Lezione 1

Nella prima lezione del corso, cercheremo di comprendere come sia fatto internamente il controller più conosciuto di Microchip, ovvero il PIC16C84 poi evoluto nel PIC16F84.
Sebbene i microcontroller della Microchip partano tutti da una stessa idea progettuale, troviamo alcune differenze tra i vari chip della stessa famiglia e, quindi dovremo soffermarci attentamente su quali siano le peculiarità del PIC16C84:

Caratteristiche principali del PIC16C84
- Architettura RISC con 35 istruzioni
- Quasi tutte le istruzioni vengono eseguite in un solo ciclo
- Velocità di 400nS per istruzione
- 1K EEPROM interna per il programma
- 36 x 8 registri interni general pourpose (SRAM)
- 15 registri special function
- 64 registri ad 8 bit EEPROM per i dati
- 8 livelli di stack
- Indirizzamento diretto, indiretto, relativo
- Quattro sorgenti di interrupt:
    - Pin INT esterno
    - Overflow del TIMER0
    - Cambio di stato sulla porta B <4..7>
    - Fine ciclo di scrittura in EEPROM dati
- 1.000.000 di cancellazioni/scritture nella EEPROM memoria dati
- Mantenimento in assenza di tensione > 40 anni
- 13 I/O pin con controllo individuale di direzione
- 20 mA di corrente erogabile per pin
- 25 mA di corrente come sink
- Contatore interno 8 bit + prescaler
- Power-on reset
- Power-up timer
- Watchdog timer
- Protezione del programma in lettura
- Funzionamento in SLEEP
- Oscillatore selezionabile tra 4
- Programmazione seriale in-circuit
- Tensione di alimentazione da 2 a 6 volt
- Assorbimento a 5V e 4MHz < 2mA
- Assorbimento in SLEEP < 1 uA

L'hardware
In figura 1 vediamo il diagramma a blocchi del PIC16C84. La prima cosa che notiamo è la separazione del bus dati dal bus indirizzi, che velocizza notevolmente le operazioni.

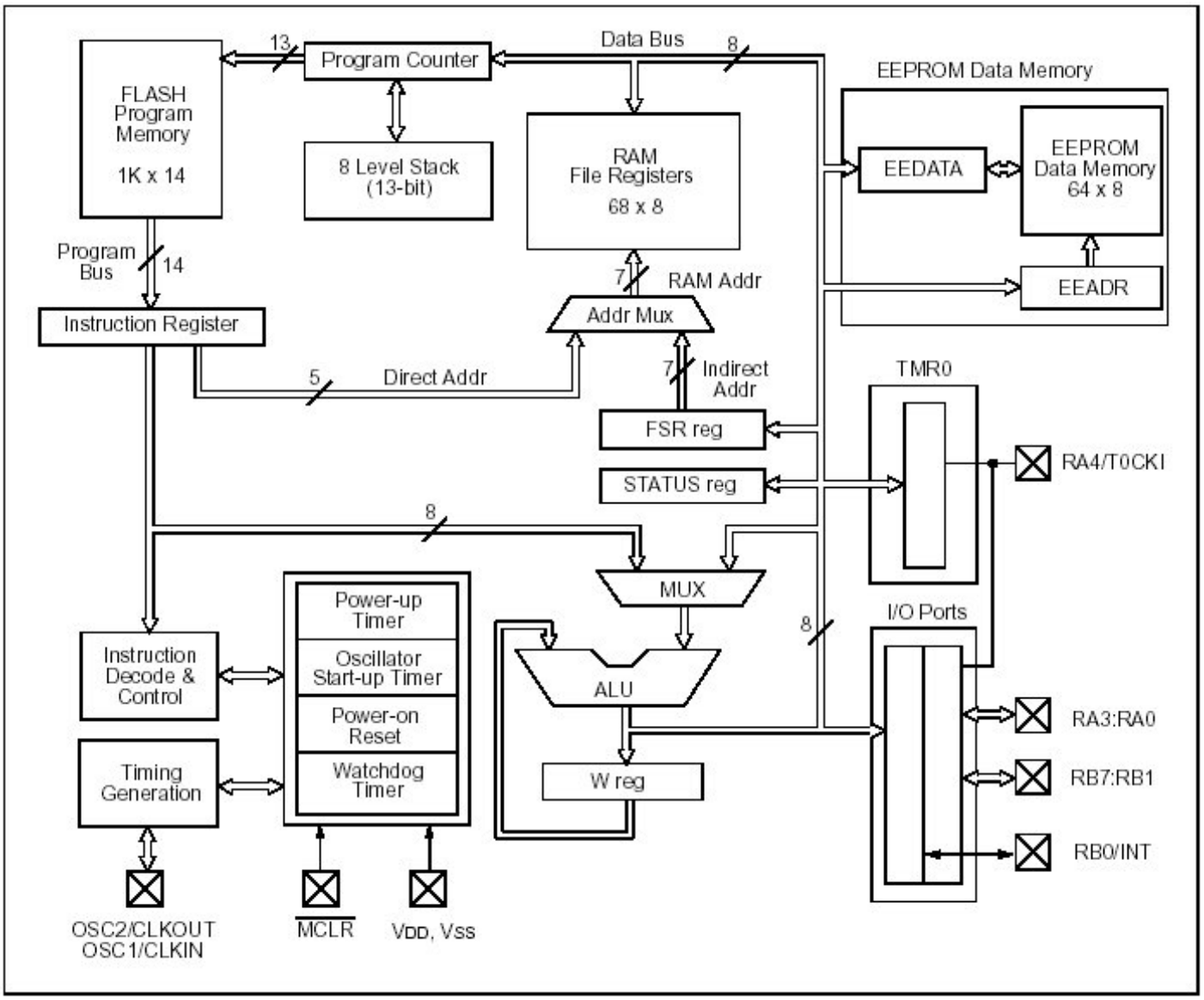


Figura 1: Diagramma a blocchi del PIC16C84

In alto a sinistra vediamo la Program Memory, ovvero la memoria dove verrà immagazzinato il programma vero e proprio. L'operazione da eseguire viene indirizzata dal Program Counter, che ha a disposizione uno stack di 8 livelli. Per coloro che sono a digiuno di controller, diciamo che, durante lo svolgimento di un programma, è possibile richiamare alcune subroutine di uso generale. Con uno stack a otto livelli, è possibile richiamare fino ad otto subroutine annidate.
Sul bus dei dati troviamo la Static RAM detta anche File Register. Questa memoria è di tipo RAM ed è possibile leggere o scrivere qualunque dei suoi registri, che possono essere di uso generale oppure dedicati a particolari funzioni che vedremo nel seguito. Sempre sul bus dati vediamo connesso il FSR (File Select Register), che potrà servire per indirizzare una cella della SRAM con modalità diversa da quella diretta.

Altro registro impiegato spesso che troviamo sul bus dati è lo STATUS Register, ovvero il registro di stato. Tramite questo registro è possibile sapere l'esito di una istruzione immediatamente dopo che è stata eseguita.
Ma il registro più sfruttato di tutti è senza dubbio il Working Register, detto anche "W" oppure registro di lavoro. Noterete che nella programmazione, su dieci linee di programma, questo registro viene chiamato almeno tre o quattro volte, poichè è l'unico registro che consente il passaggio dei dati da un registro all'altro.
Sempre sul bus dati vediamo anche la ALU, ovvero l'Unità Logico Aritmetica che si occupa di gestire tutte le istruzioni matematiche.
Come periferiche, abbiamo i 64 byte di EEPROM, gestiti dai due registri EEDATA e EEADR, il timer RTCC (Real Time Clock Counter) e i registri delle porte vere e proprie. Infatti vedremo che una porta viene trattata dal controller come un qualsiasi registro, quindi, tanto per anticipare un esempio, per far accendere un led connesso alla porta A sul pin RA0 (bit 0 del registro PORTA A) sarà sufficiente scrivere un 1 su tale registro.
In figura 2 possiamo vedere sia la mappa dei registri SRAM, sia l'intera mappatura della memoria di programma, compresi gli otto livelli dello stack.

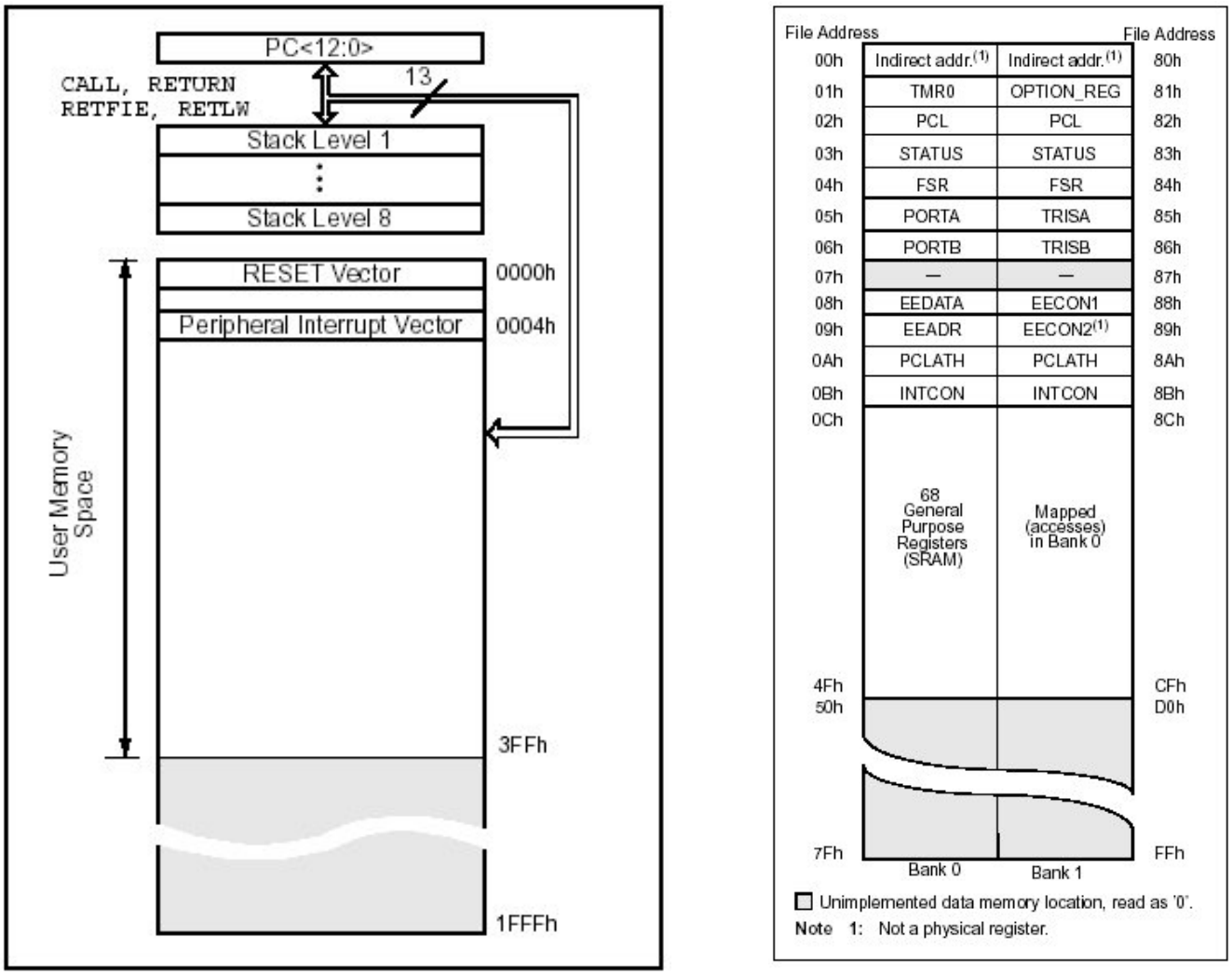
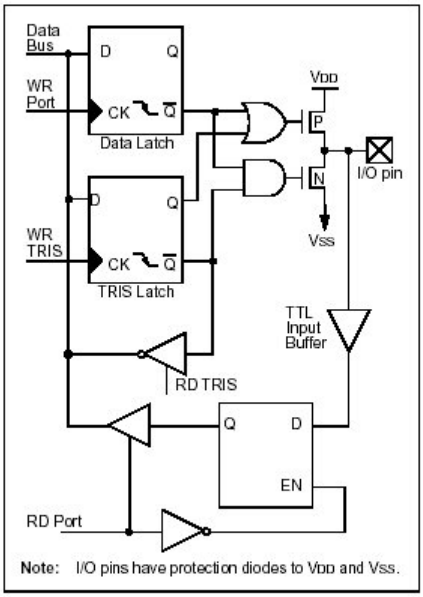
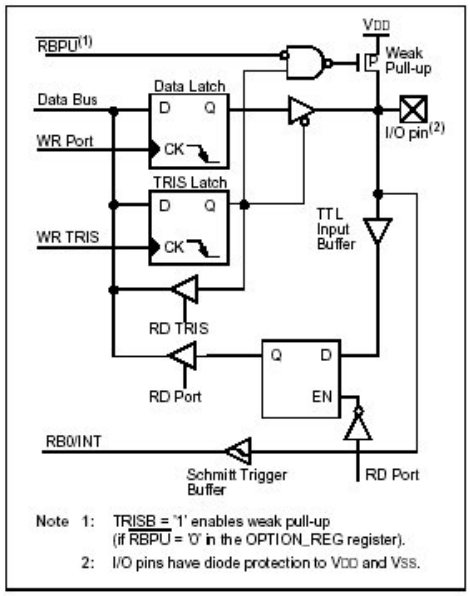


Figura 2: Mappa della SRAM e della memoria di programma

I registri SRAM, sono suddivisi in due banchi, banco 0 e banco 1, quindi per poter effettuare operazioni su di essi, dovremo prima settare correttamente il bit che seleziona l'uno o l'altro di questi due banchi.
A che cosa servono i registri dedicati lo vedremo in particolare nelle varie puntate, prendendo in esame i problemi pratici che possono sorgere in funzione di una errata programmazione.
I pin di connessione
Le due porte disponibili sul chip, hanno tre diversi interfacciamenti con i pin di I/O, a seconda delle funzioni che potrebbero svolgere.
In figura 3-a, possiamo vedere il più semplice, e cioè quello relativo alla porta A (RA0..RA3). Il flip-flop in alto a sinistra, serve per mantere in memoria il valore del bit da presentare sul pin, mentre il flip-flop sotto di lui serve per selezionare il pin in input (three-state) oppure in output. L'ultimo flip-flop esegue un latch sulla lettura del valore della tensione sul pin (ovviamente in digitale, cioè uno oppure zero).
In figura 3-b invece, troviamo il diagramma a blocchi della porta B dal pin RB4 al pin RB7. Una differenza con la porta A consiste nel poter abilitare una resistenza di pull-up interna, senza quindi necessità di componenti esterni. In più, e di maggior rilievo, abbiamo la generazione di un interrupt sul cambio di stato del valore di un pin. Tanto per fare un esempio, se prendiamo un pulsante e lo colleghiamo al pin RB7, configurando tale pin come input e abilitando la resistenza di pull-up, è possibile avere un interrupt (mascherabile e se abilitato) ogni volta che il pulsante connette il pin RB7 alla massa.
I pin da RB0 a RB7 della porta B invece, sono connessi in modo molto similare alla porta A, come è visibile in figura 3-c. La differenza più rilevante è la presenza delle resistenze di pull- up.





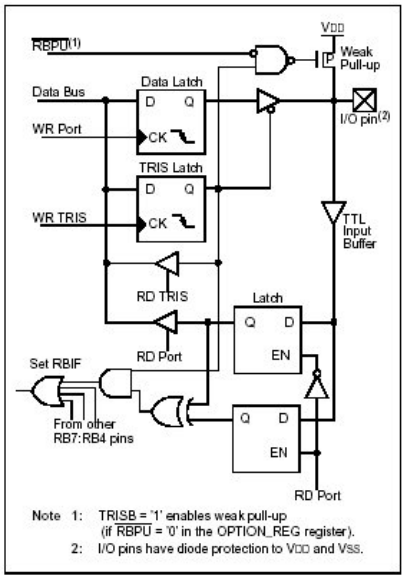


Figura 3-a. Diagramma a blocchi della porta A RA0..RA3

Figura 3-b. Diagramma a blocchi della porta A RB4..RB7

Figura 3-c. Diagramma a blocchi della porta A RB0..RB3

Il reset del chip

Una attenzione particolare merita la sezione di reset del PIC16C84 il cui diagramma a blocchi è visibile in figura 4. Ci sono diversi modi per resettare il PIC, di cui il primo è la forzatura a massa del pin MCLR (Master CLeaR). Quando questo pin viene connesso a massa, e per tutta la durata della connessione, il chip si trova in stato di reset.

Il secondo sistema per resettare il chip è l'impiego del watchdog. Poichè questo può essere abilitato o meno in fase di programmazione, il programmatore deve tenerne conto ed inserire l'istruzione CLRWDT prima dello scadere di circa 18mS se non vuole correre il rischio di trovarsi il chip resettato.

Altro reset viene generato dal riconoscimento della tensione di alimentazione (alla prima accensione), assicurandoci che il chip partirà sempre dallo stato che noi abbiamo previsto.

In aggiunta a questo reset, abbiamo la possibilità di abilitare un altro reset all'accensione, di durata maggiore rispetto al precedente. Questo meccanismo viene in genere impiegato quando il chip riceve l'alimentazione da un alimentatore collegato alla rete con un tempo di salita della tensione relativamente lungo.

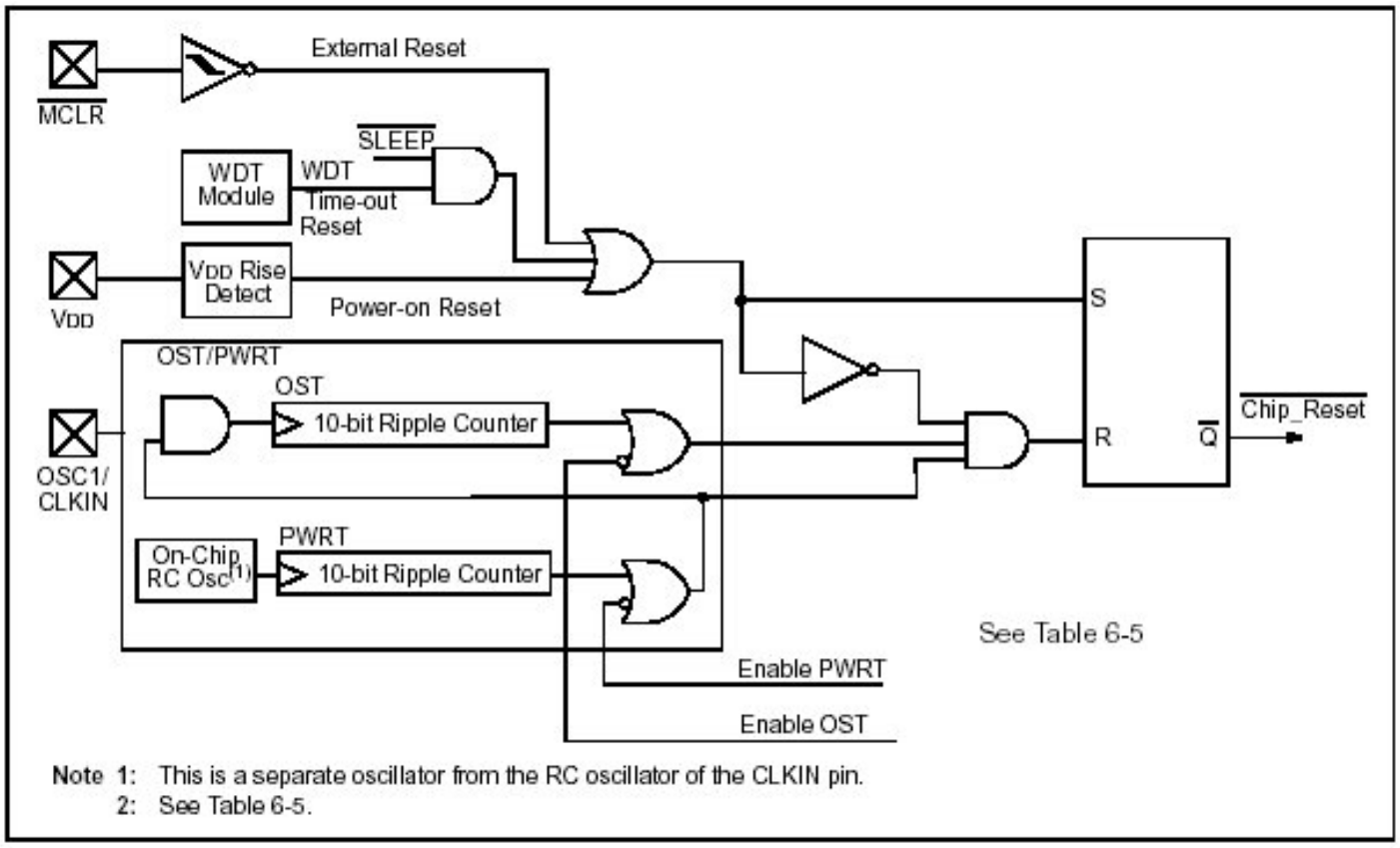


Figura 4. Diagramma a blocchi del reset del PIC16C84

Le istruzioni del PIC16C84

A questo punto ci sembra doveroso elencare le sole 37 (è un RISC!) istruzioni assembler del chip, comuni, tra l'altro, agli altri chip della famiglia PIC16CXX.

Nella tabella 1 le vediamo riassunte in tre gruppi principali: istruzioni orientate al byte, al bit e di controllo. Velocemente vediamo cosa fanno, accennando prima alle notazioni che useremo: da adesso, il registro di lavoro sarà chiamato semplicemente "w", i vari registri saranno detti "f", ovvero "File register" e le costanti saranno definite "l", ovvero "Literal" e "k" ovvero "Kostant". Il bit generico viene definito con "b".Il registro di destinazione invece viene indicato con "d", ovvero "Destination".

Mnemonic, Operands		Description	Cycles	14-Bit Opcode MSb LSb	Status Affected	Notes
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 lfff 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
DECF	f, d	Decrement f	1	00 0011 dfff ffff	Z	1,2
DECFSZ	f, d	Decrement f, Skip if 0	1 (2)	00 1011 dfff ffff		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		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 lfff ffff		
NOP	-	No Operation	1	00 0000 0xx0 0000		
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		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
BTFSC	f, b	Bit Test f, Skip if Clear	1 (2)	01 10bb bfff ffff		3
BTFSS	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	$\overline{TO}$,$\overline{PD}$	
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		
SLEEP	-	Go into standby mode	1	00 0000 0110 0011	$\overline{TO}$,$\overline{PD}$	
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	

Tabella 1. Sintesi delle istruzioni del PIC16C84

Vediamo allora le istruzioni.
ADDWF f,d: Somma il valore del registro W al valore del registro f e mette il risultato in W se d=0, in f se d=1 (default).
ANDWF f,d: Esegue l'operazione booleana AND tra il registro W ed il registro f e mette il risultato in W se d=0, in f se d=1 (default).
CLRF f: Pone a zero il valore del registro F.
CLRW: Pone a zero il valore del registro W.
COMF f,d: Complementa il valore del registro f (in pratica dove trova un "1" mette "0" e dove trova uno "0" mette un "1") e mette il risultato in W se d=0, in f se d=1 (default).
DECF f,d: Sottrae uno al registro f e mette il risultato in W se d=0, in f se d=1 (default).
DECFSZ f,d: Sottrae uno al registro f e, dopo, va a vedere se il valore del tale registro è divenuto zero. In caso affermativo, salta l'istruzione immediatamente successiva. Il risultato viene locato in W se d=0, in f se d=1 (default).
INCF f,d: Somma uno al registro f e mette il risultato in W se d=0, in f se d=1 (default).
INCFSZ f,d: Somma uno al registro f e, dopo, va a vedere se il valore del tale registro è divenuto zero. In caso affermativo, salta l'istruzione immediatamente successiva. Il risultato viene locato in W se d=0, in f se d=1 (default).
IORWF f,d: Esegue l'operazione booleana OR tra il registro W ed il registro f e mette il risultato in W se d=0, in f se d=1 (default).
MOVF f,d: Legge il valore del registro f e lo copia in se stesso se d=1 (usato raramente) oppure in W se d=0.
MOVWF f: Legge il valore di W e lo copia nel registro f.
NOP: Non esegue alcuna operazione.
RLF f,d: Esegue un'operazione di shift a sinistra del registro f. Al bit 0 viene assegnato il valore del Carry, mentre il carry prende il valore del bit 7.
RRF f,d: Esegue un'operazione di shift a destra del registro f. Al bit 7 viene assegnato il valore del Carry, mentre il carry prende il valore del bit 0.
SUBWF f,d: Sottrae il valore del registro W al valore del registro f e mette il risultato in W se d=0, in f se d=1 (default).
SWAPF f,d: Scambia i due nibble del registro f e mette il risultato in W se d=0, in f se d=1 (default).
XORWF f,d: Esegue l'operazione booleana EX-OR tra il registro W ed il registro f e mette il risultato in W se d=0, in f se d=1 (default).
BCF f,b: Pone a zero il valore del bit b del registro f
BSF f,b: Pone a uno il valore del bit b del registro f
BTFSC f,b: Testa il valore del bit b del registro f: se tale valore è "0", salta l'istruzione immediatamente successiva.
BTFSS f,b: Testa il valore del bit b del registro f: se tale valore è "1", salta l'istruzione immediatamente successiva.
ADDLW k: Somma la costante k al valore del registro W.
ANDLW k: Esegue l'operazione booleana AND tra la costante k ed il registro w:
CALL k: Effettua una chiamata alla subroutine k.
CLRWDT: Pone a zero il registro del watchdog.
GOTO k: Esegue un salto alla label k.
IORLW k: Esegue l'operazione booleana OR tra la costante k ed il registro w:

MOVLW k: Pone nel registro w il valore della costante k.
RETFIE: Consente il ritorno dopo un interrupt.
RETLW k: Consente il ritorno dopo una CALL copiando il valore della costante k nel registro W.
RETURN: Consente il ritorno dopo una CALL.
SLEEP: Pone il controller in modalità SLEEP
SUBLW k: Sottrae la costante k al valore del registro W.
XORLW k: Esegue l'operazione booleana EX-OR tra la costante k ed il registro w: OPTION: Copia il valore del registro w nel registro OPTION
TRIS f: Copia il valore del registro w nel registro di settaggio direzione delle porte A e B. f potrà valere 5 (porta A) o 6 (porta B) nel caso di un PIC a 18 pin, ma in altri f potrà assumere anche il valore 9 (porta E).

-continua