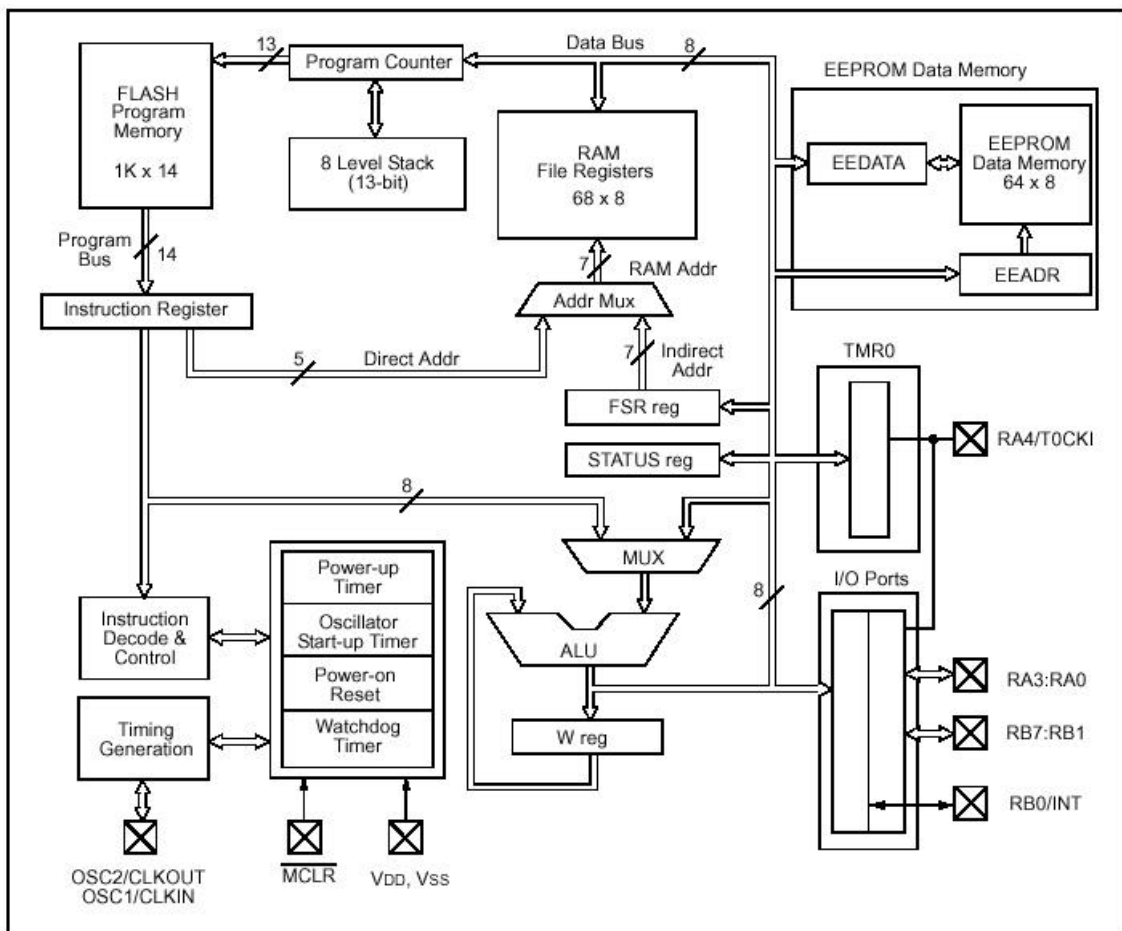
PICmicro <sup>®</sup> MICROCONTROLLER FAMILY PRODUCTS																		
Product	Program Memory			EEPROM Data Memory Bytes	RAM Bytes	IO Pins	Packages	Analog			Digital			Max. Speed MHz	ICSP™	BOR/ PLVD	CCP/ ECCP	Other Features
	Bytes	OTP/ FLASH Words	ROM Words					8-Bit ADC Channels	Comparators	PWM 10-Bit	Timers/WDT	Serial I/O						
PIC16FXXX FLASH MCU: Upwardly Compatible with PIC16CXXX/PIC16C5X/PIC12CXXX, 4-12 Interrupts, 200ns Instruction Execution, 35 Instructions, 4 Oscillator Selections, 25mA source/sink per I/O (continued)																		
PIC16F74	7168 (FLASH)	4096x14 (FLASH)	—	—	192	33	40P, 44L, 44PT	8	—	2	1-16 bit, 2-8 bit, 1-WDT	USART/ I <sup>2</sup> C/SPI	20	✓	✓	—	2	PSP, Self-read
PIC16F76	14336 (FLASH)	8192x14 (FLASH)	—	—	388	22	28SP, 28SO, 28SS, 28ML	5	—	2	1-16 bit, 2-8 bit, 1-WDT	USART/ I <sup>2</sup> C/SPI	20	✓	✓	—	2	Self-read
PIC16F77	14336 (FLASH)	8192x14 (FLASH)	—	—	388	33	40P, 44L, 44PT	8	—	2	1-16 bit, 2-8 bit, 1-WDT	USART/ I <sup>2</sup> C/SPI	20	✓	✓	—	2	PSP, Self-read
PIC16F84A	1792 (FLASH)	1024x14 (FLASH)	—	64	68	13	18P, 18SO, 20SS	—	—	—	1-8 bit 1-WDT	—	20	✓	—	—	—	—
PIC16F870	3584 (FLASH)	2048x14 (FLASH)	—	64	128	22	28SP, 28SO, 28SS	5 (10-bit)	—	1	1-16 bit, 2-8 bit, 1-WDT	AUSART	20	✓	✓	—	1	Self-Programming, ICD
PIC16F871	3584 (FLASH)	2048x14 (FLASH)	—	64	128	33	40P, 44L, 44PT	8 (10-bit)	—	1	1-16 bit, 2-8 bit, 1-WDT	AUSART	20	✓	✓	—	1	PSP, Self-Programming, ICD
PIC16F872	3584 (FLASH)	2048x14 (FLASH)	—	64	128	22	28SP, 28SO, 28SS	5 (10-bit)	—	1	1-16 bit, 2-8 bit, 1-WDT	M <sup>2</sup> C/SPI	20	✓	✓	—	1	ICD, Self-Programming
PIC16F873	7168 (FLASH)	4096x14 (FLASH)	—	128	192	22	28SP, 28SO	5 (10-bit)	—	2	1-16 bit, 2-8 bit, 1-WDT	AUSART/ M <sup>2</sup> C/SPI	20	✓	✓	—	2	ICD, Self-Programming
PIC16F873A*#	7168 (FLASH)	4096x14 (FLASH)	—	128	192	22	28SP, 28SO, 28SS, 28ML	5 (10-bit)	2	2	1-16 bit, 2-8 bit, 1-WDT	AUSART/ M <sup>2</sup> C/SPI	20	✓	✓	—	2	ICD, Self-Programming
PIC16F874	7168 (FLASH)	4096x14 (FLASH)	—	128	192	33	40P, 44L, 44PQ, 44PT	8 (10-bit)	—	2	1-16 bit, 2-8 bit, 1-WDT	AUSART/ M <sup>2</sup> C/SPI	20	✓	✓	—	2	PSP, Self-Programming, ICD
PIC16F874A*	7168 (FLASH)	4096x14 (FLASH)	—	128	192	33	40P, 44L, 44PT	8 (10-bit)	2	2	1-16 bit, 2-8 bit, 1-WDT	AUSART/ M <sup>2</sup> C/SPI	20	✓	✓	—	2	PSP, Self-Programming, ICD
PIC16F876	14336 (FLASH)	8192x14 (FLASH)	—	256	388	22	28SP, 28SO	5 (10-bit)	—	2	1-16 bit, 2-8 bit, 1-WDT	AUSART/ M <sup>2</sup> C/SPI	20	✓	✓	—	2	ICD, Self-Programming
PIC16F876A*#	14336 (FLASH)	8192x14 (FLASH)	—	256	388	22	28SP, 28SO, 28SS, 28ML	5 (10-bit)	2	2	1-16 bit, 2-8 bit, 1-WDT	AUSART/ M <sup>2</sup> C/SPI	20	✓	✓	—	2	ICD, Self-Programming
PIC16F877	14336 (FLASH)	8192x14 (FLASH)	—	256	388	33	40P, 44L, 44PQ, 44PT	8 (10-bit)	—	2	1-16 bit, 2-8 bit, 1-WDT	AUSART/ M <sup>2</sup> C/SPI	20	✓	✓	—	2	ICD, PSP, Self-Programming
PIC16F877A*	14336 (FLASH)	8192x14 (FLASH)	—	256	388	33	40P, 44L, 44PT	8 (10-bit)	2	2	1-16 bit, 2-8 bit, 1-WDT	AUSART/ M <sup>2</sup> C/SPI	20	✓	✓	—	2	ICD, PSP, Self-Programming
PIC16F877	1792 (FLASH)	1024x14 (FLASH)	—	128	224	16	18P, 18SO, 20SS	—	2	1	1-16 bit, 2-8 bit, 1-WDT	AUSART	20	✓	✓	—	1	Prog. V <sub>DD</sub> , 4MHz internal clock oscillator
PIC16F878	3584 (FLASH)	2048x14 (FLASH)	—	128	224	16	18P, 18SO, 20SS	—	2	1	1-16 bit, 2-8 bit, 1-WDT	AUSART	20	✓	✓	—	1	Prog. V <sub>DD</sub> , 4MHz internal clock oscillator
PIC16F72#	3584 (FLASH)	2048x14 (FLASH)	—	—	128	22	28SP, 28SO, 28SS, 28ML	5	—	1	1-16 bit, 2-8 bit, 1-WDT	I <sup>2</sup> C/SPI	20	✓	✓	—	1	Self-read
PIC16F73	7168 (FLASH)	4096x14 (FLASH)	—	—	192	22	28SP, 28SO, 28SS, 28ML	5	—	2	1-16 bit, 2-8 bit, 1-WDT	USART/ I <sup>2</sup> C/SPI	20	✓	✓	—	2	Self-read

Taula 1: taula de selecció de la família flash de microcontroladors PIC.

### Característiques internes generals

En termes generals podem considerar que els PIC's són microcontroladors RISC convencionals amb un bus d'adreces i un bus de dades. El bus de dades va interconnectat a tots els ports d'entrada, sortida i als diferents perifèrics de que disposi. La freqüència de treball del microcontrolador serà la freqüència d'oscil.lacio dividida entre quatre, totes les instruccions s'executen en un sol cicle excepte els salts o branques de programa que triguen dos cicles. Disposen de la seva memòria RAM i la seva ROM que en aquest cas és flash, a nivell de blocs conté un comptador de programa que carrega les instruccions de ROM, una unitat aritmètica lògica i un acumulador o registre temporal que en aquest cas s'anomena W. La configuració de la CPU s'emmagatzema en el registre de STATUS. Per altra banda es permet un adreçament directe sobre els registres a través de instruccions o un adreçament indirecte a través dels registres especials FSR i INDF. Si col.loquem un nombre a FSR en INDF queda carregat el valor que conté la

posició apuntada per FSR. A continuació hi ha dos diagrames de blocks, corresponents a el PIC16F84A i al PIC16F877. Ràpidament es pot observar que el nucli és el mateix, però que disposa de més perifèrics connectats al bus de dades i de més memòria.



*Figura 1: diagrama de blocs del PIC16F84A*

Com es pot observar aquesta família de microcontroladors també disposen de un Watch dog timer intern per verificar el correcte funcionament dels mateixos. Aquesta utilitat es pot desabilitar al programar el dispositiu, de fet al programar el dispositiu se'ns permet escollir diferents característiques del hardware. Típicament són el tipus de rellotge, el MCLR (reset hardware) intern o extern, si volem protegir el codi (no ens permetrà llegir el que hem grabat) i altres característiques en funció de mòduls interns del microcontrolador en qüestió. Si no es fan servir aquests mòduls o les especificacions en qüestió és aconsellable desconectar tots els bits de configuració dels mateixos en la grabació, així evitarem problemes.

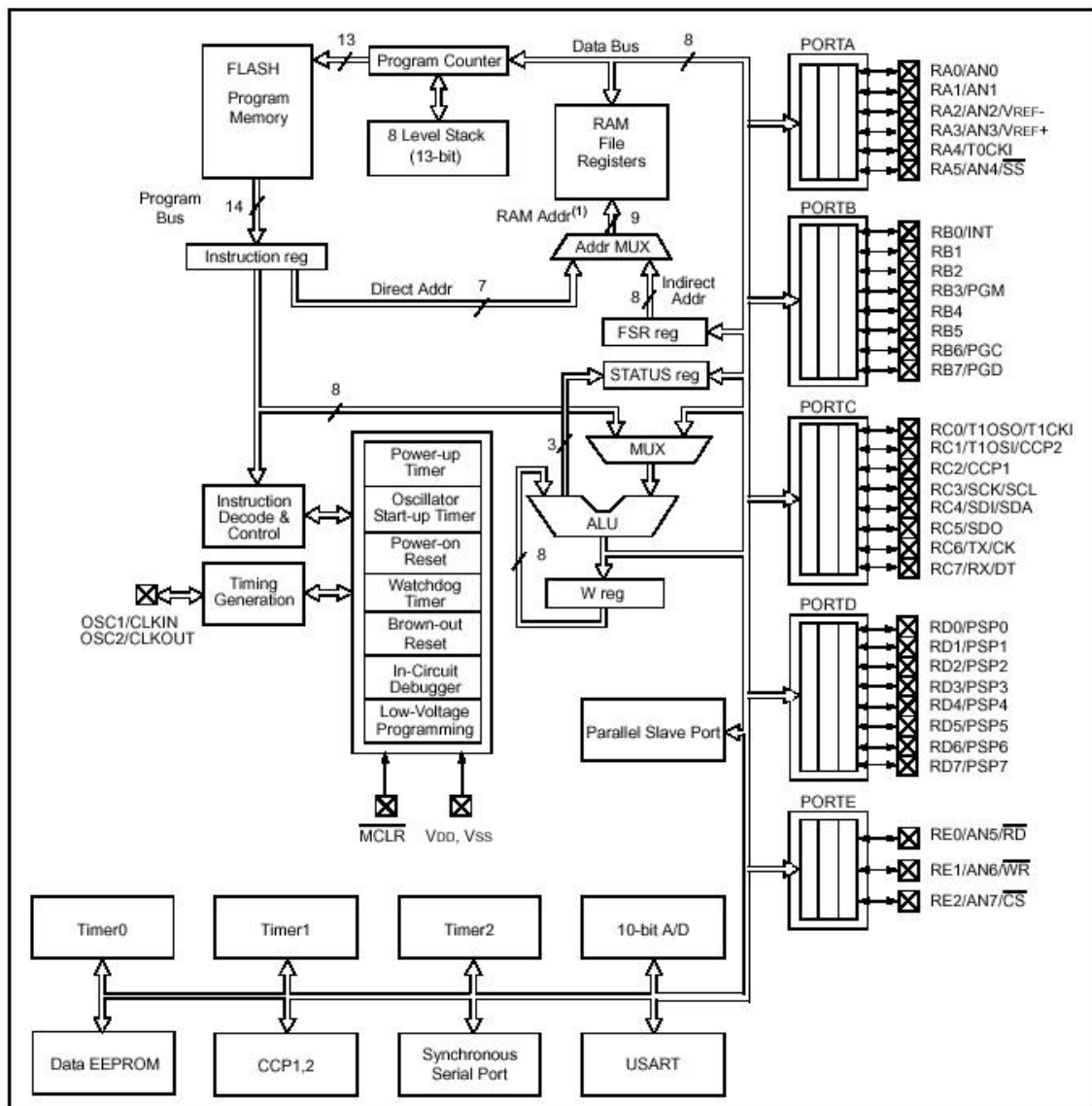


Figura 2: diagrama de blocs del PIC16F874/877

## Organització de la memòria

Els PIC's tenen una organització de la memòria molt concreta; com que el comptador de programa és de 13 bits, per tal de poder direccionar tota la memòria, aquesta s'ha dividit en bancs. Per canviar de banc s'han de modificar dos bits del registre d'estatus (RP0 i RP1). En pic's com el PIC16F84A com que disposa de poca memòria només hi ha un banc de programa i, en el cas d'intentar accedir al segon banc, ens trobarem amb que les posicions de memòria seràn les mateixes que en el primer banc. Normalment la memòria RAM es troba al final de cada banc tot i que cal comprovar-ho segons les característiques de cada microcontrolador per evitar problemes. La memòria eeprom interna de la que disposen la majoria de microcontroladors per guardar dades no volàtils té unes característiques d'accés molt concretes que es discutiran més endavant.

## El registre de STATUS

### Generalitats

El registre de STATUS conté l'estat aritmètic de la ALU, l'estat del RESET i la selecció de banc. El registre de STATUS pot ser la destinació de qualsevol operació, igual que qualsevol altre registre, però els bits de control Z, DC i C no es poden escriure, prendran el seu valor en funció de la última instrucció. Per altra banda els bits TO i PD no es poden escriure.

### REGISTRE STATUS

IRP	RP1	RP0	$\overline{TO}$	$\overline{PD}$	Z	DC	C
bit 7					bit 0		

**IRP:** Registre de selecció de banc indirecta (no utilitzat en 16F84A)

- 1 – Banc 2,3
- 0 – Banc 0,1

**RP1:RP0:** Selecció de banc (en 16F84A només RP0)

- 11 – Banc 3
- 10 – Banc 2
- 01 – Banc 1
- 00 – Banc 0

**$\overline{TO}$ :** bit de time out

- 1 – Després d'engegar, de CLRWDT o de SLEEP.
- 0 – Ha sobrepassat el WDT.

**$\overline{PD}$ :** bit d'apagat

- 1 – Després d'engegar o de CLRWDT.
- 0 – En l'execució de SLEEP.

**Z:** bit de zero.

- 1 – La operació anterior ha sigut zero.
- 0 – La operació anterior no ha sigut zero.

**DC:** bit de carry o borrow negat de digit.

- 1 – Hi ha hagut un carry en el quart bit.
- 0 – No hi ha hagut un carry en el quart bit.

En el cas del borrow la polaritat està invertida

**C:** bit de carry o borrow negat.

- 1 – Hi ha hagut un carry en el bit de més pes.
- 0 – No hi ha hagut un carry en el bit de més pes.

En el cas del borrow la polaritat està invertida.

## El registre OPTION\_REG

### Generalitats

El registre OPTION\_REG és un registre que es pot llegir i escriure que conté alguns registres de control per configurar el timer0 el WDT, el preescaler la interrupció externa INT, la del timer0 i els pull-ups del portB. El preescaler és un comptador programable que ens permet que el WDT o el timer0 no incrementin en cada operació si no en un nombre d'operacions programat pel preescaler, així aconseguim duracions majors de temps entre interrupció i interrupció del timer0 o entre que sobrepasa el WDT.

### REGISTRE OPTION\_REG

$\overline{\text{RBPU}}$	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0
--------------------------	--------	------	------	-----	-----	-----	-----

bit 7

bit 0

**$\overline{\text{RBPU}}$** : Habilitació dels pull-ups del port B.

- 1 – Els pull-ups estàn deshabilitats.
- 0 – Els pull-ups estàn activats.

**INTEDG**: Selecció de flanc de la interrupció.

- 1 – Interrupció per flanc de pujada de RB0/INT.
- 0 – Interrupció per flanc de baixada de RB0/INT.

**T0CS**: Selecció de rellotge del timer 0.

- 1 – Rellotge al pin RA4/T0CKI.
- 0 – Rellotge intern (rellotge instruccions).

**T0SE**: Selecció de flanc en l'increment del timer.

- 1 – Increment en un flanc de baixada de RA4/T0CKI.
- 0 – Increment en un flanc de pujada de RA4/T0CKI.

**PSA**: Assignació del preescaler.

- 1 – Preescaler assignat al WDT.
- 0 – Preescaler assignat al Timer0.

**PS2:PS0**: Selecció preescaler.

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

## El registre INTCON

### Generalitats

El registre INTCON és un registre de lectura/escriptura que conté els diferents valors per habilitar o deshabilitar les interrupcions del TMR0, la interrupció de RB0/INT i interrupcions globals. En el cas del PIC16F84A no hi ha cap més registre de configuració de interrupcions, però en funció del nombre de perifèrics de que disposi el pic s'afegiran registres de control d'interrupcions, normalment PIE1 i PIE2 amb els corresponents registres de configuració.

### REGISTRE OPTION\_REG

GIE	PEIE	TOIE	INTE	RBIE	TOIF	INTF	RBIF
-----	------	------	------	------	------	------	------

bit 7

bit 0

**GIE:** Habilitació global d'interrupcions.

- 1 – Habilita totes les interrupcions no emmascarades.
- 0 – Deshabilita totes les interrupcions.

**PEIE:** Habilitació de les interrupcions dels perifèrics.

- 1 – Habilita totes les interrupcions no emmascarades dels perifèrics.
- 0 – Deshabilita les interrupcions dels perifèrics.

**TOIE:** Habilitació de la interrupció per sobrepassament de TMR0.

- 1 – Habilita la interrupció del TMR0.
- 0 – Deshabilita la interrupció del TMR0.

**INTE:** Habilitació de la interrupció externa RB0/INT.

- 1 – Habilita la interrupció externa de RB0/INT.
- 0 – Deshabilita la interrupció externa de RB0/INT.

**RBIE:** Habilitació de la interrupció per canvi en el port B.

- 1 – Habilita la interrupció per canvi en el port B.
- 0 – Deshabilita la interrupció per canvi en el port B.

**TOIF:** Flag de sobrepassament del TMR0.

- 1 – S'ha sobrepassat el TMR0 (s'ha de posar a 0 per software).
- 0 – El TMR0 no ha sobrepassat.

**INTF:** Flag de l'interrupció externa RB0/INT.

- 1 – Hi ha hagut una interrupció externa a RB0/INT (s'ha de posar a 0 per software).
- 0 – No hi ha hagut una interrupció externa.

**RBIF:** Flag de l'interrupció del port B.

- 1 – Almenys un dels 8 bits del port B ha canviat (s'ha de posar a 0 per software).
- 0 – No ha canviat cap bit del port B.

## Set de instruccions

ORDRE	FLAGS	DESCRIPCIÓ
OPERACIONS ORIENTADES A REGISTRES (BYTES)		
<b>ADDWF</b>	<b>f,d</b>	C,DC,Z Suma w amb f. <sup>2</sup>
<b>ANDWF</b>	<b>f,d</b>	Z Realitza la AND lògica entre w i f. <sup>1</sup>
<b>CLRF</b>	<b>f</b>	Z Esborra el contingut de f.
<b>CLRW</b>	-	Z Esborra el contingut de l'acumulador (w).
<b>COMF</b>	<b>f,d</b>	Z Complementa el registre f. <sup>1</sup>
<b>DECF</b>	<b>f,d</b>	Z Decrementa el contingut del registre f. <sup>1</sup>
<b>DECFSZ</b>	<b>f,d</b>	Decrementa el contingut del registre f i salta la següent instrucció si el resultat és zero. <sup>1</sup>
<b>INCF</b>	<b>f,d</b>	Z Incrementa el contingut del registre f. <sup>1</sup>
<b>INCFSZ</b>	<b>f,d</b>	Incrementa el contingut del registre f i salta la següent instrucció si el resultat és zero. <sup>1</sup>
<b>IORWF</b>	<b>f,d</b>	Z Realitza la O inclusiva entre l'acumulador (w) i el registre f. <sup>1</sup>
<b>MOVF</b>	<b>f,d</b>	Z Mou el contingut de f. <sup>1</sup>
<b>MOVWF</b>	<b>f</b>	Mou el contingut de l'acumulador al registre f.
<b>NOP</b>	-	No operació.
<b>RLF</b>	<b>f,d</b>	C Rotació cap a l'esquerra a través del carry del registre f. <sup>1</sup>
<b>RRF</b>	<b>f,d</b>	C Rotació cap a la dreta a través del carry del registre f. <sup>1</sup>
<b>SUBWF</b>	<b>f,d</b>	C,DC,Z Resta el contingut de l'acumulador (w) al registre f. <sup>1</sup>
<b>SWAPF</b>	<b>f,d</b>	Intercanvia els 4 bits de menys pes amb els quatre de més pes del registre f. <sup>1</sup>
<b>XORWF</b>	<b>f,d</b>	Z Realitza la O exclusiva entre l'acumulador (w) i el registre f. <sup>1</sup>

<sup>2</sup> Si d és 0 o w el resultat es deixa en el registre w, en canvi si és 1 o f el resultat es deixa en el mateix registre sobre el que es realitza el càlcul.

<b>ORDRE</b>	<b>FLAGS</b>	<b>DESCRIPCIÓ</b>
OPERACIONS ORIENTADES A BITS DE REGISTRES		
<b>BCF</b>	<b>f, b</b>	Posa a "0" el bit "b" del registre f .
<b>BSF</b>	<b>f, b</b>	Posa a "1" el bit "b" del registre f .
<b>BTFSC</b>	<b>f, b</b>	Comprova l'estat del bit "b" del registre f, si és zero salta la següent instrucció .
<b>BTFSS</b>	<b>f, b</b>	Comprova l'estat del bit "b" del registre f, si és un salta la següent instrucció .
OPERACIONS ORIENTADES A CONSTANTS I CONTROL		
<b>ADDLW</b>	<b>k</b>	C,DC,Z Suma "k" a l'acumulador (w) .
<b>ANDLW</b>	<b>k</b>	Z AND lògica entre l'acumulador (w) i "k" .
<b>CALL</b>	<b>k</b>	Crida a una subrutina.
<b>CLRWDT</b>	-	$\overline{TO}, \overline{PD}$ Esborra el watch dog timer.
<b>GOTO</b>	<b>k</b>	Salta a una adreça.
<b>IORLW</b>	<b>k</b>	Z O inclusiva entre k i l'acumulador (w).
<b>MOVLW</b>	<b>k</b>	Escriu a l'acumulador el valor "k".
<b>RETFIE</b>	-	Retorna d'una interrupció.
<b>RETLW</b>	<b>k</b>	Retorna amb el valor "k" a l'acumulador.
<b>RETURN</b>	-	Retorna d'una subrutina.
<b>SLEEP</b>	-	$\overline{TO}, \overline{PD}$ Entra en mode d'espera.
<b>SUBLW</b>	<b>k</b>	C,DC,Z Resta l'acumulador (w) a la constant "k" .
<b>XORLW</b>	<b>k</b>	Z O exclusiva entre k i l'acumulador (w).

## EEPROM INTERNA<sup>3</sup>

### Generalitats

Per a realitzar la lectura/escriptura de la memòria interna s'utilitzen quatre registres de dades; **EECON1**, **EECON2** (que no és un registre físic), **EEDATA** i **EEADR**. **EEDATA** s'utilitza per emmagatzemar els 8 bits de dades que s'han llegit o que es volen escriure. **EEADR** conté la direcció en la que s'accedeix de la eeprom. El PIC16F84A té 64 bytes de dades en l'eeprom amb adreces que van de 0h fins a 3fh. La eeprom permet lectura i escriptura. Al escriure una posició s'esborra automàticament la posició i s'escriu tot seguit. L'escriptura està controlada per un timer intern. El temps de programació variarà en funció del voltatge, la temperatura i les toleràncies existents entre els components. Encara que el circuit tingui el flag de protecció de codi activat, la CPU pot continuar llegint o escrivint en la memòria interna, però qualsevol altre dispositiu no hi podrà accedir. El contingut del registre **EECON1** és el següent;

### REGISTRE EECON

—	—	—	EEIF	WRERR	WREN	WR	RD	
bit 7								bit 0

**EEIF:** Flag d'interruptió d'escriptura.

- 1 – Escriptura completada (s'ha de canviar a 0 per software).
- 0 – No s'ha acabat la grabació o no ha començat.

**WRERR:** Flag d'error.

- 1 – S'ha acabat l'escriptura de forma sobtada.
- 0 – L'escriptura ha acabat correctament.

**WREN:** bit d'habilitació d'escriptura.

- 1 – Permet escriptura.
- 0 – No permet escriure en l'eeprom.

**WR:** bit de control d'escriptura.

- 1 – Inici de un cicle d'escriptura. Aquest bit es neteja (posa a 0) per hardware quan l'escriptura s'ha completat.
- 0 – S'ha acabat el cicle d'escriptura.

**RD:** bit de control de lectura.

- 1 – Inici de un cicle de lectura. Aquest bit es neteja per hardware.
- 0 – No inicia cap cicle de lectura.

### Llegir d'una posició de memòria

Per a llegir una posició de memòria l'usuari ha d'escriure la direcció en el registre **EEADR** i després habilitar el bit de control **RD** (**EECON1, 0**). La dada estarà disponible en el cicle següent en el registre **EEDATA**, per aquest motiu es pot llegir en

<sup>3</sup> Basat en el datasheet del component PIC16F84A de Microchip Technology Inc. ®

el següent cicle. **EEDATA** mantindrà el seu valor fins que hi hagi una altra lectura o es modifiqui el seu contingut per l'usuari.

Exemple de seqüència de lectura;

BCF	STATUS,RP0	; Canvi al bank 0
MOVLW	ADREÇA	; variable on tinc l'adreça a llegir
MOVWF	EEADR	; ho passo al registre de lectura
BSF	STATUS,RP0	; Canvi al bank 1
BSF	EECON,RD	; bit de lectura
BCF	STATUS, RP0	; Canvi al bank 0
MOVF	EEDATA,W	; W=EEDATA

### Escriure una posició de memòria

Per a escriure una posició de la memòria el primer que s'ha de fer és escriure la adreça a escriure en el registre **EEADR** i la dada a escriure en el registre **EEDATA**. Després l'usuari ha de seguir, exactament, la següent seqüència;

BSF	STATUS,RP0	; Canvi al bank 1
BCF	INTCON,GIE	; No permet interrupcions
BSF	EECON1,WREN	; bit d'habilitació d'escriptura
MOVLW	55h	;
MOVWF	EECON2	; EECON2=55h
MOVLW	AAh	;
MOVWF	EECON2	; EECON2=AAh
BSF	EECON1,WR	; Comença l'escriptura
BSF	INTCON,GIE	; Permet interrupcions
BCF	STATUS,RP0	; Torna al bank 0

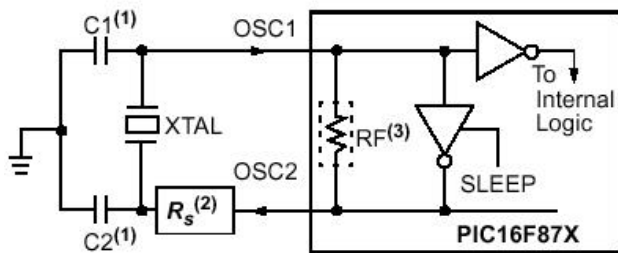
L'escriptura no començarà si no es segueix exactament la seqüència d'escriure primer **55h** a **EECON2**, després **AAh** en el mateix registre i, finalment, habilitar el bit d'escriptura **EECON1,WR**. Es recomana deshabilitar les interrupcions durant el procés.

### Verificació de l'escriptura

Depenent de l'aplicació seria convenient comprovar que la dada grabada a la eeprom és correcta, sobretot en aquelles aplicacions en que s'intenta grabar a una velocitat màxima. Una bona solució per poder recuperar dades mal escrites és la de triplicar el procés d'escriptura, sempre i quan no s'utilitzi més d'un terç de la memòria. Al llegir es comprovarà que almenys dos dels tres valors siguin iguals, així, en el cas que hi hagi alguna escriptura errònea no ens importarà. Cal tenir en compte que el **PIC16F84A** té 64 bytes de memòria EEPROM que van de la posició **0h** a la **3Fh**.

## Generalitats sobre el hardware extern

Tot i que en general els microcontroladors PIC són molt robustos cal tenir en compte alguns detalls. Un dels més importants és la tensió d'alimentació; si no es tracta d'una família especial, en general els PIC's funcionen correctament en un rang d'alimentacions de 4'5 V a 5'5 V, però cal tenir en compte que si utilitzem algun dels seus perifèrics, com per exemple els A/D's, el seu resultat es veurà modificat a causa de la diferència respecte la tensió d'alimentació nominal. **És molt important posar condensadors de desacoblament entre alimentació i massa el més pròxims al microcontrolador. De la mateixa manera que en cas de utilitzar un cristall s'ha de posar el més pròxim al microcontrolador i amb els condensadors (de 15 a 20 pF) a massa tal com s'indica en el datasheet;**



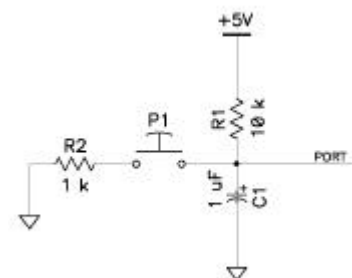
Mode	Freq.	OSC1	OSC2
XT	455 kHz	68 - 100 pF	68 - 100 pF
	2.0 MHz	15 - 68 pF	15 - 68 pF
	4.0 MHz	15 - 68 pF	15 - 68 pF
HS	8.0 MHz	10 - 68 pF	10 - 68 pF
	16.0 MHz	10 - 22 pF	10 - 22 pF

Un altre element molt important al dissenyar hardware per microcontroladors és el fet d'assegurar sempre un estat lògic en els ports, tant si es tracta d'entrada com de sortida, això s'aconsegueix utilitzant resistències de pull-up o de pull-down (en funció de l'estat assegurat) d'un valor elevat. Cal recordar que el port B disposa de resistències de pull-up internes que si no es desactiven poden donar lloc a divisors de tensió amb resistències externes.

Un punt important en el cas de tenir un pin de reset MCLR és evitar l'entrada de resets no desitjats degut a soroll. Normalment això s'aconsegueix mitjançant la utilització de pulsadors. Una altra forma de tenir un reset net d'interferències és la utilització de circuits especialment dissenyats que ens ofereix el fabricant, Microchip, en concret l'MCP100 és un circuit de tres terminals, alimentació, massa i reset que només ens provocarà un reset en la caiguda de l'alimentació. Un altre punt important pel que fa l'alimentació, en el cas que es vulguin conservar dades en eeprom és la utilització de un detector de caiguda de tensió. Normalment aquests detectors es dissenyen de forma que medeixen la tensió a l'entrada de la font d'alimentació (abans de ser regulada) i en la sortida, quan l'entrada baixa d'un cert valor, s'activa el comparador que monitoritza aquestes senyals i ens dóna una interrupció al PIC que ens permet guardar les dades a eeprom abans d'apagar-se.

Els pulsadors, normalment, es col·loquen amb una xarxa antirebot, habitualment amb l'esquema següent;

Es basa en la xarxa RC que es forma de manera que a descarregar-se tardarà un cert temps (1 ms) i a carregar-se també. Com que R1 és molt més gran que R2 tota la tensió cau a R1 pràcticament, així que al apretar el pulsador el microcontrolador veurà el zero després de 1 ms. Així evitem entrada de interferències. Per altra banda, però, una altra solució molt utilitzada i que ens evita la inclusió de hardware és la de realitzar un antirebot per software.



En el cas que ens interessi donar una ordre que ve per un polsador, aquest antirebot serà molt senzill (es suposa que almenys hi ha una resistència de pull-up a l'entrada del polzador);

```
polsador
    btfsc  _POLSADOR      ; Miro si s'ha apretat?
    goto  polsador       ; Status register per PUSH i PULL
polsadorr
    btfss  _POLSADOR      ; Miro si s'ha deixat anar
    goto  polsadorr      ; No continua fins que no s'ha tornat a deixar anar
```

Així evitem que ens entri molt ràpidament a una mateixa rutina quan només volem que hi entri una vegada. Si el problema és la entrada de interferències electromagnètiques també podem assegurar que el polsador ha estat apretat mirant l'estat varies vegades;

```
polsador
    btfsc  _POLSADOR      ; Miro si s'ha apretat?
    goto  polsador       ; Status register per PUSH i PULL
    nop
    nop
    btfsc  _POLSADOR
    goto  polsador
    nop
    nop
    btfsc  _POLSADOR
    goto  polsador
    nop
    nop
    nop
    btfsc  _POLSADOR
    goto  polsador

polsadorr
    btfss  _POLSADOR      ; Miro si s'ha deixat anar
    goto  polsadorr      ; No continua fins que no s'ha tornat a deixar anar
```

D'aquesta manera, en funció de les repeticions i els nop que tinguem el pic d'entrada haurà de tenir una durada més llarga per tal de que s'entengui que no és una simple interferència sinó que és una ordre desitjada.

## Generalitats sobre el compilador i la programació

### Organització del programa

El compilador, com la majoria, té una ordre que ens permet organitzar en quines posicions del microcontrolador volem grabar diferents parts de codi. En els microcontroladors PIC ens trobem amb les següents posicions importants;

0x000	-	Vector de reset
0x004	-	Vector de interrupció
0x20	-	Inici de la memòria ram del primer bloc

Així doncs en qualsevol programa trobarem el següent codi:

```
org 0x000
goto programa_principal
org 0x004
goto tractament_interrupció
```

Un altre punt molt important és la realització de un PUSH i un PULL a l'entrada i la sortida d'interrupció respectivament, ja que quan entrem en una interrupció podríem perdre el contingut dels registres per alguna altra operació i al tornar que el programa principal no realitzés adequadament les seves operacions.

### Exemple de programació, organització d'un programa

Inicialment sempre és aconsellable tenir una capçalera al programa;

```
;*****
;
;                               Control PIC
;                               by Albert Comerma
;                               Juny del 2002
;
;*****
;
; Nom: XXXX.ASM
; Tamany:
; RAM:
; Descripció: Regulador controlat per PIC.
;
;*****
;
; LIST P=PIC16F877
;
#include "P16F877.INC"
errorlevel -302
__CONFIG _CP_OFF & _WDT_OFF & _BODEN_OFF & _PWRTE_OFF & _XT_OSC &
_WRT_ENABLE_OFF & _LVP_OFF & _DEBUG_OFF & _CPD_OFF
```

Uns detalls sobre el codi anterior; primerament dir que el que hi ha després de ; el compilador ho interpreta com a un comentari al codi i que, per tant, no compila, és aconsellable comentar el codi, ja que al tractar-se d'ensamblador és difícil recordar que es fa després d'un temps de la programació.

La paraula LIST P=PICXXXX ens indica el PIC a utilitzar pel compilador.

La comanda `#include "PICXXX.INC"` indica al compilador que carregui la correspondència entre els diferents noms dels registres del pic i la seva localització física, d'aquesta manera ens podem referir directament al registre INTCON enlloc de a la seva posició física.

La paraula `errorlevel -xxx` ens indica un número d'error que no volem que ens ensenyi el compilador, en aquest cas es tracta de un warning cada vegada que realitzem un canvi de banc.

Finalment el `__CONFIG ...` indica al compilador els bits de configuració a carregar a l'hora de grabar el microcontrolador. Per saber el nom que reben és convenient revisar el codi que hi ha en el directori d'instal·lació de l'mplab, normalment a `C:\Archivos de programa\mplab\template\code\familiapic.asm` que conté aquestes directives importants.

```

;*****
;   DEFINICIO DE LA RAM
;*****

cblock 0x20    ; Comença el bloc de memòria RAM
FLAG          ; Variable de sistema
TEMP          ; Variable temporal
W_TEMP        ; Work register per PUSH i PULL
STATUS_TEMP   ; Status register per PUSH i PULL
PCLATH_TEMP   ; Program counter per PUSH i PULL
FSR_TEMP      ; FSR per PUSH i PULL
endc          ; Acaba el bloc de memòria RAM
    
```

En el codi anterior es defineixen els diferents noms de variables RAM del sistema, indicant que la memòria comença en la posició 0x20. En cas de superar els bytes de RAM s'hauria de posar un altre cblock indicant la posició de memòria RAM del segon banc.

```

;*****
;*****++--- PINOUT MICRO ---*****
;*****--- PORT A ---*****
#define _CTS          PORTA,0      ;OUT
#define _FC1         PORTA,1      ;IN
#define _FC2         PORTA,2      ;IN
#define _FC3         PORTA,3      ;IN
#define _FC4         PORTA,4      ;IN
#define _FC5         PORTA,5      ;IN
;*****--- PORT B ---*****
#define _EXP1         PORTB,0      ;OUT
#define _EXP2         PORTB,1      ;OUT
#define _EXP3         PORTB,2      ;OUT
#define _EXP4         PORTB,3      ;OUT
#define _EXP5         PORTB,4      ;OUT
#define _EXP6         PORTB,5      ;OUT
#define _EXP7         PORTB,6      ;OUT
#define _RTS         PORTB,7      ;IN
    
```

En el codi anterior es defineixen els noms dels pins físics de cada port i es comenta si són entrada o sortida, per tenir un recull d'aquestes entrades/sortides i un nom al que referir-se a la resta de programa sense importar la seva situació física. Això és molt útil sobretot per si en algun moment s'ha de canviar de posició alguna entrada/sortida, ja que només canviant aquesta definició i la que correspon a la configuració dels ports ja ens quedarà tot modificat.

```

;*****
;          INICI DEL RESET                                     *
;*****
          org      0x000          ; vector de reset
          goto     main          ;
          org      0x004          ; vector de interrupció
push_reg
          movwf    W_TEMP         ; PUSH dels registres
          swapf    STATUS,w       ; acumulador, STATUS,
          clrf     STATUS         ; PC i FSR
          movwf    STATUS_TEMP    ;
          movf     PCLATH,w       ;
          movwf    PCLATH_TEMP   ;
          clrf     PCLATH        ;
          bcf     STATUS,IRP      ;
          movf     FSR,w         ;
          movwf    FSR_TEMP      ; Fi de PUSH
          goto     Service       ; salt a servei interrupció
    
```

En aquest fragment de codi hi ha la l'inici del programa, quan es vé de RESET salta a la posició 0x000 on, tot seguit salta a main que serà on hi ha el programa principal. En canvi si entra una interrupció anirà a la posició 0x004 on es troba una rutina molt útil de PUSH per guardar els registres importants i després va a Service que serà la rutina de tractament de les interrupcions, al final d'aquesta rutina i haurà el PULL corresponent per restaurar l'estat dels valors dels registres just abans de tornar.

```

          org      0x010          ; Inici del programa
main
          define_ports
          clrf     INTCON
          bsf     STATUS,RP0      ; Selecciona el bank1
          bcf     STATUS,RP1      ; Selecciona el bank1
          clrf     PIE1
          clrf     PIE2
          bsf     OPTION_REG,7    ; desconecto pull-ups port B
          movlw   0x3E            ;
          movwf   TRISA          ; Config Port A
          movlw   0x80            ;
          movwf   TRISB          ; Config Port B
          movlw   0xCF
          movwf   TRISC
          movlw   0x00
          movwf   TRISD
          movlw   0x07
          movwf   TRISE
          bcf     STATUS,RP0      ; Selecciona el bank0
    
```

Tot seguit ens trobem amb un org a una posició de programa on començarà precisament el programa principal, cal tenir en compte l'espai que ocupa la part escrita anteriorment a partir de l'org 0x004 per no escriure a sobre ara amb el programa principal. El primer que es fa en aquest programa és la configuració dels ports com a entrades o sortides. En aquest punt també s'acostumen a configurar els diferents perifèrics. Aquest codi està realitzat per al 16F877 observem que configura fins al port E. Aquest port, l'E només té tres bits configurables com a entrada/sortida és molt important deixar els altres bits a zero, ja que si no activarem la recepció paral.lel i el programa no funcionarà com desitgem. Cada port es configura posant el valor de "0" si és sortida i "1" si és entrada en el registre de configuració corresponent a cada port TRISX (aquests registres, així com l'OPTION\_REG es troben en el banc 1. En aquest codi també s'observa com es canvia el banc de treball i com es deshabiliten els pull-ups del portB. Abans de tot això, però, es desactiven totes les interrupcions esborrant el registre INTCON.

```
clear_RAM          ; Esborra la RAM
  movlw 0x20        ; desde l'adreça      0x20
  movwf FSR         ; fins a la         0x7F
next_reg          ;
  clrf INDF        ;
  incf FSR,f       ;
  movlw 0x80       ;
  xorwf FSR,w      ;
  btfss STATUS,Z   ;
  goto  next_reg   ;
```

Aquesta rutina que acostuma a venir després de la configuració i abans del programa propiament dit és molt útil per assegurar que la memòria RAM està esborrada, ja que no es pot assegurar al venir de RESET. Es basa en l'adreçament indirecte utilitzant els registres FSR i INDF.

```
*****
;*          PROGRAMA          *
*****
  clrf FLAG
  bsf  INTCON,GIE  ; permeto interrupcions
  movlw 0x7F
  movwf FSR        ; posiciono FSR a últim lloc ram
inici
  btfsc FLAG,1    ; flag fi trama rebuda?
  ...
  ...
  goto inici
```

Tot seguit vindria el programa a realitzar, en aquest no es fa res, només s'inicialitzen algunes variables a valors concrets o es borren, s'engeguen les interrupcions i es va al bucle principal on es queda el programa. El programa només sortirà d'aquest bucle si hi ha una interrupció, en aquest cas aniria a la rutina que hem vist abans de PUSH i a continuació a la que ve;

```
*****
;          RUTINA D'INTERRUPCIO          *
*****
Service
  btfsc INTCON,T0IF ; mira si és la interrupció del timer0
  goto  timer       ; si? Va al tractament
  btfsc INTCON,INTF ; mira si és la interrupció de RB0
  goto  interrupció ; si? Va al tractament
fi_interrupt
  movf PCLATH_TEMP,w ; POP dels registres
  movwf PCLATH       ;
  movf FSR_TEMP,w   ;
  movwf FSR          ;
  swapf STATUS_TEMP,w ;
  swapf W_TEMP,f    ;
  swapf W_TEMP,w    ; Fi de POP
  bsf  INTCON,GIE   ; permet interruptions
  retfie           ;
```

Aquí es veu que comprova alguna de les interrupcions i si no hi ha hagut cap d'aquestes retorna. Normalment a les rutines de tractament de la interrupció el primer que trobarem és la posada a 0 del bit de la interrupció (ja sigui INTCON,T0IF o INTCON,INTF) i després la deshabilitació de interrupcions per poder treballar sense perill d'entrar un altre cop a interrupcions. Al final, hi haurà un salt incondicional a fi\_interrupt que ens realitza el PULL dels registres.

```

;*****
;*                               NOTES PEL COMPILADOR                               *
;*****

fi
end ; Final del Programa
    
```

Finalment, hi hauria les notes per al compilador, en aquest cas només l'end que indica al compilador que s'ha acabat el programa. Si en algun punt del programa fessim un salt a l'etiqueta fi, el que passaria és que el microcontrolador tornaria a començar el programa des del principi, configurant-ho tot de nou (cosa poc aconsellable).

### Algunes rutines d'interès

Abans hem vist una rutina molt útil per esborrar el bloc de memòria RAM. Tot seguit hi ha una rutina per realitzar un retard, molt útil per a programes on és necessària una intervenció humana, ja que hem de tenir en compte que la freqüència de treball d'un microcontrolador és molt alta i que això pot portar a que l'operari no s'adoni del que està realitzant, al introduir aquesta rutina, per exemple podem observar com s'encén un led si la cridem després d'engegarlo i abans de pararlo. Aquesta rutina s'ha de cridar amb un call i utilitza dos registres de RAM.

```

retard
    movlw    0xFF
    movwf    COUNT2
    clrf     COUNT256
retard1
    decfsz   COUNT256,f
    goto    retard1
    decfsz   COUNT2,f
    goto    retard1
    return
    
```

Tot seguit hi ha un parell de rutines per enviar i rebre a través del port serie i la USART interna, cal tenir en compte que per connectar aquestes sortides a un PC caldria un adaptador de nivells tipus MAX232.

```

uxmt ; rutina envia un byte
    btfss   PIR1,TXIF
    goto    uxmt
    movf    XMTUSE,w
    movwf   TXREG
    return

urec ; rutina recepció dos bytes uart interna
    ; netejo flag rebuda
    clrf    RECFLG
    bsf     RECFLG,NEWBYT
    btfsc   RCSTA,FERR
    bsf     RECFLG,FRMERR1
    movf    RCREG,w
    movwf   RECUSE1
urecx
    btfsc   RCSTA,OERR
    bsf     RECFLG,OVERERR
    bcf     RCSTA,CREN
    bsf     RCSTA,CREN
    return
    
```

Abans de poder utilitzar aquestes rutines, però cal definir, primer els registres de RAM que utilitza, les constants i, després, la configuració correcta del hardware;

```

;*****
;      DEFINICIO DE CONSTANTS      *
;*****

NEWBYT equ    0
FRMEERR1 equ  3
OVERERR equ   6

...
...

uartconfig
    bsf     STATUS,RP0      ; Selecciona el bank1
    bcf     STATUS,RP1      ; Selecciona el bank1
    movlw   0x80             ; config uart 8 bits 1 stop i interrupts
    movwf   TXSTA
    movlw   0x19             ; config 2400 baud
    movwf   SPBRG           ; config baudrate
    bcf     STATUS,RP0      ; Selecciona el bank0
    clrf    RCSTA
    bsf     RCSTA,CREN       ; config uart rep continuament
    bsf     RCSTA,SPEN       ; engego uart
    bsf     STATUS,RP0      ; bank 1
    bsf     TXSTA,TXEN       ; habilito tx
    bsf     PIE1,RCIE       ; habilito interrupt rx
    bcf     STATUS,RP0      ; bank 0
    return
    
```

Per consultar exactament que realitzen els diferents registres utilitzats en la rutina anterior, consultar els datasheets del component.

Tot seguit hi ha una rutina de *bit banging* per llegir del port serie, és important que el microcontrolador treballi a 4 MHz per tenir el timing adequat i, en aquest cas, es basa en la utilització de la interrupció externa RB0/INT en el PIC16F84A. El baudrate configurat és de 9600 bauds. Aquesta rutina ja és més complexa i requereix de més registres RAM. Al tornar de la interrupció corresponent ens trobarem amb que el byte rebut estarà guardat en el registre RECREG. El baudrate es controla a partir dels valors que escribim en els registres acount i bcount que s'utilitzen per generar el timing adequat.

```

asyrec1 btfss  INTCON,INTF
        goto   asyrecx
        movwf  tempw
        swapf  STATUS,w
        bcf    STATUS,RP0
        movwf  tempst
        movlw  09
        movwf  bitctr
        call   half
        btfss  rdport,rec
        goto   asyrec2
        bsf    comflag,0
        goto   asyrec5
asyrec2 bsf    comflag,2
asyrec3 call   full
        decfsz bitctr,f
        goto   asyrec4
        btfss  rdport,rec
        bsf    comflag,0
        bcf    INTCON,INTE
        call   full
        goto   asyrec5
asyrec4 bsf    STATUS,C
        btfss  rdport,rec
        bcf    STATUS,C
        rrf    recreg,f
    
```

```
asyrec5 bcf      INTCON,INTF
        swapf   tempst,w
        movwf  STATUS
        swapf   tempw,f
        swapf   tempw,w
asyrecx
        retlw  0

vdly    movf    bcount,w
        movwf  cntrb
vdly0   movf    acount,w
        movwf  cntra
vdly1   decfsz  cntra,f
        goto   vdly1
        decfsz  cntrb,f
        goto   vdly0
        retlw  0

half    movlw   d'2'
        movwf  bcount
        movlw  d'13'
        movwf  acount
        call   vdly
        retlw  0

full    movlw   d'2'
        movwf  bcount
        movlw  d'29'
        movwf  acount
        call   vdly
        retlw  0
```