



ELETRONICA & MICROCONTROLLORI

<http://digilander.iol.it/bufere>

CORSO 8051

(C) Copyright 1997, 1998 by Vault Information Services. All Rights Reserved.

Le informazioni sono fornite senza alcuna garanzia.

Traduzione italiana di Sergio Salvitti.

Conversione in PDF di Farmi Roberto.

INDICE

[Introduzione](#)

- [Introduzione](#)

Capitolo 1 – [Tipi di Memoria](#)

- [Memoria Programma](#)
- [RAM Interna](#)
- [RAM Esterna](#)
- [Registri Speciali \(SFRs\)](#)
- [Memoria bit mapped](#)

Capitolo 2 – [Registri Speciali](#)

- [Cosa sono i registri SFR?](#)
- [Tipi di registri SFR](#)
- [SFR Standard](#)
- [SFR Non-Standard](#)

Chapter 3 – [Registri Base](#)

- [Accumulatore](#)
- [Registri "R"](#)
- [Registro B](#)
- [Data Pointer \(DPTR\)](#)
- [Program Counter \(PC\)](#)
- [Stack Pointer \(SP\)](#)

Capitolo 4 – [Modi d'indirizzamento](#)

- [Indirizzamento Immediato](#)
- [Indirizzamento Diretto](#)
- [Indirizzamento Indiretto](#)
- [Indirizzamento Diretto Esterno](#)
- [Indirizzamento Indiretto Esterno](#)

Capitolo 5 – [Controllo di flusso del programma](#)

- [Branch Condizionale](#)
- [Jump Diretti](#)

- [Call a subroutine Dirette](#)
- [Ritorno da Subroutine](#)
- [Interrupt](#)

Capitolo 6 – [Informazioni a basso livello](#)

- [Informazioni a basso livello](#)

Capitolo 7 – [I Timer](#)

- [Come conta un timer](#)
- [Misure temporali](#)
- [Quanto tempo impiega un timer a contare?](#)
- [Timer SFR](#)
 - ◆ [Registro TMOD](#)
 - ◇ [Modo 0 – Timer a 13-bit](#)
 - ◇ [Modo 1 – Timer a 16-bit](#)
 - ◇ [Modo 2 – Timer in auto mode](#)
 - ◇ [Modo 3 – Timer splittato](#)
 - ◆ [Registro TCON](#)
- [Inizializzazione di un timer](#)
- [Lettura dello stato un timer](#)
 - ◆ [Lettura del valore di un timer](#)
 - ◆ [Accorgersi dell'overflow di un timer](#)
- [Misura della durata di un evento](#)
- [Usare un timer come contatore di eventi](#)

Capitolo 8 – [Porta Seriale](#)

- [Impostazione del modo di funzionamento](#)
- [Impostazione del Baud Rate](#)
- [Scrivere sulla porta seriale](#)
- [Leggere dalla porta seriale](#)

Capitolo 9 – [Interrupt](#)

- [Eventi che scatenano un interrupt](#)
- [Impostare un interrupt](#)
- [Sequenza di Polling](#)
- [Priorita' degli Interrupt](#)
- [Che succede quando un interrupt viene attivato?](#)
- [Cosa avviene al termine di un interrupt?](#)
- [Interrupt Seriale](#)
- [Protezione del registro](#)
- [Problemi comuni nella gestione degli interrupt](#)

Riferimenti

- [Set d'Istruzioni dell'8051](#)

SUGGERIMENTI? COMMENTI? CORREZIONI?

Ogni sforzo e' stato profuso per rendere questo corso il piu' possibile: completo accurato e facile da leggere. Comunque, se avete qualche suggerimento, commento o segnalazione di qualche errore, per favore [mandate una mail per conoscenza](#).

(C) Copyright 1997, 1998 by Vault Information Services. All Rights Reserved.

Le informazioni sono fornite senza alcuna garanzia.

Traduzione italiana di Sergio Salvitti.

Conversione in PDF di Farmi Roberto.

Corso 8051

Introduzione

Nonostante la sua età, l'8051 è uno dei microcontrollori più popolari che vengono usati al giorno d'oggi. Molti microcontrollori derivati sono stati sviluppati per essere basati ed essere compatibili con l'8051.

Avere dimestichezza con la programmazione dell'8051 risulta perciò un importante bagaglio tecnico per coloro che vogliono sviluppare dei prodotti basati che usano dei microcontrollori.

Sono disponibili molte pagine web, libri e tool per chi sviluppa con l'8051.

Mi auguro che le informazioni contenute in questo documento web vi possano aiutare a produrre programmi per l'8051.

Sebbene non sia mia intenzione proporre questo documento come alternativa ad un eventuale volume comprato dal libraio sotto casa, esso lo potrebbe sostituire tranquillamente.

Questo documento contiene tutto quello che vi necessita per imparare il linguaggio assembler dell'8051.

Certamente, questo documento è gratuito e vi renderete conto di quanto spendereste se, dopo la lettura di esso, vi sentiste ancora persi e decideste di acquistare un libro.

Questo documento è al tempo stesso un corso e un tool di riferimento. I vari capitoli spiegano l'8051 passo passo.

I capitoli sono stati scritti per coloro che cercano di imparare il linguaggio assembler dell'8051.

Le appendici sono degli utili strumenti di riferimento sia per i principianti che per i programmatori professionali.

La lettura di questo documento presuppone che abbiate:

- Una conoscenza generale di come si programma.
- Una conoscenza sui sistemi di numerazione decimale, esadecimale e binaria.
- Una conoscenza generale sull'hardware.

Tanto per dire, non serve conoscere già l'8051, ma si presume che uno abbia già avuto: una precedente esperienza di programmazione, una conoscenza di base sull'hardware e una sicura padronanza dei tre sistemi di numerazioni sopra menzionati.

Non è certo scopo di questo documento insegnarvi la conversione da base decimale a esadecimale e se non avete già avuto esperienza in proposito, le conversioni di base rimarranno per voi dei concetti non completamente chiari.

Questo documento tenta di coprire le necessità di un *programmatore tipico* per esempio ci sono alcuni

peculiarita' che in rari casi sono molto utili e il 95% dei programmatori non le usera' mai. Per rendere questo documento maggiormente applicabile ad un pubblico piu' generale possibile, alcuni dettagli potrebbero essere soltanto accennati o completamente omessi.

In ogni caso, spero troviate il documento utile. Se avete domande, commenti o suggerimenti li accetto volentieri a: csteiner@vaultbbs.com.

Buona Programmazione!

[Craig Steiner \(Autore\)](#)

[Sergio Salvitti \(Traduttore\)](#)

^ [INDICE](#)

> [Tipi di Memoria](#)

(C) Copyright 1997, 1998 by Vault Information Services. All Rights Reserved.

Le informazioni sono fornite senza alcuna garanzia.

Traduzione italiana di Sergio Salvitti.

Conversione in PDF di Farmi Roberto.

Corso 8051

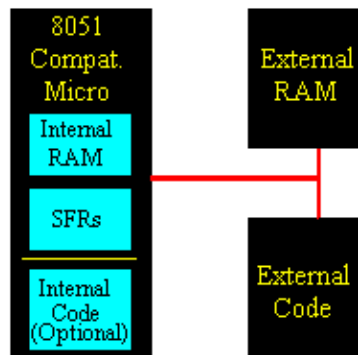
CAPITOLO 1 – Tipi di Memoria

L'8051 ha in generale tre tipi di memoria.

Per programmare efficacemente l'8051 e' necessario avere una conoscenza di base di questi tipi di memoria.

I tipi di memoria sono illustrati nel figura sottostante.

Essi sono: Memoria On–Chip, Memoria di programma esterna (External Code) e RAM esterna (External RAM).



La memoria On–Chip si riferisce a quella che fisicamente risiede nel componente. Essa puo' essere di tipi diversi, ma non preoccupatevi, la affronteremo presto nel seguito.

La memoria di programma esterna e' quella che contiene il codice e risiede fuori dal chip. Questa si presenta generalmente sotto forma di una EPROM esterna.

La RAM esterna e' la RAM che risiede fuori dal chip. Questa si presenta generalmente sotto forma di una RAM statica o di una FLASH RAM.

Memoria di Programma

La memoria di programma e' quella che contiene le istruzioni che l'8051 deve eseguire.

Questa memoria e' limitata a 64K e viene fornita in diverse forme e formati.

La memoria programma (codice) puo' essere contenuta all'interno del chip sotto forma di ROM oppure di EPROM.

Il codice puo' essere anche memorizzato in una ROM esterna o, piu' comunemente, in una EPROM.

La FLASH RAM e' un altro metodo molto usato per memorizzare un programma.

Possono essere usate diverse combinazioni di tipi di memoria, per esempio: e' possibile avere 4K di memoria di codice nella prom interna del chip e 64K di memoria di codice esterna in una EPROM.

Quando il programma e' memorizzato nella prom interna del chip i 64K massimi si riducono a 4k, 8k o 16k in funzione della versione di 8051 usata.

Ciascuna versione offre specifiche prestazioni. Un modo per distinguere i chip e' quello relativo alla quantita' di ROM/EPROM disponibile internamente.

Comunque, la memoria di programma e' comunemente implementata con una EPROM esterna. Questo e' specialmente vero nei sistemi di sviluppo a basso costo e nei sistemi sviluppati dai studenti.

Suggerimento per la programmazione: Poiche' la memoria programma e' limitata a 64k la dimensione dei programmi per l'8051 non dovrebbe superare tale limite. Alcuni assembler e compilatori offrono un accorgimento per ovviare a questa limitazione usando dei banchi di memoria. Comunque senza particolari compilatori e trucchi hardware i programmi sono limitati a 64K.

RAM esterna

Oltre alla *RAM interna*, l'8051 supporta anche la *RAM esterna*.

Come suggerisce anche il nome, la RAM esterna e' una memoria ad accesso casuale situata fuori dal chip. Essendo esterna, essa non e' flessibile in termini di accesso ed e' anche piu' lenta di quella interna. Per esempio, per incrementare una locazione di RAM interna e' richiesta una sola istruzione ed un solo ciclo d'istruzione, mentre per incrementare una variabile residente nella RAM esterna sono necessarie 4 istruzioni e 7 cicli d'istruzione. In questo caso la memoria esterna diventa 7 volte piu' lenta.

Quello che perde in velocita' lo guadagna pero' in ampiezza. Mentre la RAM interna e' limitata a 128 bytes (256 bytes nell'8052), quella esterna puo' arrivare fino a 64K.

Suggerimento per la programmazione: L'8051 puo' indirizzare soltanto 64k di RAM. Per espandere tale capacita' e' richiesto un trucco hardware. Potreste trovarvi nella necessita' di farlo a mano, poiche' molti assembler e compilatori forniscono il supporto per programmi superiori a 64K, ma non sono capaci di farlo anche con la RAM. Questo fatto e' molto strano. Nella mia passata esperienza ho notato che i programmi rientrano generalmente entro il limite di 64K mentre la RAM e' spesso insufficiente. Se necessitate quindi di un quantitativo RAM superiore a 64K, verificate che il vostro compilatore possa supportare la gestione della RAM a banchi altrimenti siate pronti a farlo a mano.

Memoria On-Chip

Come menzionato all'inizio di questo capitolo, l'8051 include internamente una certa quantita' di memoria. In effetti la memoria interna (On-Chip Memory) e' di due tipi: RAM interna e Area Registri Speciali (SFR memory).

Lo schema della memoria interna dell'8051 e' riportato nella mappa seguente:

IRAM Addr									Description
00	R0	R1	R2	R3	R4	R5	R6	R7	Reg. Bank 0
08	R0	R1	R2	R3	R4	R5	R6	R7	Reg. Bank 1
10	R0	R1	R2	R3	R4	R5	R6	R7	Reg. Bank 2
18	R0	R1	R2	R3	R4	R5	R6	R7	Reg. Bank 3
20	00	08	10	18	20	28	30	38	Bits 00-3F
28	40	48	50	58	60	68	70	78	Bits 40-7F
30	<div style="background-color: yellow; border: 2px solid black; padding: 5px; text-align: center;"> General User RAM & Stack Space (80 bytes, 30h-7Fh) </div>								General IRAM
7F									
80	<div style="background-color: cyan; border: 2px solid black; padding: 5px; text-align: center;"> Special Function Registers (SFRs) (80h - FFh) </div>								SFRs
:									
:									

Come illustrato nella mappa, l'8051 ha un banco di 128 bytes di RAM interna. Questo tipo di RAM e' la piu' veloce e la piu' flessibile in lettura e scrittura. La RAM interna pero' e' volatile, per cui, quando l'8051 viene resettato essa si cancella.

I 128 bytes di RAM interna sono divisi in porzioni come illustrato nella mappa di memoria.

I primi 8 bytes (00h-07h) sono il banco di registri 0. Cambiando il valore di alcuni registri SFR e' possibile scegliere di usare anche i banchi 1, 2, 3. Questi banchi di registri alternativi risiedono nella RAM interna agli indirizzi da 08h a 1Fh. Discuteremo dei "banchi di registri" con maggior dettaglio in un capitolo successivo. Per ora e' sufficiente sapere che essi esistono e fanno parte della RAM interna.

Anche la memoria a bit risiede nella RAM interna. Parleremo in maniera piu' approfondita della memoria a bit molto presto, ma per ora tenete soltanto a mente che essa risiede nella RAM interna a partire dall'indirizzo 20h fino all'indirizzo 7Fh.

I restanti 80 bytes della RAM interna, dall'indirizzo 30h a 7Fh, possono essere usati come variabili utente che richiedono accessi frequenti e veloci.

Questa area e' anche utilizzata dal microcontrollore come area di memoria per le operazioni di stack. Cio' limita moltissimo l'ampiezza dell'area di stack dell'8051 che e' al massimo di 80 bytes e anche meno. In realta' gli 80 byte sono usati anche dalle variabili d'utente.

Banchi di Registri

L'8051 usa i registri "R" che sono richiamati da molte delle sue istruzioni. Questi registri "R" sono numerati da 0 a 7 (R0, R1, R2, R3, R4, R5, R6, e R7). Essi sono generalmente usati per memorizzare un dato temporaneo o per muovere dati da una locazione di memoria ad un'altra.

Per esempio, per aggiungere il valore di R4 al valore dell' Accumulatore, noi eseguiremo la seguente istruzione:

ADD A,R4

Se l'Accumulatore (A) contiene il valore 6 e R4 il valore 3, al termine dell'istruzione suddetta l'accumulatore

conterra' il valore 9.

Comunque, come mostrato nella mappa di memoria, il registro R4 e' realmente parte della RAM interna, in particolare il suo indirizzo e' 04h. Questo puo' essere visto nella sezione verde della mappa di memoria. Anche l'istruzione seguente ottiene il medesimo risultato:

ADD A,04h

Questa istruzione addiziona il valore contenuto nella locazione di memoria d'indirizzo 04h al valore dell'accumulatore e carica il risultato nell' accumulatore stesso. Poiche' R4 risiede all'indirizzo 04h, l'istruzione summenzionata ottiene il medesimo risultato.

Ma osservate! Come mostra la mappa di memoria, l'8051 dispone di 4 distinti banchi di registri. All'accensione, l'8051 usa il banco 0 (indirizzi da 00h a 07h) come default. Ma il vostro programma puo' richiedere all'8051 di usare un banco di registri diverso da 0 (es. 1, 2, 3). In questo caso, R4 non si trovera' piu' all'indirizzo 04h. Per esempio, se il programma chiede all'8051 di selezionare il banco di registri numero 3, il registro R4 diventera' sinonimo di RAM interna alla locazione 1Ch.

Il concetto di banchi di registri aggiunge un livello di elevata flessibilita' all'8051, specialmente quando parliamo di interrupt (affronteremo l'argomento interrupt piu' avanti). In ogni caso, ricordate sempre che i banchi di registri risiedono nei primi 32 bytes della RAM interna.

Suggerimento per il programmatore: Se usate solo il primo banco di registri (p.e. il banco 0), le locazioni di RAM interna da 08h a 1Fh possono essere usate per i vostri scopi. Ma se pensate di usare anche i banchi di registri 1, 2, 3, state attenti a non accedere ad indirizzi di memoria al di sotto di 20h altrimenti andrete a sovrascrivere il valore di un registro R.

Memoria a bit

L'8051, e' un microcontrollore orientato alla trasmissione dati e , come tale, da all'utente la possibilita' di accedere ad un numero di *variabili a bit*. Queste variabili possono assumere o il valore 1 o il valore 0.

Ci sono 128 variabili a bit disponibili all'utente, numerate da 00h a 7Fh. L'utente puo' far uso di queste variabili con comandi quali SETB e CLR. Per esempio, per porre il bit numero 24(hex) al valore 1 dovrete eseguire la seguente istruzione:

SETB 24h

E' importante notare che la **Memoria a bit** e' realmente parte della RAM interna. Infatti, le 128 variabili a bit occupano 16 bytes della RAM interna da 20h a 2Fh. Comunque se scrivete il valore FFh nell'indirizzo 20h della RAM interna, in effetti avete settato tutti i bit da 00h a 07h.

Come dire:

MOV 20h,#0FFh

e' equivalente a:

SETB 00h

SETB 01h

SETB 02h

SETB 03h

SETB 04h
SETB 05h
SETB 06h
SETB 07h

Come appena illustrato, la memoria a bit non e' realmente un nuovo tipo di memoria ma solo una parte della RAM interna. Poiche' l'8051 fornisce delle istruzioni particolari per accedere a questi 16 bytes su base bit, e' utile ritenerla come se fosse una memoria separata. In ogni caso, ricordate che essa e' solo una parte della RAM interna e che in questa parte di RAM e' possibile effettuare delle operazioni bit a bit.

Suggerimento per il programmatore: Se il vostro programma non usa le variabili a bit, potete utilizzare le locazioni della RAM interna da 20h a 2Fh per i vostri scopi. Ma se pensate di utilizzare le variabili a bit, state molto attento agli indirizzi da 20h a 2Fh onde evitare di sovrascrivere il valore dei vostri bit!

Le variabili a bit da 00h a 7Fh sono funzioni utente definite nei tuoi programmi. Comunque le variabili a bit da 80h in su, sono attualmente utilizzate per accedere ad alcuni registri SFR su base bit. Per esempio, se le linee di uscita della porta 0 (P0.0 – P0.7) sono tutte a zero e volete settare solo il pin P0.0 dovrete eseguire la seguente istruzione:

MOV P0,#01h

oppure potreste eseguire quest'altra istruzione:

SETB 80h

Ambedue le istruzioni danno il medesimo risultato. In ogni caso, mentre il comando SETB attiva solo l'uscita P0.0 lasciando inalterato lo stato degli altri pin della porta P0, l'istruzione MOV, invece, forza tutti gli altri pin al valore 0, e in alcuni casi, questo puo' essere inaccettabile.

Suggerimento per il programmatore: Di default, l'8051 inizializza lo *Stack Pointer* (SP) al valore 08h quando il microcontrollore viene alimentato. Questo significa che lo stack iniziera' dall'indirizzo 08h e crescerà verso l'alto. Se volete usare i banchi di registri 1, 2, 3 dovete ricordarvi di inizializzare lo stack pointer sopra l'indirizzo del piu' alto banco di registri che volete adoperare, pena la sovrascrittura dei banchi di registri. Allo stesso modo, se pensate di usare la memoria a bit sarebbe una buona idea inizializzare lo stack pointer ad un valore piu' alto di 2Fh per garantire che le vostre variabili a bit siano protette dallo stack.

Registri Speciali (SFR)

I registri speciali (SFR) sono aree di memoria che controllano specifiche funzionalita' dell'8051. Per esempio, 4 registri SFR consentono l'accesso alle 32 linee di input/output dell'8051. Un altro registro SFR consente di leggere e scrivere sulla porta seriale dell'8051. Altri registri SFR permettono all'utente di: impostare il baud rate della seriale, controllare ed accedere ai timer e configurare il sistema degli interrupt dell'8051.

Quando state programmando, i registri SFR vi danno l'illusione di risiedere nella Memoria Interna. Per esempio se volete scrivere il valore "1" alla locazione 50 hex dovete eseguire la seguente istruzione:

MOV 50h,#01h

Allo stesso modo, se volete scrivere il valore "1" sulla porta seriale dell'805, dovete scrivere questo valore nel registro SFR denominato **SBUF** all' indirizzo e' 99 Hex.

Vedi come esempio la seguente istruzione:

MOV 99h,#01h

Come potete osservare, sembra che l'area SFR sia parte della Memoria Interna. Questo non e' vero! Quando utilizzate questo metodo di accesso alla memoria (indirizzamento diretto), ogni istruzione che usa degli indirizzi da 00h a 77Fh fa riferimento alla RAM interna, mentre se usa degli indirizzi da 80h a FFh fa riferimento ad un registro di controllo SFR.

Suggerimento per il programmatore: I registri SFR sono usati per controllare le funzionalita' dell'8051. Ogni registro SFR ha un sua funzione specifica ed un particolare formato del quale parleremo piu' avanti. Non tutti gli indirizzi al di sopra di 80h sono usati dai registri SFR. Comunque, questa area e' riservata e non puo' essere usata come RAM addzionale anche se ad un particolare indirizzo non corrisponde alcun registro SFR.

^ [INDICE](#)
< [Introduzione](#)
> [Registri Speciali](#)

(C) Copyright 1997, 1998 by Vault Information Services. All Rights Reserved.

Le informazioni sono fornite senza alcuna garanzia.

Traduzione italiana di Sergio Salvitti.

Conversione in PDF di Farmi Roberto.

Corso 8051

CAPITOLO 2 – I REGISTRI SFR

Cosa sono i registri SFR?

L'8051 e' un microcontrollore flessibile con un discreto numero di modi operativi. Il vostro programma puo' leggere e/o modificare il modo operativo dell'8051 cambiando i valori dei registri SFR.

Si accede ai registri SFR come se fossero una parte della normale RAM interna. L'unica differenza e' che la RAM interna e' compresa fra gli indirizzi da 00h a 7Fh, mentre i registri SFR risiedono nell'area dall'indirizzo 80h all'indirizzo FFh.

Ogni registro SFR ha un proprio indirizzo ed un proprio nome. La tabella sottostante fornisce: una rappresentazione grafica dei registri SFR dell'8051, la denominazione e l'indirizzo.

80	P0	SP	DPL	DPH				PCON	87
88	ICON	TMOD	TL0	TL1	TH0	TH1			8F
90	P1								97
98	SCON	SBUF							9F
A0	P2								A7
A8	IE								AF
B0	P3								B7
B8	IP								B9
C0									C7
C8									CF
D0	PSW								D7
D8									DF
E0	ACC								E7
E8									EF
F0	B								F7
F8									FF

 Blue background are I/O port SFRs
Yellow background are control SFRs
Green background are other SFRs

Come potete osservare, anche se il range degli indirizzi da 80h a FFh offre 128 possibili bytes, ci sono solo 21 registri SFR nell'8051 standard. Tutti gli altri indirizzi sono considerati non validi. Leggere o scrivere in questi registri puo' produrre un valore indefinito oppure un comportamento anomalo.

Suggerimento per il programmatore: Si raccomanda di non leggere o scrivere nei registri SFR non assegnati Cio' puo' provocare un comportamento anomalo e puo' rendere il vostro programma incompatibile con altri derivati dell'8051 che usano quei registri SFR per altri scopi.

Tipi di SFR

Come risulta evidente dalla tabella, i registri SFR che hanno uno sfondo blu sono relativi a operazioni di I/O. L'8051 dispone di quattro porte di I/O ad 8 bit per un totale di 32 linee di I/O. Per porre una linea di I/O e' allo stato alto o basso o leggere il suo valore vengono usati i registri SFR in verde.

I registri SFR con lo sfondo in giallo servono a controllare il modo operativo o la configurazione dell'8051. Per esempio il registro **TCON** controlla i timer, mentre il registro **SCON** controlla la porta seriale.

I restanti registri SFR a sfondo verde, sono registri ausiliari nel senso che essi non configurano direttamente il modo operativo del microprocessore, ma l'8051 non potrebbe operare senza di essi. Per esempio, una volta che la porta seriale e' stata configurata usando il registro **SCON** il programma puo' leggere o scrivere sulla porta seriale usando il registro **SBUF**.

Suggerimento per il programmatore: I registri SFR il cui nome appare scritto in rosso nella tabella sovrastante sono quelli ai quali si puo' accedere con operazioni bit a bit (p.e. usando le istruzioni **SETB** e **CLR**). Gli altri registri SFR non possono essere usati nella modalita' bit a bit. Come potete notare, tutti i registri SFR i cui indirizzi sono multipli interi di 8 possono essere usati con modalita' bit a bit.

Descrizione dei registri SFR

Questa sezione vi fornira' una breve panoramica dei registri SFR standard che trovate nella mappa. Non e' intenzione di questa sezione spiegare completamente la funzionalita' di ogni registro SFR, poiche' cio' verra' affrontato in capitoli diversi del corso. Questa sezione serve solo a darvi un'idea generale di cosa facciano i registri SFR.

P0 (Porta 0, Indirizzo 80h, Indirizzabile a bit): Questa e' la porta P0 di input/output. Ciascun bit di questo registro SFR corrisponde ad un pin del microcontrollore. Per esempio, il bit 0 e' il pin P0.0, il bit 7 e' il pin P0.7. Settare un bit di questo registro SFR equivale a forzare ad un livello alto il corrispondente pin di I/O, mentre resettarlo equivale a forzarlo ad un basso livello.

Suggerimento per il programmatore: Siccome l'8051 dispone di 4 porte di I/O (P0, P1, P2 e P3), se il vostro hardware usa della RAM esterna oppure della ROM per il codice, le porte P0 a P2 non possono essere utilizzate. Questo e' dovuto al fatto che in questa modalita' di funzionamento, l'8051 usa queste porte per indirizzare la memoria esterna. In questo caso potete usare solo le porte P1 e P3.

SP (Stack Pointer, Indirizzo 81h): Questo e' lo stack pointer del microcontrollore. Questo registro SFR punta al prossimo valore dello stack nella RAM interna. Se caricate un valore nello stack (operazione di push) tale valore sara' memorizzato all'indirizzo SP+1. Supponiamo che SP contenga il valore 07h, una istruzione di PUSH carichera' il valore nello stack all'indirizzo 80h. Questo registro SFR viene modificato da tutte le istruzioni che comportano operazioni sullo stack quali: PUSH, POP, LCALL, RET, RETI, e ogni volta che il microcontrollore esegue un'operazione di interrupt.

Suggerimento per il programmatore: Lo stack pointer allo start up, e' inizializzato a 07h. Questo significa che lo stack parte da 08h e si espande in alto nella RAM interna. Poiche' i banchi di registri 1, 2 e 3 come pure le variabili a bit occupano la RAM interna da 08h a 2Fh, e' necessario inizializzare diversamente lo stack pointer del vostro programma se intendete usare i banchi di registri e/o la memoria a bit. Non e' una cattiva idea inizializzare lo stack pointer ad un valore pari a 2Fh nella prima istruzione di ogni vostro programma a meno che non siate sicuri al 100% di non dover usare i registri a banchi e le variabili a bit.

DPL/DPH (Data Pointer Basso/Alto, Indirizzi 82h/83h): I registri DPL e DPH lavorano insieme per

formare un valore a 16-bit chiamato *Data Pointer*. Il Data Pointer e' usato nelle operazioni che riguardano la memoria RAM esterna ed alcune istruzioni che riguardano il codice di programma. Essendo un valore intero unsigned, esso puo' rappresentare valori da 0000h a FFFFh (da 0 a 65535 decimale).

Suggerimento per il programmatore: DPTR e' in realta' un valore a 16-bit ottenuto raggruppando insieme i registri DPH e DPL. In verita' quando avete a che fare con il registro DPTR quasi sempre usate un byte alla volta. Per esempio, per mettere nello stack il registro DPTR (operazione di push) dovete prima effettuare il push del registro DPL e quindi quello del registro DPH. Non potete semplicemente effettuare il push del registro DPTR. Eccezionalmente esiste un'istruzione per "incrementare il DPTR". Quando eseguite questa istruzione, i due byte operano come se fossero un solo registro a 16-bit. Comunque non esiste un'istruzione per decrementare DPTR. Se volete decrementare DPTR dovete scrivere un sezione di codice apposta.

PCON (Power Control, Indirizzo 87h): Il registro PCON (Power Control) e' usato per controllare il power mode dell'8051. In alcune modalita' di funzionamento e' permesso all'8051 di mettersi nello stato "sleep" che richiede molta minor potenza. Questa modalita' di funzionamento e' controllata dal registro PCON. In piu', uno dei bit di PCON e' usato per raddoppiare il baud rate effettivo della porta seriale dell'8051.

TCON (Timer Control, Indirizzo 88h, Indirizzabile a bit): Il registro TCON (Timer Control) e' usato per configurare e modificare il modo di funzionamento dei timer dell'8051. Questo registro abilita o disabilita il funzionamento dei due timer e contiene un flag per indicare quando un timer va in overflow. Alcuni bit del registro TCON non sono relativi all'uso del timer ma servono a configurare gli interrupt esterni e contengono dei flag che segnalano che un interrupt esterno e' stato attivato.

TMOD (Timer Mode, Indirizzo 89h): Il registro TMOD (Timer Mode) e' usato per configurare il modo di funzionamento di ciascuno dei due timer. Mediante tale registro e' possibile far funzionare un timer come un timer a 16-bit, oppure un timer a 8-bit con auto-caricamento, oppure come un timer a 13 bit o come due timer separati. Inoltre e' possibile configurare il timer come contatore quando un pin esterno e' attivo oppure come "contatore di eventi" segnalati da un apposito pin esterno.

TL0/TH0 (Timer 0 Basso/Alto, Indirizzi 8Ah/8Bh): Questi due registri uniti insieme danno lo stato del timer 0. Il loro comportamento e' determinato da come e' stato configurato il timer 0 nel registro TMOD; in ogni caso essi contano sempre in modo ascendente. Quello che e' possibile configurare e' la maniera e il momento nel quale essi devono incrementare il valore.

TL1/TH1 (Timer 1 Basso/Alto, Indirizzi 8Ch/8Dh): Questi due registri uniti insieme danno lo stato del timer 1. Il loro comportamento e' determinato da come e' stato configurato il timer 1 nel registro TMOD; in ogni caso essi contano sempre in avanti. E' possibile configurare sia la maniera che il momento nel quale il timer deve incrementare il suo valore.

P1 (Porta 1, Indirizzo 90h, Indirizzabile a bit): Questa e' la porta P1 di input/output. Ciascun bit di questo registro SFR corrisponde ad un pin del microcontrollore. Per esempio, il bit 0 e' il pin P1.0, il bit 7 e' il pin P1.7. Settare un bit di questo registro SFR equivale a forzare a livello alto il corrispondente pin di I/O, mentre resettarlo equivale a forzarlo a livello basso.

SCON (Serial Control, Indirizzo 98h, Indirizzabile a bit): Il registro SCON (Serial Control) e' usato per configurare il comportamento della porta seriale dell'8051. Questo registro controlla: il baud rate della porta seriale, l'abilitazione della ricezione e contiene dei flag per indicare quando un byte e' stato trasmesso o ricevuto con successo.

Suggerimento per il programmatore: Per usare la porta seriale dell'8051 e' generalmente necessario inizializzare i seguenti registri SFR: SCON, TCON e TMOD. Cio' e' dovuto al fatto che il registro SCON e'

usato solo per controllare la porta seriale. Nella maggior parte dei casi il programma prevede di usare uno dei timer come generatore di baud rate. In questo caso va configurato il timer corrispondente mediante i registri TCON e TMOD.

SBUF (Serial Buffer, Indirizzo 99h): Il registro SBUF (Serial Buffer) e' usato per inviare e ricevere i dati dalla porta seriale dell'8051. Ogni valore scritto nel registro SBUF e' inviato sulla porta seriale dal pin TXD. Allo stesso modo, ogni valore ricevuto dalla seriale attraverso il pin RXD e' reso disponibile al programma applicativo dell'utente mediante una lettura del registro SBUF. In altre parole il registro SBUF serve da porta di uscita quando viene scritto e da porta d'ingresso quando viene letto.

P2 (Porta 2, Indirizzo A0h, Indirizzabile a bit): Questa e' la porta P2 di input/output. Ciascun bit di questo registro SFR corrisponde ad un pin del microcontrollore. Per esempio, il bit 0 e' il pin P2.0, il bit 7 e' il pin P2.7. Settare un bit di questo registro SFR equivale a forzare ad un livello alto il corrispondente pin di I/O, mentre resettarlo equivale a forzarlo ad un basso livello.

IE (Interrupt Enable, Indirizzo A8h): Il registro IE (Interrupt Enable) e' usato per abilitare e disabilitare gli interrupt. I 7 bit meno significativi di IE sono usati per abilitare o disabilitare gli interrupt individualmente, mentre il bit piu' significativo e' usato per abilitare o disabilitare tutti gli interrupt contemporaneamente. Per cui, se tale bit e' nello stato 0 tutti gli interrupt sono disabilitati anche se il corrispondente bit e' attivo.

P3 (Porta 3, Indirizzo B0h, Indirizzabile a bit): Questa e' la porta P3 di input/output. Ciascun bit di questo registro SFR corrisponde ad un pin del microcontrollore. Per esempio, il bit 0 e' il pin P3.0, il bit 7 e' il pin P3.7. Settare un bit di questo registro SFR equivale a forzare ad un livello alto il corrispondente pin di I/O, mentre resettarlo equivale a forzarlo ad un basso livello.

IP (Interrupt Priority, Indirizzo B8h, Indirizzabile a bit): Il registro IP (Interrupt Priority) e' usato per cambiare la priorita' di ciascun interrupt. Sull'8051 un interrupt puo' essere a bassa priorita' (0) oppure ad alta priorita' (1). Un interrupt puo' interrompere solo un altro interrupt a priorita' piu' bassa. Per esempio, se configurate l'8051 in modo tale che tutti gli interrupt siano a bassa priorita' eccetto quello relativo alla porta seriale, esso puo' sempre sospendere il sistema anche se c'e' un altro interrupt attivo. Nel momento in cui e' attivo l'interrupt della porta seriale, pero', nessun altro interrupt e' in grado di sospendere la routine in corso poiche' essa ha priorita' piu' elevata.

PSW (Program Status Word, Indirizzo D0h, Indirizzabile a bit): Il registro PSW (Program Status Word) e' usato per memorizzare alcuni bit importanti che vengono aggiornati nell'esecuzione delle istruzioni dell'8051. Il registro PSW contiene: il carry flag, il carry flag ausiliario, il flag di overflow ed il flag di parita'. Il registro PSW contiene anche i flag utilizzati per selezionare il banco di registri "R" attivo.

Suggerimento per il programmatore: Se scrivete un routine di interrupt (interrupt handler), e' buona norma *salvare sempre* lo stato del registro PSW nello stack all'inizio della routine e di ripristinarlo appena prima di ritornare al programma principale. Molte istruzioni modificano lo stato dei bit del registro PSW, per cui se la vostra routine di interrupt non preserva lo stato del registro PSW, il programma avra' dei comportamenti strani e imprevedibili e sara' difficile trovare il problema a causa del comportamento non deterministico del microcontrollore.

ACC (Accumulatore, Indirizzo E0h, Indirizzabile a bit): L'accumulatore e' il registro dell'8051 piu' usato poiche' esso viene coinvolto in quasi tutte le istruzioni. Esso risiede all'indirizzo E0h, cio' significa che l'istruzione **MOV A,#20h** in realta' corrisponde all'istruzione **MOV E0h,#20h**. E' comunque meglio usare la prima istruzione poiche' richiede un solo byte mentre la seconda ne richiede almeno due.

B (Registro B, Indirizzo F0h, Indirizzabile a bit): Il registro B e' utilizzato in due istruzioni: la

moltiplicazione e la divisione. Esso e' anche comunemente usato dai programmatori come registro ausiliario per memorizzare dei valori temporanei.

Altri registri SFR

La tabella precedente e' un sommario di tutti i registri SFR di un 8051 standard. Tutti i microcontrollori derivati dall'8051 devono avere questi registri come base per mantenere la compatibilita' indietro con lo standard MSC51.

E' pratica comune, delle fabbriche di semiconduttori che vogliono sviluppare un nuova versione di 8051, aggiungere altri registri SFR per supportare delle nuove funzioni nel chip derivato.

Per esempio, il microcontrollore Dallas Semiconductor DS80C320 e' compatibile verso l'alto con l'8051. Cio' significa che un qualsiasi programma sviluppato per un 8051 standard dovrebbe girare senza modifiche sul DS80C320. In questo caso tutti i registri SFR che abbiamo finora menzionato devono esistere anche nel componente della Dallas.

Il DS80C320, pero', fornisce molte nuove funzionalita' in piu' rispetto all'8051 standard che devono in qualche modo essere gestite. Cio' viene realizzato aggiungendo dei nuovi registri SFR a quelli gia' menzionati. Per esempio, poiche' il DS80C320 dispone di due porte seriali (invece di una soltanto come nell'8051) sono stati aggiunti due nuovi registri SFR denominati rispettivamente SBUF2 e SCON2. In aggiunta ai normali registri, il DS80C320 riconosce come validi anche questi due nuovi registri SFR e usa il loro valore per determinare il modo operativo della seconda porta seriale. Ovviamente, a questi due nuovi registri devono essere assegnati degli indirizzi non utilizzati nell'8051 originale. In questa maniera, un nuovo chip derivato dall'8051 puo' essere sviluppato con la capacita' di poter interpretare i programmi dell'8051 esistenti senza modificare il codice.

Suggerimento per il programmatore: Qualora state scrivendo un programma che utilizza dei nuovi registri SFR che sono specifici di un chip derivato e non sono contemplati nell'elenco dei registri che abbiamo analizzato, ricordate che esso non girera' su un 8051 standard. Percio' usate i registri SFR non standard solo se ritenete di dover usare il vostro programma su quel microcontrollore specifico. Nella stessa maniera, qualora forniate tale programma ad una terza parte, avvisateli che il vostro codice usa dei registri SFR non standard, per evitargli dei possibili rompicapo.

^ [INDICE](#)

< [Tipi di memoria](#)

> [Registri Base](#)

(C) Copyright 1997, 1998 by Vault Information Services. All Rights Reserved.

Le informazioni sono fornite senza alcuna garanzia.

Traduzione italiana di Sergio Salvitti.

Conversione in PDF di Farmi Roberto.

Corso 8051

CAPITOLO 3 – Registri Base

Accumulatore

Se avete lavorato con un qualsiasi altro linguaggio assembler il concetto di registro *Accumulatore* vi sarà già familiare.

L'Accumulatore, come suggerisce il nome, è usato come registro generale per accumulare i risultati di un gran numero di istruzioni. Esso può contenere un valore di 8-bit (1-byte) ed è il registro più versatile tra quelli dell'8051 per l'elevato numero di istruzioni che lo usano. Più della metà delle 255 istruzioni dell'8051 lo usano o lo manipolano in qualche maniera.

Per esempio se volete aggiungere il numero 10 al numero 20, il risultato dell'addizione verrà memorizzato nell'accumulatore. Una volta che avete il risultato nell'accumulatore il processo può continuare e il dato può eventualmente essere spostato in un'altro registro oppure nella memoria.

I registri "R"

I registri "R" sono un set di 8 registri denominati: R0, R1 etc. fino a R7.

Essi sono usati come registri ausiliari in molte operazioni. Per continuare con l'esempio precedente nel quale state addizionando 10 a 20. Supponiamo che il valore iniziale 10 risieda nell'accumulatore mentre il valore 20 risieda nel registro R4. Per eseguire l'operazione di addizione dovrete usare la seguente istruzione:

ADD A,R4

Al termine dell'istruzione l'Accumulatore conterrà il valore 30.

Possiamo dire che sono dei registri ausiliari o "di aiuto" molto importanti. L'Accumulatore da solo sarebbe poco utile se non avesse il supporto di questi registri.

I registri "R" sono anche usati per memorizzare dei valori temporanei. Per esempio, supponiamo vogliate addizionare il valore di R1 con R2 e sottrarre il risultato alla somma di R3 con R4. Una possibile strada per far ciò potrebbe essere:

```
MOV A,R3 ; Sposta il valore di R3 nell'accumulatore  
ADD A,R4 ; Addizionalo al valore di R4  
MOV R5,A ; Salva il risultato temporaneamente in R5  
MOV A,R1 ; Sposta il valore di R1 nell'accumulatore  
ADD A,R2 ; Addizionalo al valore di R1  
SUBB A,R5 ; Sottrai il risultato al valore di R5 (che contiene R3 + R4)
```

Come potete osservare, abbiamo usato R5 per memorizzare temporaneamente la somma di R3 con R4.

Certamente non e' questo il modo piu' efficiente per calcolare $(R1+R2) - (R3 +R4)$ ma tale esempio ci e' servito per illustrare l'uso dei registri "R" come memoria temporanea.

Il registro "B"

Il registro "B" e' molto simile all'Accumulatore nel senso che esso puo' memorizzare una parola di 8-bit.

Esso e' usato soltanto in due istruzioni dell'8051: MUL AB e DIV AB. Quindi se volete moltiplicare o dividere facilmente e velocemente un numero con un altro, potete memorizzare il secondo numero nel registro "B" ed usare queste due istruzioni.

Oltre che alle due istruzioni MUL e DIV, il registro "B" e' spesso usato come un altro registro temporaneo come se fosse il nono registro "R".

Data Pointer (DPTR)

Il registro Data Pointer (DPTR) e' un registro dell'8051 accessibile soltanto a 16-bit. Gli altri registri A, B ed R sono tutti ad un singolo byte.

Il DPTR, come suggerisce il nome (Puntatore dati) e' usato per indirizzare dei dati. Esso e' usato da molti comandi che richiedono all'8051 l'accesso alla memoria esterna. Infatti tutte le volte che l'8051 deve indirizzare un byte della memoria esterna usa l'indirizzo contenuto nel registro DPTR.

Anche se lo scopo iniziale del DPTR e' quello di indirizzare la memoria esterna, molti programmatori sfruttano il fatto che esso sia l'unico vero registro a 16 bit e lo usano spesso per memorizzare valori a 2-byte. Questo modo di usarlo, quindi, non ha nulla a che fare con le locazioni di memoria.

Program Counter (PC)

Il Program Counter (PC – Contatore di Programma) e' un indirizzo a 2-bytes che punta alla locazione di memoria programma dove l'8051 andra' a prendere la prossima istruzione da eseguire. Allo startup, l'8051 inizializza PC sempre al valore 0000h e lo incrementa ogni volta che un istruzione viene eseguita. Bisogna pero' notare che non sempre il PC viene incrementato di uno. Poiche' alcune istruzioni richiedono 2 o 3 byte il PC verra' incrementato di 2 o di 3 opportunamente.

Il Program Counter e' speciale, nel senso che non esiste una maniera diretta per modificarne il valore. Non e' possibile, cioe' eseguire un'operazione del tipo $PC=2430h$. L'unica maniera alternativa e' quella di eseguire: LJMP 2340h.

E' inoltre interessante notare che mentre e' possibile cambiare il valore del PC (magari eseguendo un'istruzione di salto) sembrerebbe impossibile leggere il valore del Program Counter. Questo, pero', non e' completamente vero. In realta' esiste un trucco per determinare quale sia il valore attuale del PC. Questo trucco verra' svelato in un capitolo successivo.

Stack Pointer (SP)

Lo Stack Pointer (SP – Puntatore dello Stack), come tutti gli altri registri esclusi DPTR e PC, puo' contenere un valore a 8-bit. Lo stack pointer e' usato per indicare la cima dello stack (catasta).

Quando effettuate il push di un valore nello stack (lo accatastate), l'8051 prima incrementa il valore di SP e poi memorizza il valore nella locazione di memoria puntata da esso.

Quando effettuate il pop di un valore dallo stack (lo togliete dalla catasta), l'8051 prende il valore puntato dallo Stack Pointer e dopo decrementa il suo valore.

L'ordine con cui sono effettuate le operazioni di stack e' importante. Allo startup, l'8051 inizializza automaticamente SP al valore 07h. Se immediatamente richiedete un push, il valore verra' memorizzato all'indirizzo 08h della RAM interna. Cio' e' giustificato da quello che avevano detto precedentemente. Prima viene incrementato SP (da 07h a 08h) e poi viene memorizzato il valore nella locazione di memoria all'indirizzo (08h).

La modifica di SP e' effettuata in maniera automatica dall'8051 quando esegue le sei istruzioni seguenti: PUSH, POP, ACALL, LCALL, RET, e RETI. E' inoltre usato nelle operazioni di interrupt (di questo parleremo in seguito, per il momento non preoccupatevi!).

^ [INDICE](#)

< [Registri speciali](#)

> [Modi d'indirizzamento](#)

(C) Copyright 1997, 1998 by Vault Information Services. All Rights Reserved.

Le informazioni sono fornite senza alcuna garanzia.

Traduzione italiana di Sergio Salvitti.

Conversione in PDF di Farmi Roberto.

Corso 8051

CAPITOLO 4 – Modi d'indirizzamento

Quando parliamo di modi d'indirizzamento, ci riferiamo a come una locazione di memoria viene indirizzata.

L'8051 dispone dei seguenti modi d'indirizzamento:

Indirizzamento Immediato	MOV A,#20h
Indirizzamento Diretto	MOV A,30h
Indirizzamento Indiretto	MOV A,@R0
Esterno Diretto	MOVX A,@DPTR
Indiretto a Codice	MOVC A,@A+DPTR

Ognuno di questi modi d'indirizzamento fornisce importanti flessibilita'.

Indirizzamento Immediato

L'Indirizzamento Immediato e' chiamato cosi' perche' il valore da memorizzare risiede immediatamente nel codice operativo nella memoria programma. Cio' significa che l'istruzione contiene in se' stessa il valore da memorizzare.

Per esempio, l'istruzione:

MOV A,#20h

usa l'indirizzamento immediato perche' l'accumulatore sara' caricato con il valore che segue immediatamente (una costante); in questo caso 20 (esadecimale).

L' indirizzamento immediato e' molto veloce poiche' il valore da caricare e' gia contenuto nell'istruzione stessa. Come inconveniente, pero', il valore e' stabilito al momento della compilazione e questo ne limita la flessibilita'.

Indirizzamento Diretto

L'Indirizzamento Diretto e' chiamato cosi' perche' il valore da memorizzare in memoria e' ottenuto direttamente prendendolo da un'altra locazione di memoria.

Per esempio, l'istruzione:

MOV A, 30h

leggera' il valore contenuto nella locazione di RAM interna all'indirizzo 30 (esadecimale) e lo porra' nell'accumulatore.

L'Indirizzamento Diretto e' generalmente veloce, poiche', anche se il valore da caricare non e' incluso nell'istruzione, esso e' comunque facilmente accessibile, in quanto risiede nella memoria interna dell'8051. Esso e' inoltre molto piu' flessibile dell'Indirizzamento Immediato poiche' il valore da caricare e' contenuto in un determinato indirizzo e quindi puo' essere modificato.

Va inoltre notato che, ogni volta che usate l'indirizzamento diretto con valori compresi da 00h a 7Fh fate riferimento alla Memoria Interna. Mentre se usate indirizzi da 80h a FFh fate riferimento ai registri SFR che servono a controllare i modi operativi del microcontrollore stesso.

A questo punto sorge spontanea la domanda; " Se l'indirizzamento diretto nell'area da 80h a FFh fa riferimento ai registri SFR, come posso indirizzare i 128 byte superiori della RAM interna dell'8052?". La risposta e': Non potete usare l'indirizzamento diretto per far riferimento a quell'area di memoria. Ad ogni modo potete accedere a tale parte di RAM usando il prossimo modo d'indirizzamento: quello indiretto.

Indirizzamento Indiretto

L'indirizzamento Indiretto e' un modo molto potente che in molti casi fornisce un elevato grado di flessibilita'. Esso e' inoltre l'unica maniera per accedere ai 128 byte extra della RAM interna dell'8052.

L'indirizzamento indiretto assume la seguente forma:

MOV A,@R0

Questa istruzione dice all'8051 di analizzare il valore contenuto nel registro R0. In seguito il microcontrollore carichera' nell'accumulatore il valore contenuto nella RAM interna alla locazione il cui indirizzo e' contenuto in R0.

Facciamo un esempio: supponiamo che R0 contenga il valore 40h e la RAM interna all'indirizzo 40h contenga il valore 67h. Al momento dell'esecuzione dell'istruzione, l'8051 va a leggere l'indirizzo della RAM in R0, poi usa tale indirizzo per prendere il valore 67h e lo carica nell'accumulatore.

L'indirizzamento indiretto e' sempre riferito alla RAM interna; non e' mai relativo ad un registro SFR. Qualcuno potrebbe pensare di usare l'indirizzamento indiretto per scrivere sulla porta seriale (SFR 99h) nella seguente maniera:

MOV R0,#99h ;Carica in R0 l'indirizzo della porta seriale
MOV @R0,#01h ;Invia 01h sulla porta seriale... **SBAGLIATO!**

Questa operazione e' concettualmente non valida. Siccome l'indirizzamento indiretto fa sempre riferimento alla RAM interna, le due istruzioni di sopra, scriveranno 01h all'indirizzo 99h dell'8052. Nel caso di un 8051, le due istruzioni produrranno un risultato indefinito, perche' esso dispone di soli 128 byte.

Esterno Diretto

La Memoria Esterna puo' essere usata con un modo d'indirizzamento chiamato "Esterno Diretto". Il suo nome deriva dal fatto che esso e' simile all'indirizzamento diretto, ma e' usato per accedere alla memoria esterna invece che interna.

Ci sono solo due istruzioni che usano questo modo d'indirizzamento:

MOVX A,@DPTR
MOVX @DPTR,A

Come potete notare, ambedue le istruzioni utilizzano DPTR. In queste istruzioni, il registro DPTR deve essere caricato con l'indirizzo della memoria esterna alla quale si vuole accedere. Una volta che il registro DPTR contiene l'indirizzo corretto, la prima istruzione caricherà il valore della memoria nell'accumulatore, mentre la seconda farà l'operazione opposta, spostando il valore dell'accumulatore nella memoria esterna.

Esterno Indiretto

La memoria esterna può anche essere indirizzata usando una forma di indirizzamento indiretto chiamato "esterno indiretto". Questa forma di indirizzamento è usualmente utilizzata in progetti relativamente piccoli che hanno una piccola quantità di RAM esterna.

Un esempio di tale indirizzamento è il seguente:

MOVX @R0,A

Ancora una volta, il valore di R0 è usato come indirizzo di memoria esterna dove caricare il valore dell'accumulatore. Poiché il valore di @R0 può soltanto essere incluso nel range da 00h a FFh, il progetto dovrebbe effettivamente avere una RAM esterna di soli 256 byte. Ci sono degli accorgimenti relativamente semplici implementati in hardware/software per accedere a più di 256 byte di memoria esterna usando l'indirizzamento in oggetto, comunque risulta più facile usare in questi casi l'indirizzamento Esterno Diretto.

^ [INDICE](#)

< [Registri Base](#)

> [Controllo di flusso del programma](#)

(C) Copyright 1997, 1998 by Vault Information Services. All Rights Reserved.

Le informazioni sono fornite senza alcuna garanzia.

Traduzione italiana di Sergio Salvitti.

Conversione in PDF di Farmi Roberto.

Corso 8051

CAPITOLO 5 – Controllo del flusso di programma

Al powerup, l'8051 scrive il valore 0000h nel PC. Da quel momento l'8051 inizia ad eseguire le istruzioni sequenzialmente in memoria a meno che un'istruzione del programma non alteri il PC. Ci sono diverse istruzioni che possono modificare il valore del PC tra le quali: le istruzioni di branching condizionali, i salti diretti e le chiamate e i ritorni a/da subroutine. In più gli interrupt, quando abilitati, possono causare la deviazione del flusso di programma dal suo schema sequenziale.

Branching Condizionale

L'8051 dispone di una serie di istruzioni che, come gruppo, sono denominate istruzioni di "branching (ramificazione) condizionale". Queste istruzioni forzano l'esecuzione del programma ad abbandonare la loro natura sequenziale se alcune condizioni risultano vere.

Prendiamo, per esempio, l'istruzione JB. Questa istruzione significa "Salta da una parte del programma se il Bit e' ad uno (Set)". Un esempio di questa istruzione potrebbe essere:

```
JB 45h,HELLO
NOP
HELLO: ....
```

In questo caso, l'8051 analizzerà il contenuto del bit alla locazione 45h. Se il bit e' settato l'esecuzione del programma salterà immediatamente alla label HELLO, tralasciando l'istruzione NOP. Se il bit non e' settato, la condizione di branch non sarà vera e l'esecuzione del programma continuerà, come usuale, con l'esecuzione dell'istruzione NOP che segue l'istruzione in corso.

Il branching condizionale e' realisticamente il blocco fondamentale della logica di programmazione. Infatti ogni decisione nel flusso del codice coinvolge un branching condizionale. Esso può essere considerato come la struttura "IF...THEN" del linguaggio assembler dell'8051.

Una cosa importante vale la pena sottolineare circa il branching condizionale e riguarda il fatto che il programma può saltare ad una locazione di memoria collocata 128 byte prima o 127 byte dopo l'indirizzo dal quale inizia l'istruzione di branch condizionale. Ciò significa che, nell'esempio precedente, la label HELLO deve trovarsi nel range +/- 128 byte dal punto in cui risiede l'istruzione di branching condizionale.

Salto Diretti

Anche se il branching condizionale e' estremamente importante, spesso e' necessario saltare direttamente ad una locazione di memoria senza necessariamente prendere alcuna decisione logica. Ciò equivale al "Goto" del BASIC. In questo caso volete che il programma continui a girare da un determinato indirizzo senza condizioni.

Cio' e' ottenuto nell'8051 usando le istruzioni "Direct Jump e Call". Come appena illustrato, la suite di istruzioni forza il flusso del programma a cambiare in maniera incondizionata.

Consideriamo il seguente esempio:

```
LJMP NEW_ADDRESS  
.  
.  
.  
NEW_ADDRESS: ....
```

L'istruzione LJMP in questo esempio significa "Long Jump (Salto lungo)." Quando l'8051 esegue questa istruzione il PC e' caricato con l'indirizzo corrispondente a NEW_ADDRESS e il programma riprende sequenzialmente da questo punto.

L' ovvia differenza tra le istruzioni precedenti e il branching condizionale e' che nel primo caso il programma e' sempre costretto a saltare mentre nel secondo esso salta solo se la condizione richiesta risulta vera.

C'e' da ricordare che oltre a LJMP, ci sono altre due istruzioni che causano il salto diretto del programma e sono: SJMP e AJMP. Funzionalmente queste due istruzioni si comportano alla stessa maniera di LJMP ma differiscono come segue:

- L'istruzione SJMP, come quelle relative al branching condizionale, puo' saltare solo ad indirizzi che distanti +/- 128 byte dall'istruzione stessa.
- L'istruzione AJMP puo' saltare ad un indirizzo che risiede nello stesso blocco di memoria di 2k ove si trova l'istruzione stessa. Supponendo che l'istruzione AJMP inizi all'indirizzo 650h, essa puo' richiedere al programma di saltare in un indirizzo compreso tra 0000h e 07FFh (0 e 2047, decimale).

Qualcuno potrebbe domandarsi, "Perche' dovrei usare le istruzioni SJMP e AJMP che hanno delle restrizioni sull'ampiezza del salto se esse fanno la stessa cosa dell'istruzione LJMP che puo' saltare ovunque?". La risposta e' semplice: L'istruzione LJMP richiede 3 byte di memoria programma mentre le altre due ne richiedono soltanto due. Per cui, se state sviluppando un'applicazione con problemi di memoria limitata, potete spesso salvare un byte di memoria utilizzando le istruzioni a 2 byte AJMP/SJMP.

Ultimamente, ho scritto un programma di 2100 byte totali ma disponevo di una memoria di soli 2k (2048 byte). Sostituendo tutte le istruzioni LJMP con AJMP il programma si e' accorciato fino a diventare di 1950 byte. A questo punto, senza cambiare la logica nel mio programma, risparmiando 150 byte e' stato possibile far entrare il programma nei 2048 byte previsti.

NOTA: Alcuni assembler di qualita' effettuano la conversione automaticamente. Essi sostituiscono, cioe' le istruzioni LJMP con le SJMP laddove e' possibile. Questa e' una capacita' molto elegante e potente che vorreste vedere in un assembler se pensate di sviluppare molti progetti che hanno consistenti restrizioni di memoria.

Chiamate dirette

Un'altra operazione che sara' familiare ai programmatori esperti e' l'istruzione LCALL. Essa e' simile al comando "Gosub" in BASIC.

Quando l'8051 esegue l'istruzione LCALL, pone immediatamente il contenuto del Program Counter nello

stack (operazione di push) e continua l'esecuzione del codice all'indirizzo contenuto nell'istruzione LCALL.

Ritorno da Routine

Un'altra istruzione che puo' causare il cambio sul flusso di programma e' l'istruzione di "Ritorno da Subroutine", conosciuta come RET nel linguaggio assembler dell'8051.

L'istruzione RET, quando eseguita, fa tornare il programma all'indirizzo che seguiva l'istruzione che aveva effettuato la chiamata a subroutine. Piu' in dettaglio, esso ritorna all'indirizzo caricato nello stack.

L'istruzione RET e' diretta nel senso che cambia sempre il flusso del programma senza condizioni, ma e' variabile poiche' dipende da quale subroutine era stata chiamata originariamente.

Interrupt

Un interrupt e' una caratteristica speciale che permette all'8051 di fornire l'illusione del "multi-tasking" anche se esso esegue una sola istruzione alla volta. La parola "interrupt" puo' essere spesso sostituita con la parola "evento".

Un interrupt viene attivato quando il corrispondente evento si presenta. In questo caso, l'8051 congela momentaneamente la normale esecuzione del programma ed esegue una speciale sezione di codice denominata "interrupt handler". Questo spezzone di codice sbriga le funzioni richieste dall'evento e restituisce il controllo all'8051 dal punto in cui era stato fermato, in maniera tale che l'esecuzione del programma possa proseguire come se non fosse mai stato interrotto.

L'argomento degli interrupt e' qualcosa di complicato e molto importante. Per questa ragione un intero capitolo sara' dedicato a questo tema. Per ora, e' sufficiente dire che gli interrupt possono causare un cambio al flusso del programma.

^ [INDICE](#)

< [Modi d'indirizzamento](#)

> [Informazioni a basso livello](#)

(C) Copyright 1997, 1998 by Vault Information Services. All Rights Reserved.

Le informazioni sono fornite senza alcuna garanzia.

Traduzione italiana di Sergio Salvitti.

Conversione in PDF di Farmi Roberto.

Corso 8051

CAPITOLO 6 – Informazioni a basso livello

Per capire ed usare al meglio l'8051, e' necessario conoscere alcune informazioni di dettaglio che riguardano il timing.

L'8051 opera sulla base di un clock fornito da un quarzo esterno. Esso e' un dispositivo elettrico che, una volta alimentato, emette degli impulsi elettrici a frequenza fissa. E' possibile trovare sul mercato quarzi che oscillano su una qualsiasi frequenza in funzione dell'applicazione richiesta. Nell'uso dell'8051 le frequenze piu' usate sono 12 e 11,059 Megahertz (normalmente la seconda e' la piu' comune). Perche' uno dovrebbe usare un valore di frequenza con un valore non intero? C'e' una ragione per fare cio'? Essa e' stata infatti scelta per generare agevolmente la frequenza del generatore di baud rate. Ne parleremo piu' diffusamente nel capitolo sulla comunicazione seriale. Al momento assumiamo di usare un quarzo che oscilla a 11,059 MHz.

I Microcontrollori (e molti altri dispositivi elettronici) usano i quarzi per sincronizzare le operazioni. L'8051 usa il quarzo proprio per questo scopo. In effetti, l'8051 opera sulla base dei cosiddetti "cicli macchina". Un singolo ciclo macchina e' la minima quantita' temporale che un'istruzione dell'8051 richiede per essere eseguita. Comunque molte istruzioni richiedono piu' cicli macchina.

In realta', un ciclo macchina corrisponde a 12 impulsi dell'oscillatore a quarzo. Quindi, se un'istruzione viene completata in un ciclo macchina, essa richiede 12 impulsi dell'oscillatore. Essendo nota la frequenza del quarzo a 11.059.000 cicli al secondo, possiamo calcolare quanti cicli macchina al secondo possono essere eseguiti dall'8051:

$$11.059.000 / 12 = 921.583$$

Questo vuol dire che l'8051, sincronizzato con un oscillatore a 11,059 MHz puo' eseguire 921.583 cicli macchina al secondo. La maggior parte delle istruzioni dell'8051 sono a ciclo-singolo, per cui possiamo ritenere grossolanamente che l'8051 esegua circa un milione di istruzioni al secondo. Nel caso volessimo essere piu' realistici, tenendo conto che il tempo di esecuzione dipende dal un numero di cicli macchina sicuramente maggiore di uno, possiamo stimare mediamente circa 600.000 istruzioni al secondo.

Per esempio, se utilizzassimo esclusivamente istruzioni a 2 cicli macchina, l'8051 eseguirebbe 460.791 istruzioni al secondo. L'8051 ha anche due istruzioni molto lente che richiedono 4 cicli. Nel caso in cui per assurdo utilizzaste solo queste istruzioni, l'8051 eseguirebbe soltanto 230.395 istruzioni al secondo.

E' importante insistere sul fatto che non tutte le istruzioni vengono eseguite con la stessa quantita' di tempo. L'istruzione piu' veloce richiede un ciclo macchina (12 impulsi di clock), molte altre richiedono 2 cicli macchina (24 impulsi di clock) e le due piu' lente istruzioni matematiche richiedono 4 cicli macchina (48 impulsi di clock).

NOTA: Molti derivati dell'8051 hanno il timing delle istruzioni diverso. Per esempio, molte versioni dell'8051 ottimizzate in velocita', eseguono un ciclo macchina in soli 4 impulsi di clock invece di 12, per cui un particolare chip sarebbe in effetti tre volte piu' veloce del classico 8051 a parita' di frequenza di clock.

Poiche' tutte le istruzioni richiedono quantita' di tempo variabili, qualcuno potrebbe chiedersi: Come posso tenere traccia del tempo trascorso in una applicazioni che richiede risposte temporali critiche se non ho un riferimento temporale del mondo esterno?

Fortunamente, l'8051 dispone di timer che ci permettono di gestire gli eventi temporali con notevole precisione. Ma questo e' argomento del prossimo capitolo.

^ [INDICE](#)

< [Controllo di flusso del programma](#)

> [ITimer](#)

(C) Copyright 1997, 1998 by Vault Information Services. All Rights Reserved.

Le informazioni sono fornite senza alcuna garanzia.

Traduzione italiana di Sergio Salvitti.

Conversione in PDF di Farmi Roberto.

Corso 8051

CAPITOLO 7 – I Timer

L'8051 classico dispone di due timer, che possono essere configurati, controllati e letti in maniera indipendente. I timer dell'8051 hanno tre funzioni generali:

- Tenere traccia del tempo totale trascorso e/o calcolare il tempo trascorso tra due eventi,
- Contare il numero degli eventi o
- Generare il baud rate per la porta seriale.

I tre modi di usare un timer verranno discussi separatamente. I primi due saranno affrontati in questo capitolo mentre l'uso del timer come generatore di baud rate verrà discusso nel capitolo relativo alla porta seriale.

Come fa un timer a contare?

Come fa un timer a contare? La risposta è semplice. Esso conta sempre incrementando il suo valore. Non importa se viene usato come timer o come contatore o come generatore di baud rate. È sempre incrementato dal microcontrollore.

Suggerimento per il programmatore: Alcuni derivati dell'8051 consentono al programma di configurare il timer in maniera tale che possa contare sia in avanti che indietro. Ciò è al di fuori dello scopo di questo corso che è orientato all'8051 standard. Parliamo di questa possibilità nel caso in cui foste assolutamente nella necessità di avere un timer che conta all'indietro e quindi sappiate che esistono dei microcontrollori derivati dall'8051 che sono in grado di farlo.

USARE UN TIMER PER MISURARE UN TEMPO

Ovviamente uno degli usi primari dei timer è quello di misurare il tempo. Parleremo prima di questo uso e poi di quello relativo al conteggio di eventi. Quando un timer è usato per misurare il tempo è anche chiamato "interval timer" poiché esso sta misurando l'intervallo temporale tra due eventi.

Quanto tempo impiega un timer a contare?

Quando il timer è nel modo "interval timer" ed è correttamente configurato, esso viene incrementato di uno ogni ciclo macchina, ovvero 12 impulsi di clock. Perciò un timer attivo sarà incrementato:

$$11.059.000 / 12 = 921.583$$

921.583 volte al secondo. A differenza delle istruzioni che hanno durata variabile, il ciclo di un timer ha durata fissa; uno impulso per ciclo macchina. Se un timer ha contato da 0 a 50.000 possiamo calcolare:

$$50.000 / 921.583 = 0,0542$$

che sono trascorsi 0,0542 secondi .

Ovviamente non e' molto utile sapere che sono trascorsi 0,0542 secondi. Se voi voleste far partire un evento ogni secondo dovrete aspettare il conteggio del timer da 0 a 50.000 per 18,45 volte. Come si puo' aspettare per un tempo non intero? Non e' possibile. Allora proviamo a fare un calcolo impreciso.

Supponiamo di voler sapere quante volte il timer viene incrementato ogni 0,05 secondi. Possiamo fare questo semplice calcolo:

$$0,05 * 921.583 = 46.079.$$

Il risultato ci dice che impieghiamo 0,05 secondi (un ventesimo di secondo) per contare da 0 a 46.079. Piu' precisamente il tempo trascorso e' di 0,49999837 secondi, cio' vuol dire che mancano 0,000000163 secondi all'appello, comunque e' abbastanza preciso anche per i lavori commissionati dal governo. Considerate che se costruite un orologio basato sull'8051, con tali ipotesi, ammesso che il quarzo sia perfetto, avrete un secondo in anticipo ogni due mesi. Beh, io credo che cio' sia accurato per la maggior parte delle applicazioni. Magari il mio orologio andasse avanti soltanto di un secondo ogni due mesi!

Una volta stabilito che esso impiega un ventesimo di secondo e vogliamo eseguire un evento ogni secondo sara' sufficiente che il timer conti da 0 a 46.079 per venti volte; a questo punto eseguite le azioni richieste dall'evento, resettate il timer e aspettate che il timer completi di nuovo il conteggio altre venti volte. In questa maniera eseguirete effettivamente il vostro evento una volta al secondo con una precisione al ventesimo di secondo.

Abbiamo ora un sistema capace di misurare il tempo. Quello che ci serve ora e' sapere come controllare il timer e inizializzarlo in maniera che possiamo dargli le informazioni corrette.

I registri Timer SFR

Come precedentemente affermato, l'8051 ha due timer che funzionano ambedue nella stessa maniera. Uno e' chiamato TIMER0 e l'altro TIMER1. I due timer condividono due registri SFR (TCON e TMOD) che servono a controllarli. Ciascun timer pero' dispone di due SFR per loro uso esclusivo (TH0/TL0 e TH1/TL1).

Ai registri SFR abbiamo dato un nome mnemonico per individuarli facilmente, in effetti essi hanno un valore numerico. E' spesso utile conoscere l'indirizzo corrispondente al nome mnemonico del registro SFR. Per quello che riguarda i timer essi sono:

Nome SFR	Descrizione	Indirizzo SFR
TH0	Timer 0 Byte Alto	8Ch
TL0	Timer 0 Byte Basso	8Ah
TH1	Timer 1 Byte Alto	8Dh
TL1	Timer 1 Byte Basso	8Bh
TCON	Timer Control	88h
TMOD	Timer Mode	89h

Quando inserite il nome di un registro SFR nell'assembler, esso viene internamente automaticamente convertito in un indirizzo, Per esempio:

MOV TH0,#25h

carica il valore 25h nel registro TH0, tenuto conto che esso risiede nell'indirizzo 8Ch, sara' convertita in:

MOV 8Ch,#25h

Ora, torniamo a parlare del timer 0. Esso ha due registri TH0 e TL0, che senza complicarci la vita, possiamo ritenere essere il byte piu' e meno significativo del timer stesso. Se il timer 0 ha valore 0, entrambi i suddetti registri avranno valore 0. Quando il timer assume valore 1000 (decimale), TH0 conterra' il valore 3 (decimale) e TL0 il valore 232 (decimale).

Per ricordarvi come funziona la notazione byte basso/alto, ricordatevi di moltiplicare il byte alto per 256 e aggiungere il byte basso al risultato ottenuto. Ovvero:

$$TH0 * 256 + TL0 = 1000$$

Il timer 1 funziona esattamente nella stessa maniera ma dispone dei registri TH1 e TL1.

Poiche' solo due byte sono usati per ogni timer risulta evidente che il massimo valore che un timer puo' assumere e' 65.535 (decimale). Nel momento in cui viene raggiunto tale valore un ulteriore impulso di clock lo resettera' o per meglio dire lo mandera' in *overflow*.

Il registro SFR TMOD

Il registro TMOD e' usato per controllare il modo operativo di ambedue i timer. Ogni bit del registro fornisce al microcontrollore una specifica informazione che riguarda il funzionamento del timer. I 4 bit piu' significativi (da 4 a 7) sono relativi al timer 1 mentre i 4 bit meno significativi (da 0 a 3) hanno il medesimo significato ma sono relativi al timer 0.

I bit del registro TMOD hanno il seguente significato :

TMOD (89h) SFR

Bit	Nome	Spiegazione della funzione	Timer
7	GATE1	Quando questo bit e' settato, il timer 1 e' attivo solo quando il pin P3.3 e' nello stato alto. Se tale bit e' resettato il timer 1 sara' svincolato dallo stato del pin P3.3.	1
6	C/T1	Quando questo bit e' settato il timer 1 conterra' il numero degli eventi sul pin T1 (P3.5). Se il bit e' resettato il timer 1 verra' incrementato ogni ciclo macchina.	1
5	T1M1	Bit 1 di modo del timer 1(vedi sotto)	1

4	T1M0	Bit 0 di modo del timer 1(vedi sotto)	1
3	GATE0	Quando questo bit e' settato, il timer 0 e' attivo solo quando il pin P3.2 e' nello stato alto. Se tale bit e' resettato il timer 0 sara' svincolato dallo stato del pin P3.2.	0
2	C/T0	Quando questo bit e' settato il timer 0 contera' il numero degli eventi sul pin T1 (P3.5). Se il bit e' resettato il timer 0 verra' incrementato ogni ciclo macchina.	0
1	T0M1	Bit 1 di modo del timer 0(vedi sotto)	0
0	T0M0	Bit 0 di modo del timer 0(vedi sotto)	0

Come potete vedere dalla tabella di sopra, quattro bit, due per ciascun timer sono usati per specificare il modo operativo. I modi operativi sono:

TxM1	TxM0	Timer Mode	Descrizione
0	0	0	Timer a 13-bit
0	1	1	Timer a 16-bit
1	0	2	Timer a 8-bit con auto-reload
1	1	3	Timer in splitted mode

Modo a 13-bit (modo 0)

Il modo "0" corrisponde al funzionamento del timer a 13 bit. Questo e' un retaggio del passato che e' stato mantenuto per avere la compatibilita' con il suo predecessore: l'8048.

Generalmente questo modo di funzionamento non viene utilizzato in nuovi sviluppi.

In questa modalita' di funzionamento, TLx contera' da 0 a 31. Quando TLx viene incrementato da 31, esso si resettera' e incrementera' THx. Percio' effettivamente, soltanto 13 bit del timer a 2 byte sono utilizzati: i bit 0-4 di TLx e i bit 0-7 di THx. Questo significa anche in definitiva che il timer puo' contare solo fino ad un valore pari a 8192. Se caricate il timer con il valore 0, esso andra' in overflow e quindi a 0 dopo 8192 cicli macchina.

Modo a 16-bit (modo 1)

Il modo "1" corrisponde al funzionamento del timer a 16 bit. Questo e' un modo molto utilizzato. TLx e' incrementato da 0 a 255. All'overflow di TLx, THx viene incrementato di 1. Tenuto conto che e' un timer a 16 bit, esso puo' assumere 65536 valori distinti. Se caricate il timer con il valore 0, esso andra' in overflow e quindi a 0, dopo 65536 cicli macchina.

Modo a 8-bit (modo 2)

Il modo "2" corrisponde al funzionamento del timer ad 8 bit con auto-reload. Quando il timer e' configurato in modo 2, THx contiene il valore che deve essere caricato in TLx quando va in overflow. TLx inizia a contare in avanti. Quando raggiunge 255 e viene ulteriormente incrementato invece di tornare a 0 (come nei modi 0 e 1), assume il valore caricato in THx.

Se per esempio TH0 contiene il valore FDh e TL0 il valore FEh il conteggio procedera' nella seguente maniera:

Ciclo Macchina	Valore di TH0	Valore di TL0
1	FDh	FEh
2	FDh	FFh
3	FDh	FDh
4	FDh	FEh
5	FDh	FFh
6	FDh	FDh
7	FDh	FEh

Notate che il valore di TH0 non cambia mai e TL0 viene incrementato. Una volta raggiunto un valore pari a FFh, al successivo ciclo macchina TL0 assume lo stesso valore di TH0.

Ma qual'e' il vantaggio di usare tale modo di funzionamento? Ipotizziamo che vogliate che il timer assuma dei valori compresi tra 200 e 255. Nel modo 0 e 1 siete costretti a verificare da programma quando il timer va in overflow e quindi settarlo al valore 200. Questo lavoro richiede tempo prezioso e non vi garantisce un'elevata accuratezza a meno che non vogliate rimanere a controllare solo lo stato del timer. Quando usate il modo 2, non dovete preoccuparvi di tutto questo, poiche' l'hardware del microcontrollore lo fara' automaticamente per voi.

Il modo 2 e' usato molto spesso per generare il baud rate. Ne parleremo piu' approfondidamente nel capitolo dedicato alla Comunicazione Seriale.

Modo Split (modo 3)

Il modo "3" e' il cosiddetto modo a timer separati. Quando il timer 0 e' configurato per lavorare in questa modalita', si trasforma in due timer a 8 bit separati. Per capirci meglio, il timer 0 diventa TL0 e il timer 1 diventa TH0. Ambedue i timer contano da 0 a 255 e dopo l'overflow tornano a 0. Tutti i bit relativi al Timer 1 vengono assegnati a TH0.

Una volta che il Timer 0 e' configurato in modo split, il vero Timer 1 (cioe' TH1 e TL1) puo' essere configurato nei modi 0, 1, 2 come sempre, ma non puo' essere abilitato/disabilitato poiche' i suoi bit di controllo sono assegnati a TH0. Allora, il vero Timer 1 funzionera' in maniera libera e si incrementera' ad ogni ciclo macchina senza condizioni.

L'unico vero uso di questa modalita' e' quello per cui sia necessario avere due timer separati e, in aggiunta, un generatore di baud rate. In questo caso il Timer 1 viene utilizzato come baud generator e i due registri TH0 e TL0 come se fossero due timer separati.

Il registro TCON

Come ultimo c'e' un altro registro SFR che controlla i due timer e fornisce importanti informazioni sul loro funzionamento. Il registro TCON ha la seguente struttura:

TCON (88h) SFR

Bit	Nome	Indirizzo a Bit	Spiegazione della funzione	Timer
7	TF1	8Fh	Timer 1 Overflow. Questo bit e' settato quando il timer 1 e' andato in overflow	1
6	TR1	8Eh	Timer 1 Run. Quando questo bit viene settato il timer 1 e' abilitato, altrimenti e' fermo.	1
5	TF0	8Dh	Timer 0 Overflow. Questo bit e' settato quando il timer 0 e' andato in overflow	0
4	TR0	8Ch	Timer 0 Run. Quando questo bit viene settato il timer 0 e' abilitato, altrimenti e' fermo.	0

Come potete notare, abbiamo definito solo 4 degli 8 bit. Cio' e' dovuto al fatto che gli altri 4 bit del registro TCON non hanno nulla a che vedere con i timer. Essi sono relativi agli interrupt e quindi ne discuteremo nel capitolo dedicato a questo argomento.

Nella tabella precedente, c'e' un ulteriore colonna che riguarda l'indirizzo a bit. E' stata riportata perche' tale registro puo' essere indirizzato a bit, cio' vuol dire che, se vogliamo settare il bit TF1, che e' il bit piu' significativo di TCON, potremmo eseguire la seguente istruzione:

```
MOV TCON, #80h
```

oppure, visto che il registro e' indirizzabile a bit, potremo piu' semplicemente eseguire quest'altra istruzione:

```
SETB TF1
```

In quest'ultimo caso abbiamo il vantaggio di settare il bit piu' significativo di TCON senza cambiare il valore degli altri bit del registro. Generalmente infatti, quando si abilita o disabilita un timer, non si vuole modificare il valore degli altri bit di TCON, allora e' possibile sfruttare il fatto che tale registro e' indirizzabile a bit.

Inizializzare un timer

Ora che conosciamo tutti i registri SFR relativi ai timer, siamo in grado di scrivere il codice per inizializzare un timer e farlo partire.

Come vi ricordate, bisogna per prima cosa decidere in quale modalita' il timer verra' utilizzato. In questo caso scegliamo un modo a 16 bit che non ha alcuna dipendenza da pin esterni.

Dobbiamo per prima cosa, inizializzare il registro TMOD. Se lavoriamo con il timer 0 useremo i 4 bit meno significativi di TMOD. I primi due bit GATE0 e C/T0 vanno ambedue posti a 0 poiche' vogliamo che il timer sia indipendente da pin esterni. Il modo a 16-bit corrisponde al modo "1" e quindi dobbiamo: resettare T0M1 e settare T0M0. In breve l'unico bit da settare e' il bit 0 di TMOD. Per inizializzare il timer eseguiremo

la seguente istruzione:

```
MOV TMOD,#01h
```

Il timer 0 e' ora configurato per funzionare a 16 bit, ma e' disabilitato. Per attivarlo dobbiamo settare il bit TR0. Cio' puo' essere fatto mediante la seguente istruzione:

```
SETB TR0
```

Non appena verranno eseguite le precedenti due istruzioni, il timer 0 iniziera' a contare in avanti incrementando il suo valore ad ogni ciclo macchina (ogni 12 impulsi di clock).

Leggere lo stato di un Timer

Ci sono due maniere per leggere lo stato di un timer a 16-bit in funzione della specifica applicazione. Voi potete o leggere il valore attuale a 16 bit oppure stabilire soltanto se il timer e' andato in overflow.

Legger il valore del timer

Se il timer e' configurato nel modo ad 8 bit, sia nel modo con auto-reload che in split mode, la lettura del valore e' effettuata semplicemente leggendo il byte corrispondente al timer che state usando.

Quando pero' state usando il modo a 13 o 16 bit la lettura e' leggermente piu' complessa. Cio' e' dovuto al fatto che mentre state leggendo il valore del byte meno significativo del timer e questo valore e' pari a 255, potreste leggere un valore errato del byte piu' significativo che, in quel momento si sta incrementando. Per esempio se lo stato del timer e' 14/255 (byte alto = 14, byte basso = 255), potremmo leggere il valore 15/255 nel momento in cui il timer sta cambiando di stato al valore 15/0.

Quale potrebbe essere la soluzione? In verita' essa non e' molto complessa. Dovreste leggere prima il byte alto, poi quello basso e rileggere nuovamente il byte alto confrontandolo con il valore letto precedentemente. Sei i due valori coincidono, il timer non ha cambiato stato durante la lettura e quindi il valore letto e' utile, altrimenti va ripetuto il ciclo precedente. Il codice sara' del tipo:

```
REPEAT: MOV A,TH0
        MOV R0,TL0
        CJNE A,TH0,REPEAT
        ...
```

Un'altra maniera piu' semplice e' quella di congelare momentaneamente il timer (es. CLR TR0), leggere il valore del timer, e riabilitare subito il timer fermato (es. SETB TR0). In questo caso, essendo il timer fermo durante la lettura, non servono particolari precauzioni nella lettura. Ovviamente questo implica che il timer perdera' alcuni colpi di ciclo macchina. Dipende allora dalla specifica applicazione se cio' sia o meno tollerabile.

Rivelare l'Overflow di un timer

In alcuni casi e' sufficiente conoscere soltanto se il timer e' stato resettato o meno. Questo quando non interessa conoscere il valore del timer ma solo sapere se il timer ha raggiunto il massimo del conteggio (overflow). Una volta che il timer raggiunge il valore di 255, il microcontrollore lo riporta automaticamente a zero e setta il bit TFX corrispondente nel registro TCON. Per cui quando il bit TF0 e' settato vuol dire che il

timer 0 ha raggiunto l'overflow e alla stessa maniera se TF1 e' settato vuol dire che il timer 1 e' tornato a zero.

Possiamo usare questo approccio per costringere il programma ad eseguire delle operazioni ad intervalli di tempo prestabiliti. Riprendiamo pure l'esempio precedente nel quale era richiesto il conteggio da 0 a 46.079 per un periodo di un ventesimo di secondo. Se vogliamo impostare un tempo pari a quello indicato, tenuto conto che il timer segnala l'overflow quando passa per lo zero, dobbiamo settare il timer ad un valore pari a 65.536 meno 46.079, cioe' 19.457. Quindi per eseguire una pausa di un ventesimo di secondo usiamo il seguente frammento di codice:

```
MOV TH0,#76; Byte alto di 19.457 ( $76 * 256 = 19.456$ )  
MOV TL0,#01; Byte basso di 19.457 ( $19.456 + 1 = 19.457$ )  
MOV TMOD,#01; Configura il Timer 0 nel modo a 16-bit  
SETB TR0; Abilita il conteggio del timer Timer 0  
JNB TF0,$; Se il flag TF0 non e' settato rimane nella medesima istruzione.
```

Il simbolo "\$" viene usato nella maggior parte degli assembler per indicare l'indirizzo dell'istruzione in corso. Per cui, finche' il flag TF0 non e' settato e quindi il timer non e' andato in overflow, il programma rimane in loop nell'istruzione in corso. Al termine del ventesimo di secondo, il timer tornera' a zero settando il flag e a quel punto il programma uscirà fuori dal loop.

Misurare la durata di un evento

L'8051 dispone di un altro utile modo che puo' essere usato per misurare la durata di un evento.

Facciamo un esempio: vogliamo risparmiare energia in un ufficio e vogliamo conoscere quanto tempo una lampada rimane accesa durante la giornata. Quando la lampada e' accesa vogliamo conteggiare il tempo. Quando la lampada e' spento fermiamo il conteggio. Una possibile soluzione e' quella di collegare lo stato dell'interruttore ad un pin del micro. A questo punto il programma deve continuamente monitorare lo stato del pin utilizzato e attivare o disattivare il timer in funzione dello stato del pin stesso. Anche se il sistema funziona correttamente, In realta', l'8051 fornisce un metodo piu' semplice di quello appena prospettato.

Se diamo un'occhiata al registro TMOD, c'e' un bit chiamato GATE0. Finora abbiamo lasciato sempre il bit a zero perche' volevano che il timer continuasse a girare indipendentemente dallo stato dei fili esterni. Adesso pero', tale capacita' ci torna utile. Tutto quello che serve e' di collegare lo stato dell'interruttore al pin INTO (P3.2) dell'8051 e settare il bit GATE0. A questo punto, il timer eseguirà il conteggio solo quando il pin P3.2 si trovera' nello stato alto (interruttore on, luce accesa). Quando il pin P3.2 si trovera' nello stato basso (interruttore off, luce spenta) il timer si bloccherà automaticamente.

USARE I TIMER COME CONTATORI DI EVENTI

Finora abbiamo discusso di come un timer possa tener traccia del tempo trascorso, ma l'8051 ci permette di usare il timer come contatore di eventi.

Quando puo' essere utile questa funzione? Supponiamo di avere un sensore nel mezzo di una strada che invia un impulso ogni volta che passa un'auto. Questo potrebbe essere utilizzato per misurare l'intensita' del traffico su quella strada. Potremmo allora collegare il sensore ad un pin di I/O dell'8051 e monitorare lo stato del pin per contare il numero degli impulsi. Fare questo non e' eccessivamente difficile, ma richiede l'utilizzo di un po' di codice. Supponiamo per esempio di aver collegato il sensore al pin P1.0; il codice che conta il numero di auto che passano sara' del tipo:

JNB P1.0,\$; Aspetta che un'auto attivi la linea.

JB P1.0,\$; La linea e' ora attiva, la macchina sta passando.

INC COUNTER ; La linea e' bassa, quindi la macchina e' passata, possiamo contare l'evento.

Come potete osservare servono soltanto tre linee di codice. Ma che succede se dobbiamo fare qualche altra cosa nello stesso tempo, visto che il programma rimane quasi sempre fermo in loop sull'istruzione JNB? Certamente e' possibile trovare una soluzione, ma questa renderebbe il codice grande, complesso e poco elegante.

Fortunamente l'8051 ci fornisce un'alternativa per usare il timer come contatore di eventi senza doverci preoccupare di come realizzarlo in software. E' estremamente semplice e indolore: basta configurare un bit addizionale.

Per esempio vogliamo utilizzare il Timer 0 per conteggiare il numero delle auto che transitano. Allora, se diamo uno sguardo ai registri TCON possiamo notare la presenza di un bit chiamato C/T0, bit 2 di TCON (TCON.2). Se tale bit e' settato, il timer 0, invece di incrementare se stesso ad ogni ciclo macchina, effettua il monitoring della linea P3.4. In questa maniera incrementera' se stesso ogni volta che la linea passa dal valore alto a quello basso. A questo punto e' sufficiente collegare il sensore sulla linea P3.4 e configurare il registro TCON opportunamente.

E' importante pero' notare che l'8051 controlla lo stato della linea P3.4 una volta per ciclo macchina. Questo significa che, se la frequenza con la quale cambia lo stato del pin e' troppo elevata, l'8051 non riuscirà piu' a contare il numero di eventi in maniera corretta. Piu' precisamente, l'8051 riesce a contare un numero di eventi ad un massimo di un ventiquattresimo della frequenza di clock. Cio' vuol dire che, se per esempio usiamo un quarzo che oscilla a 12 MHz, esso riesce a conteggiare fino a 500.000 eventi al secondo ($12 \text{ MHz} * 1/24 = 500.000$).

[^ INDICE](#)

[< Informazioni a basso livello](#)

[> Porta Seriale](#)

(C) Copyright 1997, 1998 by Vault Information Services. All Rights Reserved.

Le informazioni sono fornite senza alcuna garanzia.

Traduzione italiana di Sergio Salvitti.

Conversione in PDF di Farmi Roberto.

Corso 8051

CAPITOLO 8 – Porta Seriale

Una delle caratteristiche piu' potenti dell'8051 e' l'UART integrata, conosciuta anche come porta seriale. Questo vuol dire che puoi facilmente leggere e scrivere dati sulla porta seriale. Senza tale periferica, simulare il comportamento della porta seriale richiederebbe un processo noioso di controllo dei pin di I/O in rapida successione per campionare un bit alla volta compresi i bit di start, stop e parita'.

Nel nostro caso invece, l'unica operazione da effettuare e' quella di configurare la porta seriale con il modo operativo ed il baud rate. Una volta configurata la porta e' sufficiente leggere o scrivere un registro SFR per ricevere od inviare dei dati in linea. L'8051 dal suo canto ci avvertira' automaticamente quando avra' finito di trasmettere un byte oppure quando avra' terminato di ricevere un byte. Non ci dobbiamo assolutamente preoccupare della trasmissione a livello di bit con un evidente risparmio di codice e tempo di elaborazione.

Inizializzare il modo di funzionamento della porta seriale

Il primo passo da compiere per usare la porta seriale e' quello di configurarla. Dobbiamo informare l'8051 di quanti bit vogliamo usare, quale sara' la sorgente del baud rate ed in quale modo verra' ricavato il baud rate dalla sorgente assegnata.

Prima di tutto analizziamo il registro SCON "Serial Control" e definiamo il significato di ogni bit:

	Nome	Indirizzo a bit	Spiegazione della funzione
7	SM0	9Fh	Bit 0 di modo
6	SM1	9Eh	Bit 1 di modo
5	SM2	9Dh	Abilitazione della Comunicazione Multiprocessore (vedi in seguito)
4	REN	9Ch	Receiver Enable. Questo bit deve essere settato per abilitare la ricezione.
3	TB8	9Bh	Transmit bit 8. Nono bit da trasmettere nel modo 2 e 3
2	RB8	9Ah	Receive bit 8. Nono bit ricevuto nel modo 2 e 3
1	TI	99h	Transmit Flag. Settato quando il byte e' stato trasmesso.
0	RI	98h	Receive Flag. Settato quando un byte e' stato ricevuto.

Vanno inoltre definiti i bit SM0 e SM1 con la seguente tabella:

SM0	SM1	Modo Seriale	Spiegazione	Baud Rate
0	0	0	Shift Register a 8-bit	Oscillatore/ 12
0	1	1	Uart a 8-bit	Dal Timer 1 (*)
1	0	2	Uart a 9-bit	Oscillatore/ 32 (*)

1	1	3	Uart a 9-bit	Dal Timer 1 (*)
---	---	---	--------------	-----------------

(*) Nota : Il baud rate indicato in questa tabella e' raddoppiato se PCON.7 (SMOD) e' settato.

Il registro SCON permette di configurare la porta seriale. Ora analizziamo ciascun bit in dettaglio.

I primi 4 bit (bit da 4 a 7) servono alla configurazione.

I bit **SM0** e **SM1** permettono di impostare il *modo seriale* da 0 a 3. I quattro modi sono definiti nella precedente tabella. Come potete osservare, il modo seriale seleziona sia il modo operativo che la maniera con la quale viene calcolato il baud rate. Nei modi 0 e 2 il baud rate e' fisso e basato sulla frequenza dell'oscillatore principale. Nei modi 1 e 3 il baud rate e' variabile e dipende dalla frequenza con la quale va in overflow il timer 1. Ne parleremo piu' diffusamente tra poco.

Il bit successivo, **SM2** e' un flag per impostare la "Comunicazione Multiprocessore". Normalmente, alla ricezione di un byte, l'8051 attivera' il flag "RI" (Receive Interrupt). Questo segnala al programma che un dato e' stato ricevuto e deve quindi essere processato. Quando SM2 e' attivo, il flag "RI" verra' attivato soltanto se il nono bit ricevuto ha il valore uguale ad "1". Per cui se SM2 e' attivo e il nono bit ricevuto e' zero, il flag RI non si attivera' mai. Questa funzionalita' puo' essere utile in alcune applicazioni che usano funzioni avanzate di comunicazione seriale. Per il momento e' opportuno porre il bit SM2 sempre a zero in maniera da avere il flag RI attivo ogni volta che viene ricevuto un carattere,

Il bit successivo, e' **REN** "Receiver Enable." Questo bit e' molto semplice. Se volete abilitare la ricezione della porta seriale dovete settarlo.

Gli ultimi 4 bit (bit da 0 a 3) sono bit operativi. Essi vengono impiegati durante la fase di ricezione e trasmissione dei dati e non sono quindi usati per configurare la porta seriale.

Il bit **TB8** e' usato nei modi 2 e 3 nei quali vengono trasmessi nove bit di dati. I primi 8 corrispondono a quelli inseriti nel registro SBUF e il nono bit e' preso da TB8.

Il bit **RB8** funziona essenzialmente come TB8, ma e' usato durante la ricezione. Nei modi 2 e 3 vengono ricevuti 9 bit. I primi 8 corrispondono a quelli letti in SBUF ed il nono viene copiato in RB8.

Il bit **TI** "Transmit Interrupt." viene usato per segnalare la fine della trasmissione di un byte. Quando viene scritto un byte in SBUF, la porta seriale impiega un po' di tempo per inviare i dati dalla porta seriale. Se il programma scrive un nuovo dato prima che il precedente sia stato completamente inviato, verranno mischiati i bit in trasmissione. Il bit TI serve a segnalare al programma quando il byte e' stato completamente trasmesso e il buffer di trasmissione e' di nuovo pronto per inviare un altro byte.

Il bit **RI** "Receive Interrupt" e' simile al bit TI ma indica che un byte e' stato ricevuto dalla porta seriale. In questo caso il programma applicativo deve leggere alla svelta il valore da SBUF prima che un altro byte possa essere ricevuto.

Inizializzare il Baud Rate della porta seriale

Una volta che la porta seriale e' stata configurata, bisogna impostare il baud rate. Cio' e' applicabile soltanto ai modi 1 e 3 poiche' nei modi 0 e 2 il baud rate e' fisso ed e' legato alla frequenza di oscillazione del quarzo del micro. Nel modo 0, il baud rate e' sempre ricavato dividendo la frequenza di clock principale per 12. Questo significa che se usate un quarzo a 11,059 MHz il baud rate sara' pari a 921.583 baud. Nel modo 2

invece la il baud rate e' sempre ricavato dividendo la frequenza di clock principale per 64, e quindi se usiamo un quarzo come quello dell'esempio precedente il baud rate sara' pari a 172.797.

Nei modi 1 e 3, il baud rate dipende dalla frequenza con la quale il timer 1 raggiunge l'overflow. Piu' spesso il timer 1 va in overflow, maggiore sara' il baud rate. Esistono varie maniere per fare cio', ma la piu' comune e' quella di configurare il timer 1 a 8-bit con auto-reload (modo 2) ed impostare il valore di reload in TH1. In questo modo il timer 1 andra' periodicamente in overflow generando il baud rate.

Per determinare il valore che deve essere scritto in TH1 per un dato baud rate, e' possibile usare la seguente equazione (assumiamo che PCON.7 sia impostato a zero):

$$TH1 = 256 - ((Fq / 384) / Baud) - Fq = \text{Frequenza del quarzo}$$

Se PCON.7 e' settato, il baud rate vale il doppio, per cui va usata quest'altra equazione:

$$TH1 = 256 - ((Fq / 192) / Baud)$$

Per esempio, se abbiamo un quarzo che oscilla a 11,059 MHz e vogliamo impostare il baud rate a 19.200:

$$\begin{aligned} TH1 &= 256 - ((Fq / 384) / Baud) \\ TH1 &= 256 - ((11059000 / 384) / 19200) \\ TH1 &= 256 - ((28,799) / 19200) \\ TH1 &= 256 - 1.5 = 254.5 \end{aligned}$$

Come possiamo notare, per ottenere 19.200 baud dovremmo impostare TH1 a 254.5. Se lo impostiamo a 254 avremo 14.400 baud e se lo impostiamo a 255 avremo 28.800.

Come fare?

Tranquilli, per impostare 19.200 baud, imposteremo semplicemente PCON.7 (SMOD) a 1. Questo ci raddoppiera' il baud rate e utilizzando la seconda equazione avremo:

$$\begin{aligned} TH1 &= 256 - ((Fq / 192) / Baud) \\ TH1 &= 256 - ((11059000 / 192) / 19200) \\ TH1 &= 256 - ((57699) / 19200) \\ TH1 &= 256 - 3 = 253 \end{aligned}$$

Adesso possiamo usare 253 che e' un numero intero.

Scrivere un dato sulla porta seriale

Configurata la porta seriale, siamo ora in grado di ricevere e trasmettere dei dati. Se ritenete che la configurazione della porta seriale sia semplice, l'uso della stessa e' banale. Per scrivere un byte sulla porta seriale sara' sufficiente scriverlo sul registro **SBUF** (99h).

Per inviare la lettera "A" useremo la seguente istruzione:

```
MOV SBUF,#?A?
```

Dopo l'esecuzione dell'istruzione precedente, l'8051 comincera' a trasmettere il carattere attraverso la porta seriale. Ovviamente la trasmissione non e' istantanea, essa richiede una quantita' di tempo prestabilita. Poiche'

l'8051 non dispone di un buffer di trasmissione aggiuntivo, dobbiamo essere completamente certi che il carattere sia stato inviato prima di tentare l'invio del prossimo.

L'8051 ci segnalera' quando la trasmissione e' stata completata settando il bit TI di SCON. Solo a quel punto saremo in grado di inviarne un altro.

Consideriamo il seguente codice:

```
CLR TI ;Accertati che il bit TI sia a zero
MOV SBUF,#?A? ;Invia il carattere A sulla seriale
JNB TI,$ ;Aspetta che il bit TI sia settato
```

Le tre istruzioni precedenti trasmettono un carattere e attendono che il bit TI sia settato prima di proseguire. Infatti l'ultima istruzione si puo' interpretare nel modo seguente: " Rimani all'interno dell'istruzione stessa finche' TI e' uguale a zero."

Leggere dalla porta seriale

Leggere i dati dalla porta seriale e' semplice come scriverli. E' sufficiente leggere il valore dal registro SBUF quando il flag RI di SCON e' stato attivato.

Il codice seguente vi mostra come questo puo' essere fatto:

```
JNB RI,$ ; Aspetta finche' il bit RI non venga attivato
MOV A,SBUF ; Leggi il dato dalla porta seriale
```

La prima linea del codice attende che l'8051 attivi il flag RI. Cio' viene fatto automaticamente quando il carattere e' completamente ricevuto.

Finche' la condizione e' vera il programma ripete l'istruzione "JNB".

Appena RI si attiva, la condizione diventa falsa e il programma procede con l'istruzione successiva.

[^ INDICE](#)

[< I Timer](#)

[> Interrupt](#)

(C) Copyright 1997, 1998 by Vault Information Services. All Rights Reserved.

Le informazioni sono fornite senza alcuna garanzia.

Traduzione italiana di Sergio Salvitti.

Conversione in PDF di Farmi Roberto.

CAPITOLO 9– Interrupt

Come implicito nel nome stesso, un **interrupt** e' un evento che interrompe la normale esecuzione di un programma.

Come gia' precedentemente affermato, il flusso di un programma e sempre di tipo sequenziale e puo' essere alterato da particolari istruzioni che volutamente intendono deviare il flusso del programma stesso. Gli interrupt forniscono invece un meccanismo di "congelamento" del flusso del programma in corso, eseguono una subroutine (sottoprogramma) e ripristinano il normale funzionamento del programma come se nulla fosse accaduto. Questa subroutine, denominate "interrupt handler" (gestore di interrupt) viene eseguita soltanto quando si verifica un determinato evento che attiva l'interrupt. L'evento puo' essere: il timer che va in overflow, la ricezione di un byte dalla seriale, la fine della trasmissione di un byte dalla seriale e uno o due eventi esterni. L'8051 puo' essere configurato per gestire un interrupt handler per ognuno di questi eventi.

La capacita' di interrompere la normale esecuzione di un programma quando un particolare evento si verifica, rende il programma stesso molto piu' efficiente nel gestire processi asincroni. Qualora non disponessimo degli interrupt e volessimo gestire piu' eventi dal programma principale, saremmo costretti a scrivere un codice poco elegante e difficile da capire. Allo stesso tempo il nostro programma, in particolari condizioni, diventerebbe inefficiente, dal momento che perderemmo del tempo prezioso ad aspettare un evento che non avviene molto frequentemente.

Per esempio, supponiamo di avere una grande memoria programma di 16k che esegue molte subroutine per eseguire molti lavori. Supponiamo inoltre che il nostro programma debba automaticamente far cambiare lo stato del pin P3.0 ogni volta che il timer 0 va in overflow.

Il codice per effettuare tale azione non e' tanto difficile:

```
JNB TF0,SKIP_TOGGLE  
CPL P3.0  
CLR TF0  
SKIP_TOGGLE: ...
```

Poiche' il flag TF0 e' settato ogni volta che il timer 0 va in overflow, il codice di sopra fara' cambiare lo stato di P3.0 di conseguenza. Questo persegue il nostro scopo ma e' inefficiente. L'istruzione JNB consuma 2 cicli d'istruzione per determinare che il flag non e' attivo e saltare il codice non necessario. Quando il timer va in overflow le due istruzioni CPL e CLR richiedono due cicli d'istruzione per essere eseguite. Per semplificare i calcoli ammettiamo che il resto del codice nel programma richieda 98 cicli d'istruzione. Quindi, in totale il nostro codice consuma 100 cicli d'istruzione. Se il timer e' configurato in modo 16-bit, esso andra' in overflow ogni 65536 cicli macchina. In tutto questo tempo dovremmo aver eseguito 655 JNB test per un totale di 1310 cicli d'istruzione ed altri 2 per eseguire il vero codice utile. Per ottenere il nostro scopo abbiamo speso il 2,002% del nostro tempo per controllare quando cambiare lo stato di P3.0. Il nostro codice e' brutto perche' dovremmo ripetere questa operazione di controllo per ogni ciclo del loop del programma principale.

Per fortuna, questo non serve. Gli interrupt ci permettono di dimenticarci di controllare la condizione che scatena l'interrupt stesso. Il microcontrollore eseguirà il controllo automaticamente e quando la condizione sarà vera, richiamerà la subroutine (chiamata interrupt handler), eseguirà il codice in esso contenuta e ritornerà. In questo caso la nostra subroutine dovrebbe ridursi a:

CPL P3.0 RETI

Prima di tutto va notato che l'istruzione CLR TF0 è sparita. Questo perché, quando l'8051 esegue la routine dell'interrupt del timer 0, cancella automaticamente il flag TF0. Avrete inoltre notato che invece della normale istruzione RET abbiamo usato l'istruzione RETI. Quest'ultima fa le stesse cose della precedente ma informa l'8051 che la routine di interrupt è finita.

Dovete sempre terminare un interrupt handler con RETI.

Tornando all'esempio precedente, ogni 65536 cicli d'istruzione eseguiremo le istruzioni CPL e RETI che richiedono solo 3 cicli. Come risultato finale il nostro codice è 437 volte più efficiente del precedente senza contare che è più facile da leggere e da capire. Dobbiamo solo predisporre l'interrupt e dimenticarci, fiduciosi che l'8051 eseguirà il nostro codice quando serve.

Lo stesso concetto si applica alla ricezione dei dati dalla porta seriale. Una possibilità è quella di controllare continuamente lo stato del flag RI in un loop infinito. Oppure noi potremmo controllare il flag all'interno del più grande loop del programma principale. Nel peggiore dei casi potremmo correre il rischio di *perdere* il carattere ricevuto. Se un carattere viene ricevuto subito dopo che è stato effettuato il controllo del flag RI, nella prosecuzione del resto del programma, prima di controllare di nuovo RI, un nuovo carattere potrebbe arrivare e noi avremo perso il precedente. Usando l'interrupt, l'8051 ferma il programma e richiama l'interrupt handler per servire la ricezione del carattere. Così non saremo costretti a scrivere un antiestetico controllo nel nostro codice e nemmeno correremo il rischio di perdere i caratteri ricevuti.

Quali eventi possono attivare un interrupt

Possiamo configurare l'8051 affinché ognuno di questi eventi possa causare un interrupt:

- Overflow del Timer 0
- Overflow del Timer 1
- Ricezione/trasmissione di un carattere dalla seriale
- Evento esterno 0 (INT 0).
- Evento esterno 1 (INT 1).

Ovviamente è necessario poter distinguere tra vari interrupt ed eseguire dei codici differenti per ognuno di loro. Quando l'interrupt viene attivato, il microcontrollore salterà a degli indirizzi di memoria predefiniti come mostrato nella tabella che segue:

Interrupt	Flag	Indirizzo dell'Interrupt Handler
INT 0	IE0	0003h
Timer 0	TF0	000Bh
INT 1	IE1	0013h
Timer 1	TF1	001Bh

Seriale	RI/TI	0023h
---------	-------	-------

Se consultiamo la tabella, possiamo affermare che, se il timer 0 va in overflow, il programma principale sospendera' momentaneamente il programma e saltera' all'indirizzo 0003H dove avremmo dovuto scrivere l'interrupt handler opportuno.

Configurare gli interrupt

Al power up, per default, tutti gli interrupt sono disabilitati. Questo significa che, anche se per esempio il flag TF0 e' attivo, l'interrupt non verra' accettato. Il vostro programma deve specificatamente dire all'8051 che intende abilitare la gestione degli interrupt e indicare quali interrupt sono abilitati ad essere serviti.

Il vostro programma puo' abilitare o disabilitare gli interrupt mediante il registro IE (A8h):

Bit	Nome	Indirizzo a Bit	Funzione
7	EA	AFh	Abilita globalmente gli inetrupt
6	-	A Eh	Non definito
5	-	ADh	Non definito
4	ES	ACh	Abilita l'interrupt della seriale
3	ET1	ABh	Abilita l'interrupt del Timer 1
2	EX1	AAh	Abilita l'interrupt esterno INT 1
1	ET0	A9h	Abilita l'interrupt del Timer 0
0	EX0	A8h	Abilita l'interrupt esterno INT 0

Come potete notare, ciascun interrupt dell'8051 dispone si un suo bit nel registro IE. Potete abilitare un determinato interrupt, settando il corrispondente bit. Per esempio, se desiderate abilitare l'interrupt del Timer 1, potreste usare ambedue le seguenti istruzioni:

MOV IE,#08h oppure
SETB ET1

Ambedue le istruzioni settano il bit 3 di IE, e abilitano l'interrupt del Timer 1. Affinche' l'interrupt del Timer 1 (e questo vale anche per gli altri) sia effettivamente abilitato, e' necessario settare anche il bit 7 di IE. Se tale bit rimane a zero, nessun interrupt anche abilitato sara' attivo. Porre ad uno il bit 7 di IE abilita *globalmente* tutti gli interrupt i cui bit sono allo stato on.

Questo bit e' utile durante l'esecuzione di un programma che abbia una porzione di codice con criticita' temporali. Allora, per evitare che quella parte di codice possa essere interrotta da un qualsiasi interrupt, sara' sufficiente resettare il bit 7 di IE e settarlo di nuovo quando la sezione critica e' terminata.

Sequenza di Polling

Durante ogni istruzione, l'8051 verifica che non vi sia un interrupt da servire. Durante tale controllo, esso segue il seguente ordine:

- Interrupt esterno INT 0
- Interrupt Timer 0
- Interrupt esterno INT 1
- Interrupt Timer 1
- Interrupt della seriale

Cio' significa che, se ul'interrupt della seriale si attiva nello stesso momento doi quello esterno INT 0, quest'ultimo verra' servito per primo e quello relativo alla porta seriale immediatamente dopo.

Priorita' degli Interrupt

L'8051 dispone di due livelli di priorita' degli interrupt: **alto e basso**. Usando tali priorita' potete cambiare l'ordine con il quale gli interrupt vengono serviti.

Per esempio, se avete abilitato sia l'interrupt del Timer 1 che quello della porta seriale e ritenete che la ricezione di un carattere sia molto piu' importante della gestione del timer, potreste assegnare alla seriale un *priorita' alta*, in maniera da cambiare l'ordine standard con il quale il micro serve normalmente gli interrupt.

La priorita' degli interrupt e' controllata dal registro **IP** (B8h).

Esso ha il seguente formato:

Bit	Nome	Indirizzo a Bit	Funzione
7	–	–	Non definito
6	–	–	Non definito
5	–	–	Non definito
4	PS	BCh	Priorita' Interrupt Seriale
3	PT1	BBh	Priorita' Interrupt Timer 1
2	PX1	BAh	Priorita' Interrupt esterno INT 1
1	PT0	B9h	Priorita' Interrupt Timer 0
	PX0	B8h	Priorita' Interrupt esterno INT 0

Quando consideriamo la priorita' degli interrupt, vanno applicate le seguenti regole:

- Nulla puo' interrompere un interrupt ad alta priorita', nemmeno un altro dello stesso tipo.
- Un interrupt ad alta priorita' puo' invece interromperne uno a bassa priorita'.
- Un interrupt a bassa priorita' puo' essere iniziato soltanto quando nessun altro interrupt e' attivo.
- Quando si verificano due richieste contemporanee di interrupt, quello a priorita' piu' alta viene servito per primo. Se ambedue gli interrupt hanno la medesima priorita' allora viene scelto secondo l'ordine della sequenza di [polling](#).

Cosa succede quanto si verifica una richiesta d'interrupt?

Al momento che un interrupt viene richiesto, il microcontrollore automaticamente esegue le seguenti

operazioni:

- Il valore del Program Counter viene salvato nello stack, a partire dal byte meno significativo.
- Tutti gli interrupt della medesima o piu' bassa priorita' sono bloccati.
- Se l'interrupt riguarda o un timer o una richiesta esterna, viene disattivato il flag corrispondente.
- L'esecuzione del programma salta all'indirizzo dell'interrupt handler corrispondente

Una speciale attenzione e' richiesta circa il terzo passo nel quale il flag viene automaticamente cancellato. Cio' significa che non e' necessario farlo all'interno del codice.

Che succede quando un interrupt finisce?

Un interrupt termina quando viene eseguita l'istruzione RETI (Return from Interrupt). A quel punto il microcontrollore esegue i seguenti passi:

- Due byte vengono prelevati dallo stack (operazione di pop) e messi nel Program Counter per tornare al programma principale.
- Lo stato dell' interrupt e' rimesso nella condizione precedente all'inizio dell'interrupt stesso

Interrupt seriali

Gli interrupt seriali sono leggermente diversi da tutti gli altri. Cio' e' dovuto al fatto che ci sono due flag di interrupt: **RI** e **TI**. Se uno dei due e' attivo allora anche l'interrupt della seriale diventera' attivo. Questo vuol dire che, al momento di una richiesta di interrupt sulla seriale, non conosciamo quali dei due o tutti e due i flag sono attivi. Allora, l'interrupt handler deve verificare lo stato dei flag e comportarsi di conseguenza. Inoltre deve anche cancellare i flag poiche' l'8051 volutamente non lo fa in maniera automatica.

Un breve esempio di codice vi chiarira' il tutto:

```
INT_SERIAL: JNB RI,CHECK_TI ; Se il flag RI non e' attivo vai a CHECK_TI
               MOV A,SBUF      ; leggi il buffer di ricezione
               CLR RI          ; Cancella il flag RI
CHECK_TI:   JNB TI,EXIT_INT ; Se il flag TI non e' attivo vai a EXIT_TI
               CLR TI          ; Cancella il flag TI prima di inviare un nuovo carattere
               MOV SBUF,#?A? ; Copia il nuovo carattere da inviare nel buffer di trasmissione.
EXIT_INT:   RETI
```

Una Importante Considerazione sugli interrupt: Preservare il valore di un Registro

Una regola importante deve essere applicata da tutti gli interrupt handler: Ogni interrupt deve lasciare il processore nel medesimo stato che esso aveva al momento di iniziare l'interrupt stesso.

Il programma principale non tiene conto del fatto che i vari interrupt sono eseguiti di *nascosto*.

Prendiamo in considerazione la seguente porzione di codice:

```
CLR C ; Cancella il carry
```

MOV A,#25h ; Carica 25h nell'accumulatore
ADDC A,#10h ; Aggiungi 10h, tenendo conto del carry

Al termine dell'esecuzione di queste istruzioni, l'accumulatore conterra' il valore 35h.

Ma cosa accadrebbe se appena conclusa l'istruzione MOV viene servito un interrupt e questo cambia sia il carry che il valore dell'accumulatore ponendolo ad uno? Quando l'interrupt restituisce il controllo al programma principale, l'istruzione ADDC addiziona' 10h a 40h e aggiungera' 1h perche' trovera' il bit di carry settato. In questo caso l' accumulatore conterra' il valore 51h invece di 35h.

Quello che e' successo nella relata', e' che l'interrupt handler non ha preservato il valore del registro che ha usato. La regola pero' dice : ***Ogni interrupt deve lasciare il processore nel medesimo stato che esso aveva al momento di iniziare l'interrupt stesso.***

Cosa vuol dire l'affermazione precedente? Vuol dire che, se l'interrupt usa l'accumulatore, deve assicurare che il valore di esso rimanga inalterato; cio' viene ottenuto con delle operazioni di PUSH e POP. Per esempio:

```
PUSH ACC  
PUSH PSW  
MOV A,#0FFh  
ADD A,#02h  
POP PSW  
POP ACC  
RETI
```

Le istruzioni dell'interrupt handler sono MOV e ADD che modificano sia l'accumulatore che il bit di carry; allora e' necessario salvare nello stack sia l'accumulatore che il registro PSW con due operazioni di PUSH. Una volta eseguite le due istruzioni proprie della routine di interrupt dovranno essere ripristinati i valori dei due registri con due operazioni di POP (Attenzione all'ordine inverso con il quale devono essere eseguite). A questo punto e' possibile terminare l'interrupt con l'istruzione RETI.

Il programma principale trovera' la situazione dei suoi registri immutata e quindi non si accorgera' minimamente dell' interruzione.

In generale, la routine di interrupt deve preservare il contenuto dei seguenti registri:

- PSW
- DPTR (DPH/DPL)
- PSW
- ACC
- B
- Registers R0–R7

In particolare fate attenzione al registro PSW, che contiene i flag di stato del processore. E' buona norma salvare **sempre** il contenuto di PSW mediante le operazioni di push e pop all'inizio e alla fine della routine d'interrupt, *a meno che non siate assolutamente sicuri* di non modificare alcun bit di PSW.

Notate che l'assembler non vi permette di eseguire la seguente istruzione:

```
PUSH R0
```

Questo e' dovuto al fatto che l'indirizzo di R0 dipende dal banco di registri "R" selezionato e potrebbe riferirsi alla ram interna in posizione 00h o 08h o 10h oppure 18h.

Percio', se state usando un registro "R" all'interno della routine di interrupt, effettuate l'operazione di push facendo riferimento all'indirizzo assoluto del registro stesso.

Per esempio, invece di usare PUSH R0, dovrete usare:

PUSH 00h

Naturalmente l'istruzione e' corretta solo se state utilizzando il set di registri di default, altrimenti dovete fare il PUSH dell'indirizzo assoluto corrispondente al registro che effettivamente state usando.

Problemi Comuni con l'uso degli Interrupt

Gli interrupt sono degli strumenti molto potenti, ma se usati in maniera non corretta, sono una fonte di ore spese a trovare problemi nel programma.

Gli errori sulle routine di interrupt sono spesso molto difficili da diagnosticare e correggere.

Se state utilizzando degli interrupt e il vostro programma va in crash oppure ha dei comportamenti strani con risultati randomici, ricordatevi di verificare che le regole appena esposte non siano state violate.

In generale i problemi piu' comuni derivano da:

- **Non aver preservato il valore di un registro che viene usato nella routine di interrupt.**
Controllate che tutti i registri che usate siano preventivamente salvati nello stack con un'operazione di push
- **Non aver ripristinato il valore del registro prima dell'uscita dalla routine dell'interrupt.**
Controllate di aver effettuato lo stesso numero di operazioni di push e di pop in ordine inverso
- **Aver usato l'istruzione RET invece di RETI oppure non aver usato alcuna istruzione di return.**
In questo caso l'interrupt non termina e quindi verra' eseguito una volta soltanto e poi misteriosamente sembrera' come se esso fosse bloccato.
- **Non aver chiuso la routine di interrupt con l'istruzione RETI.**

Alcuni simulatori dell'8051 come : [Vault Information Services– 8052 Simulator for Windows](#), hanno delle speciali caratteristiche che vi notificano il fatto che avete dimenticato di preservare il valore di un registro oppure avete commesso un qualche errore riguardante l'uso improprio della routine di inerrupt.

^ [INDICE](#)

< [Porta Seriale](#)

> [Set d'istruzioni 8051](#)

(C) Copyright 1997, 1998 by Vault Information Services. All Rights Reserved.

Le informazioni sono fornite senza alcuna garanzia.

Traduzione italiana di Sergio Salvitti.

Conversione in PDF di Farmi Roberto.

Corso 8051

Set d'Istruzioni

NOTAZIONI

- **A** – Accumulatore (Registro "A")
 - **AB** – Tra l'Accumulatore ed il registro "B"
 - **DPTR** – Data Pointer
 - **C** – Carri bit
 - **Rn** – Registro (R7–R0)del banco di registri "R" attualmente selezionato
 - **@Ri** – Indirizzamento indiretto tramite un registro R (R0 o R1) nella RAM interna da 0 a 255.
 - **@DPTR** – Indirizzamento diretto tramite DPTR
 - **@A+DPTR** – Indirizzamento indiretto tramite la somma di DPTR piu' "A"
 - **@A+PC** – Indirizzamento indiretto tramite la somma di PC piu' "A"
 - **iram addr** – Indirizzo della RAM interna
 - **#data** – Indirizzamento immediato
 - **bit addr** –indirizzo del bit
 - **/bit addr** – indirizzo del bit (prendi il bit negato)
 - **reladdr** – Indirizzo relativo (da +128 a –127)
 - **page i** – pagina i (0 –7)
 - **code addr** – Indirizzo di programma
-

ISTRUZIONI ARITMETICHE

- **ADD, ADDC**: Somma l'Accumulatore (con il riporto)
 - **DA**: Aggiusta la parte decimale
 - **DEC**: Decrementa il registro
 - **DIV**: Dividi l'Accumulatore per il registro B
 - **INC**: Incrementa il registro
 - **MUL**: Moltiplica l'Accumulatore per il registro B
 - **SUBB**: Sottrai dall'Accumulatore con il prestito
-

ISTRUZIONI LOGICHE

- **ANL**: AND logico
- **CLR**: Azzera il registro

- **CPL**: Complementa il registro
 - **ORL**: OR logico
 - **RL**: Ruota l'accumulatore a sinistra
 - **RLC**: Ruota l'Accumulatore a sinistra attraverso il Carry
 - **RR**: Ruota l'accumulatore a destra
 - **RRC**: Ruota l'Accumulatore a destra attraverso il Carry
 - **SWAP**: Scambia i nibble dell'Accumulatore
 - **XRL**: OR esclusivo logico
-

ISTRUZIONI DI TRASFERIMENTO DATI

- **MOV**: Trasferisci un byte
 - **MOVC**: Trasferisci un byte della memoria programma
 - **MOVX**: Trasferisci un byte della memoria estesa
 - **POP**: Prendi l'Accumulatore dallo stack
 - **PUSH**: Metti l'accumulatore nello stack
 - **XCH**: Scambia i byte
 - **XCHD**: Scambia i digit
-

ISTRUZIONI SU VARIABILI A BIT

- **ANL**: AND per variabili a bit
 - **CLR**: Azzera il bit
 - **CPL**: Complementa il bit
 - **JB**: Salta se il bit e' a uno
 - **JBC**: Salta se il bit e' a uno e resettalo
 - **JC**: Salta se il Carry e' a uno
 - **JNB**: Salta se il bit non e' a uno
 - **JNC**: Salta se il Carry non e' a uno
 - **MOV**: Trasferisci un bit
 - **ORL**: OR a bit
 - **SETB**: Poni il bit a uno
-

ISTRUZIONI DI SALTO

- **ACALL**: Chiamata a subroutine con indirizzo assoluto
- **AJMP**: Salto assoluto
- **CJNE**: Compara e salta se non uguale
- **DJNZ**: Decrementa il registro e salta se il risultato non e' nullo
- **JMP**: Salta all'indirizzo
- **JNZ**: Salta se l'Accumulatore non e' nullo
- **JZ**: Salta se l'Accumulatore e' nullo

- **LCALL**: Chiamata a subroutine con indirizzo a 16-bit
 - **LJMP**: Salto con indirizzo a 16-bit
 - **NOP**: Nessuna operazione
 - **RET**: Torna da subroutine
 - **RETI**: Torna da interrupt
 - **SJMP**: Salto con indirizzo relativo a 8-bit
-

ELENCO ALFABETICO DELLE ISTRUZIONI – 8051

- **ACALL**: Chiamata a subroutine con indirizzo assoluto
- **ADD, ADDC**: Somma l'Accumulatore (con il riporto)
- **AJMP**: Salto assoluto
- **ANL**: AND logico
- **ANL**: AND per variabili a bit
- **CJNE**: Compara e salta se non uguale
- **CLR**: Azzera il registro
- **CLR**: Azzera il bit
- **CPL**: Complementa il registro
- **CPL**: Complementa il bit
- **DA**: Aggiusta la parte decimale
- **DEC**: Decrementa il registro
- **DIV**: Dividi l'Accumulatore per il registro B
- **DJNZ**: Decrementa il registro e salta se il risultato non e' nullo
- **INC**: Incrementa il registro
- **JB**: Salta se il bit e' a uno
- **JBC**: Salta se il bit e' a uno e resettalo
- **JC**: Salta se il Carry e' a uno
- **JMP**: Salta all'indirizzo
- **JNB**: Salta se il bit non e' a uno
- **JNC**: Salta se il Carry non e' a uno
- **JNZ**: Salta se l'Accumulatore non e' nullo
- **JZ**: Salta se l'Accumulatore e' nullo
- **LCALL**: Chiamata a subroutine con indirizzo a 16-bit
- **LJMP**: Salto con indirizzo a 16-bit
- **MOV**: Trasferisci un byte
- **MOV**: Trasferisci un bit
- **MOVC**: Trasferisci un byte della memoria programma
- **MOVB**: Trasferisci un byte della memoria estesa
- **MUL**: Moltiplica l'Accumulatore per il registro B
- **NOP**: Nessuna operazione
- **ORL**: OR logico
- **ORL**: OR a bit
- **POP**: Prendi l'Accumulatore dallo stack
- **PUSH**: Metti l'accumulatore nello stack
- **RET**: Torna da subroutine
- **RETI**: Torna da interrupt
- **RL**: Ruota l'accumulatore a sinistra

- [RLC](#): Ruota l'Accumulatore a sinistra attraverso il Carry
 - [RR](#): Ruota l'accumulatore a destra
 - [RRC](#): Ruota l'Accumulatore a destra attraverso il Carry
 - [SETB](#): Poni il bit ad uno
 - [SJMP](#): Salto con indirizzo relativo a 8-bit
 - [SUBB](#): Sottrai dall'Accumulatore con il prestito
 - [SWAP](#): Scambia i nibble dell'Accumulatore
 - [XCH](#): Scambia i byte
 - [XCHD](#): Scambia i digit
 - [XRL](#): OR esclusivo logico
-

[^] [INDICE](#)

(C) Copyright 1997, 1998 by Vault Information Services. All Rights Reserved.

Le informazioni sono fornite senza alcuna garanzia.

Traduzione italiana di Sergio Salvitti.

Conversione in PDF di Farmi Roberto.

ISTRUZIONI MATEMATICHE

ADD, ADDC

Istruzione: ADD, ADDC

Funzione: Somma l'Accumulatore, Somma l'Accumulatore con riporto

Sintassi: ADD A,*operando*
ADDC A,*operando*

Istruzione	OpCode	N.Byte	N.Cicli	Flag
ADD A,#data	0x24	2	1	C, AC, OV
ADD A,iram addr	0x25	2	1	C, AC, OV
ADD A,@R0	0x26	1	1	C, AC, OV
ADD A,@R1	0x27	1	1	C, AC, OV
ADD A,R0	0x28	1	1	C, AC, OV
ADD A,R1	0x29	1	1	C, AC, OV
ADD A,R2	0x2A	1	1	C, AC, OV
ADD A,R3	0x2B	1	1	C, AC, OV
ADD A,R4	0x2C	1	1	C, AC, OV
ADD A,R5	0x2D	1	1	C, AC, OV
ADD A,R6	0x2E	1	1	C, AC, OV
ADD A,R7	0x2F	1	1	C, AC, OV

Istruzione	OpCode	N.Byte	N.Cicli	Flag
ADDC A,#data	0x34	2	1	C, AC, OV
ADDC A,iram addr	0x35	2	1	C, AC, OV
ADDC A,@R0	0x36	1	1	C, AC, OV
ADDC A,@R1	0x37	1	1	C, AC, OV
ADDC A,R0	0x38	1	1	C, AC, OV
ADDC A,R1	0x39	1	1	C, AC, OV
ADDC A,R2	0x3A	1	1	C, AC, OV
ADDC A,R3	0x3B	1	1	C, AC, OV
ADDC A,R4	0x3C	1	1	C, AC, OV
ADDC A,R5	0x3D	1	1	C, AC, OV
ADDC A,R6	0x3E	1	1	C, AC, OV
ADDC A,R7	0x3F	1	1	C, AC, OV

Descrizione: ADD e ADC sommano entrambi il valore operando all'Accumulatore e mettono il risultato nell'Accumulatore stesso. Il valore dell'operando non viene modificato. ADD e ADC funzionano nella stessa maniera eccetto il fatto che ADC tiene conto anche del riporto (Carry Flag).

Il bit **Carry (C)** e' settato se c'e' un riporto del bit 7. In altre parole, se il valore della somma senza segno tra l'Accumulatore e l'operando (e anche di C nel caso di ADC) supera il valore di 255 il bit C e' settato altrimenti resettato.

Il bit **Auxillary Carry (AC)** e' settato se c'e' un riporto del bit 3. In altre parole, se il valore della somma senza segno tra l'Accumulatore e l'operando (e anche di C nel caso di ADC) supera il valore di 15 il bit AC e' settato altrimenti resettato.

Il bit **Overflow (OV)** e' settato se il bit 6 o il bit 7 hanno il riporto, ma non entrambi. In altri termini, se se il valore della somma con segno tra l'Accumulatore e l'operando (e anche di C nel caso di ADC) va fuori del range (da -128 a +127) il bit OV e' settato altrimenti resettato.

Vedi anche: [SUBB](#), [DA](#), [INC](#), [DEC](#), [Set d'istruzione](#).

DA

Istruzione: DA

Funzione: Aggiusta il valore Decimale dell'Accumulatore

Sintassi: DA A

Istruzione	OpCode	N.Byte	N.Cicli	Flag
DA	0xD4	1	1	C

Descrizione: DA aggiusta il contenuto dell'Accumulatore al corrispondente numero BCD (Binary Coded Decimal) dopo che due numeri BCD sono stati addizionati con ADD o ADDC. Se il bit C e' settato o se il valore del nibble meno significativo supera il valore 9, 0x06 viene aggiunto all'Accumulatore. Se il bit C era gia' da uno prima dell' inizio dell'istruzione oppure 0x06 era gia' stato addizionato nella prima fase, 0x60 viene aggiunto all'Accumulatore.

Il bit **Carry(C)** e' settato se il risultato finale supera 0x99, altrimenti e' resettato.

Vedi anche: [ADD](#), [ADDC](#), [Set d'istruzioni](#)

DEC

Istruzione: DEC

Funzione: Decrementa il Registro

Sintassi: DEC *registro*

Istruzione	OpCode	N.Byte	N.Cicli	Flag
DEC A	0x14	1	1	Inv
DEC <i>iram addr</i>	0x15	2	1	Inv
DEC @R0	0x16	1	1	Inv
DEC @R1	0x17	1	1	Inv
DEC R0	0x18	1	1	Inv

DEC R1	0x19	1	1	Inv
DEC R2	0x1A	1	1	Inv
DEC R3	0x1B	1	1	Inv
DEC R4	0x1C	1	1	Inv
DEC R5	0x1D	1	1	Inv
DEC R6	0x1E	1	1	Inv
DEC R7	0x1F	1	1	Inv

Descrizione: DEC decrementa il valore del *registro* di 1. Se il valore iniziale del *registro* e' zero, esso sara' portato a 255 (0xFF esadecimale). Nota: Il bit C **non** viene settato nel passaggio da 0 a 255 (rolls over).

Vedi anche: [INC](#), [SUBB](#), [Set d'istruzioni](#)

DIV

Istruzione: DIV

Funzione: Divide l'Accumulatore per B

Sintassi: DIV AB

Istruzione	OpCode	N.Byte	N.Cicli	Flag
DIV AB	0x84	1	1	C, OV

Descrizione: Divide il valore senza segno dell'Accumulatore per il valore senza segno del registro "B". Il quoziente della divisione viene posto nell'Accumulatore ed il resto in "B".

Il bit **Carry (C)** e' sempre azzerato.

Il bit **Overflow (OV)** e' settato se viene tentata una divisione per zero, altrimenti e' resettato.

Vedi anche: [MUL AB](#), [Instruction Set](#)

INC

Istruzione: INC

Funzione: Incrementa il Registro

Sintassi: INC *registro*

Istruzione	OpCode	N.Byte	N.Cicli	Flag
INC A	0x04	1	1	Inv
INC <i>iram addr</i>	0x05	2	1	Inv
INC @R0	0x06	1	1	Inv
INC @R1	0x07	1	1	Inv
INC R0	0x08	1	1	Inv
INC R1	0x09	1	1	Inv
INC R2	0x0A	1	1	Inv
INC R3	0x0B	1	1	Inv
INC R4	0x0C	1	1	Inv
INC R5	0x0D	1	1	Inv
INC R6	0x0E	1	1	Inv
INC R7	0x0F	1	1	Inv
INC DPTR	0xA3	1	2	Inv

Descrizione: INC incrementa il valore del registro di 1. Se il valore iniziale del registro e' pari a 255 (0xFF esadecimale) tale incremento lo portera' a zero. Nota: il bit C non viene settato dal passaggio da 255 a 0 (rolls over).

Nel caso di "INC DPTR", viene incrementato il valore a 2 byte di DPTR come un intero senza segno. Se il valore iniziale di DPTR e' 65535 (0xFFFF esadecimale) l'incremento portera' DPTR a zero. Anche in questo

caso il bit C non viene settato.

Vedi anche: [ADD](#), [ADDC](#), [DEC](#), [Set d'istruzioni](#)

MUL

Istruzione: MUL

Funzione: Moltiplica l'Accumulatore per B

Sintassi: MUL AB

Istruzione	OpCode	N.Byte	N.Cicli	Flag
MUL AB	0xA4	1	4	C, OV

Descrizione: Moltiplica il valore senza segno dell'Accumulatore per il valore senza segno del registro "B". Il byte piu' significativo del risultato e' posto nell'Accumulatore e quello meno significativo in B.

Il bit **Carry (C)** e sempre azzerato.

Il bit **Overflow (OV)** e' settato se il risultato dell'operazione e' maggiore di 255, altrimenti e' resettato.

Vedi anche: [DIV](#), [Set d'istruzioni](#)

SUBB

Istruzione: SUBB

Funzione: Sottrai dall'Accumulatore con il prestito

Sintassi: SUBB A,*operando*

Istruzione	OpCode	N.Byte	N.Cicli	Flag
SUBB A, <i>#data</i>	0x94	2	1	C, AC, OV

SUBB A, <i>iram addr</i>	0x95	2	1	C, AC, OV
SUBB A,@R0	0x96	1	1	C, AC, OV
SUBB A,@R1	0x97	1	1	C, AC, OV
SUBB A,R0	0x98	1	1	C, AC, OV
SUBB A,R1	0x99	1	1	C, AC, OV
SUBB A,R2	0x9A	1	1	C, AC, OV
SUBB A,R3	0x9B	1	1	C, AC, OV
SUBB A,R4	0x9C	1	1	C, AC, OV
SUBB A,R5	0x9D	1	1	C, AC, OV
SUBB A,R6	0x9E	1	1	C, AC, OV
SUBB A,R7	0x9F	1	1	C, AC, OV

Descrizione: SUBB sottrae il valore dell'operando dal valore dell'Accumulatore, lasciando il risultato nell'Accumulatore stesso. Il valore dell'operando non viene modificato.

Il bit **Carry (C)** viene settato se e' stato richiesto il prestito per il bit 7, altrimenti e' resettato. In altre parole, se il valore da sottrarre e' maggiore dell'Accumulatore il bit C e' settato.

Il bit **Auxillary Carry (AC)** e' settato se il prestito e' richiesto dal bit 3. In altre parole, il bit e' settato se il nibble meno significativo del valore da sottrarre e' stato piu' alto del nibble meno significativo dell'Accumulatore.

Il bit **Overflow (OV)** e' settato se il prestito e' stato richiesto dal bit 6 o dal bit 7, ma non da entrambi. In altri termini, il bit OV e' settato se la sottrazione di due byte con segno ha dato come risultato un valore fuori del range (da -128 a +127), altrimenti e' resettato.

Vedi anche: [ADD](#), [ADDC](#), [DEC](#), [Set d'istruzioni](#)

^ [INDICE](#)
[SET D'ISTRUZIONI](#)

(C) Copyright 1997, 1998 by Vault Information Services. All Rights Reserved.

Le informazioni sono fornite senza alcuna garanzia. Per favore [vedi dettagli](#).

Contattaci per l'uso e/o il permesso di divulgazione di questo corso.

Traduzione italiana di: **Sergio Salvitti**

Corso 8051 – SET D'ISTRUZIONI

ISTRUZIONI LOGICHE

ANL

Istruzione: ANL

Funzione: AND a bit

Sintassi: ANL *Operando 1*, *Operando 2*

Istruzione	OpCode	N.Byte	N.Cicli	Flag
ANL <i>iram addr</i> ,A	0x52	2	1	Inv
ANL <i>iram addr</i> ,# <i>data</i>	0x53	3	2	Inv
ANL A,# <i>data</i>	0x54	2	1	Inv
ANL A, <i>iram addr</i>	0x55	2	1	Inv
ANL A,@R0	0x56	1	1	Inv
ANL A,@R1	0x57	1	1	Inv
ANL A,R0	0x58	1	1	Inv
ANL A,R1	0x59	1	1	Inv
ANL A,R2	0x5A	1	1	Inv
ANL A,R3	0x5B	1	1	Inv
ANL A,R4	0x5C	1	1	Inv
ANL A,R5	0x5D	1	1	Inv

ANL A,R6	0x5E	1	1	Inv
ANL A,R7	0x5F	1	1	Inv

Descrizione: ANL esegue l'operazione di "AND" bit a bit tra l'*operando 1* e l'*operando 2*. Il risultato viene memorizzato nell'*operando 1*. Il valore dell'*operando 2* non viene modificato. L'AND logico compara ogni bit del primo operando con il bit corrispondente del secondo e viene settato solo e solo se entrambi i bit erano ad uno, altrimenti viene resettato.

Vedi anche: [ORL](#), [XRL](#), [Set d'istruzioni](#)

CLR

Istruzione: CLR

Funzione: Pone a zero il Registro

Sintassi: CLR A

Istruzione	OpCode	N.Byte	N.Cicli	Flag
CLR A	0xE4	1	1	Inv

Descrizione: CLR A pone a zero tutti i bit dell'Accumulatore.

Vedi anche: [Set d'istruzioni](#)

CPL

Istruzione: CPL

Funzione: Complementa il Registro

Sintassi: CPL A

Istruzione	OpCode	N.Byte	N.Cicli	Flag
------------	--------	--------	---------	------

CPL A	0xF4	1	1	Inv
-------	------	---	---	-----

Descrizione: CPL A complementa l'Accumulatore

Vedi anche: [Set d'istruzioni](#)

ORL

Istruzione: ORL

Funzione: OR a bit

Sintassi: ORL *Operando1,Operando2*

Istruzione	OpCode	N.Byte	N.Cicli	Flag
ORL <i>iram addr,A</i>	0x42	2	1	Inv
ORL <i>iram addr,#data</i>	0x43	3	2	Inv
ORL <i>A,#data</i>	0x44	2	1	Inv
ORL <i>A,iram addr</i>	0x45	2	1	Inv
ORL <i>A,@R0</i>	0x46	1	1	Inv
ORL <i>A,@R1</i>	0x47	1	1	Inv
ORL <i>A,R0</i>	0x48	1	1	Inv
ORL <i>A,R1</i>	0x49	1	1	Inv
ORL <i>A,R2</i>	0x4A	1	1	Inv
ORL <i>A,R3</i>	0x4B	1	1	Inv
ORL <i>A,R4</i>	0x4C	1	1	Inv
ORL <i>A,R5</i>	0x4D	1	1	Inv
ORL <i>A,R6</i>	0x4E	1	1	Inv

ORL A,R7	0x4F	1	1	Inv
----------	------	---	---	-----

Descrizione: ORL esegue l'operazione di "OR" bit a bit tra l'*operando 1* e l'*operando 2*. Il risultato viene memorizzato nell'*operando 1*. Il valore dell'*operando 2* non viene modificato. L'OR logico compara ogni bit del primo operando con il bit corrispondente del secondo e viene resettato se tutti e due bit erano ad zero, altrimenti viene settato.

Vedi anche: [ANL](#), [XRL](#), [Set d'istruzioni](#)

RL

Istruzione: RL

Funzione: Ruota l'Accumulatore a sinistra

Sintassi: RL A

Istruzione	OpCode	N.Byte	N.Cicli	Flag
RL A	0x23	1	1	C

Descrizione: Esegue lo shift dei bit dell'Accumulatore a sinistra. Il bit piu' a sinistra (bit 7) dell'Accumulatore e' caricato nel bit piu' a destra (bit 0).

Vedi anche: [RLC](#), [RR](#), [RRC](#), [Set d'istruzioni](#)

RLC

Istruzione: RLC

Funzione: Ruota l'Accumulatore a sinistra attraverso il carry

Sintassi: RLC A

Istruzione	OpCode	N.Byte	N.Cicli	Flag
RLC A	0x33	1	1	C

Descrizione: Esegue lo shift dei bit dell'Accumulatore a sinistra. Il bit piu' a sinistra (bit 7) dell'Accumulatore viene caricato nel Carry Flag e il valore originale del Carry e' caricato nel bit piu' a destra (bit 0). Questa funzione e' usata per effettuare la motiplicazione veloce per due.

Vedi anche: [RL](#), [RR](#), [RRC](#), [Set d'istruzioni](#)

RR

Istruzione: RR

Funzione: Ruota l'Accumulatore a edestra

Sintassi: RR A

Istruzione	OpCode	N.Byte	N.Cicli	Flag
RR A	0x03	1	1	Inv

Descrizione: Esegue lo shift a destra dei bit dell'Accumulatore. Il bit piu' a destra (bit 0) dell'Accumulatore e' caricato nel bit piu' a sinistra (bit 7).

Vedi anche: [RL](#), [RLC](#), [RRC](#), [Set d'istruzioni](#)

RRC

Istruzione: RRC

Funzione: Ruota l'Accumulatore a destra attraverso il Carry

Sintassi: RRC A

Istruzione	OpCode	N.Byte	N.Cicli	Flag
------------	--------	--------	---------	------

RRC A	0x13	1	1	C
-------	------	---	---	---

Descrizione: Esegue lo shift dei bit dell'Accumulatore a destra. Il bit piu' a destra (bit 0) dell'Accumulatore e' caricato nel Carry Flag e il valore originale del Carry e' caricato nel bit piu' a sinistra (bit 7). Questa funzione e' usata per dividere velocemente per due.

Vedi anche: [RL](#), [RLC](#), [RR](#), [Set d'istruzioni](#)

XRL

Istruzione: XRL

Funzione: OR esclusivo a bit

Sintassi: XRL *Operando1,Operando2*

Istruzione	OpCode	N.Byte	N.Cicli	Flag
XRL <i>iram addr,A</i>	0x62	2	1	Inv
XRL <i>iram addr,#data</i>	0x63	3	2	Inv
XRL <i>A,#data</i>	0x64	2	1	Inv
XRL <i>A,iram addr</i>	0x65	2	1	Inv
XRL <i>A,@R0</i>	0x66	1	1	Inv
XRL <i>A,@R1</i>	0x67	1	1	Inv
XRL <i>A,R0</i>	0x68	1	1	Inv
XRL <i>A,R1</i>	0x69	1	1	Inv
XRL <i>A,R2</i>	0x6A	1	1	Inv
XRL <i>A,R3</i>	0x6B	1	1	Inv
XRL <i>A,R4</i>	0x6C	1	1	Inv
XRL <i>A,R5</i>	0x6D	1	1	Inv
XRL <i>A,R6</i>	0x6E	1	1	Inv

XRL A,R7	0x6F	1	1	Inv
----------	------	---	---	-----

Descrizione: XRL esegue l'OR esclusivo "XOR" bit a bit tra l'operando 1 e l'operando 2, lasciando il risultato nell'operando 1. Il valore dell'operando 2 non viene modificato. Lo XOR logico compara i bit dell'operando 1 con i corrispondenti dell'operando 2 e setta il bit corrispondente se essi sono diversi altrimenti lo resetta.

Vedi anche: [ANL](#), [ORL](#), [Set d'istruzioni](#)

[^ INDICE](#)
[SET D'ISTRUZIONI](#)

(C) Copyright 1997, 1998 by Vault Information Services. All Rights Reserved.

Le informazioni sono fornite senza alcuna garanzia. Per favore [vedi dettagli](#).

[Contattaci](#) per l'uso e/o il permesso di divulgazione di questo corso.

Traduzione italiana di: [Sergio Salvitti](#)

ISTRUZIONI DI TRASFERIMENTO DATI

MOV

Istruzione: MOV

Funzione: Trasferisci un byte di Memoria

Sintassi: MOV *Operando1,Operando2*

Istruzione	OpCode	N.Byte	N.Cicli	Flag
MOV @R0,#data	0x76	2	1	Inv
MOV @R1,#data	0x77	2	1	Inv
MOV @R0,A	0xF6	1	1	Inv
MOV @R1,A	0xF7	1	1	Inv
MOV @R0,iram addr	0xA6	2	2	Inv
MOV @R1,iram addr	0xA7	2	2	Inv
MOV A,#data	0x74	2	1	Inv
MOV A,@R0	0xE6	1	1	Inv
MOV A,@R1	0xE7	1	1	Inv
MOV A,R0	0xE8	1	1	Inv
MOV A,R1	0xE9	1	1	Inv
MOV A,R2	0xEA	1	1	Inv
MOV A,R3	0xEB	1	1	Inv

MOV A,R4	0xEC	1	1	Inv
MOV A,R5	0xED	1	1	Inv
MOV A,R6	0xEE	1	1	Inv
MOV A,R7	0xEF	1	1	Inv
MOV A, <i>iram addr</i>	0xE5	2	1	Inv
MOV DPTR, <i>#data16</i>	0x90	3	2	Inv
MOV R0, <i>#data</i>	0x78	2	1	Inv
MOV R1, <i>#data</i>	0x79	2	1	Inv
MOV R2, <i>#data</i>	0x7A	2	1	Inv
MOV R3, <i>#data</i>	0x7B	2	1	Inv
MOV R4, <i>#data</i>	0x7C	2	1	Inv
MOV R5, <i>#data</i>	0x7D	2	1	Inv
MOV R6, <i>#data</i>	0x7E	2	1	Inv
MOV R7, <i>#data</i>	0x7F	2	1	Inv
MOV R0,A	0xF8	1	1	Inv
MOV R1,A	0xF9	1	1	Inv
MOV R2,A	0xFA	1	1	Inv
MOV R3,A	0xFB	1	1	Inv
MOV R4,A	0xFC	1	1	Inv
MOV R5,A	0xFD	1	1	Inv
MOV R6,A	0xFE	1	1	Inv
MOV R7,A	0xFF	1	1	Inv
MOV R0, <i>iram addr</i>	0xA8	2	2	Inv
MOV R1, <i>iram addr</i>	0xA9	2	2	Inv

MOV R2, <i>iram addr</i>	0xAA	2	2	Inv
MOV R3, <i>iram addr</i>	0xAB	2	2	Inv
MOV R4, <i>iram addr</i>	0xAC	2	2	Inv
MOV R5, <i>iram addr</i>	0xAD	2	2	Inv
MOV R6, <i>iram addr</i>	0xAE	2	2	Inv
MOV R7, <i>iram addr</i>	0xAF	2	2	Inv
MOV <i>bit addr</i> ,C	0x92	2	2	Inv
MOV <i>iram addr</i> ,# <i>data</i>	0x75	3	2	Inv
MOV <i>iram addr</i> ,@R0	0x86	2	2	Inv
MOV <i>iram addr</i> ,@R1	0x87	2	2	Inv
MOV <i>iram addr</i> ,R0	0x88	2	2	Inv
MOV <i>iram addr</i> ,R1	0x89	2	2	Inv
MOV <i>iram addr</i> ,R2	0x8A	2	2	Inv
MOV <i>iram addr</i> ,R3	0x8B	2	2	Inv
MOV <i>iram addr</i> ,R4	0x8C	2	2	Inv
MOV <i>iram addr</i> ,R5	0x8D	2	2	Inv
MOV <i>iram addr</i> ,R6	0x8E	2	2	Inv
MOV <i>iram addr</i> ,R7	0x8F	2	2	Inv
MOV <i>iram addr</i> ,A	0xF5	2	1	Inv
MOV <i>iram addr</i> , <i>iram addr</i>	0x85	3	1	Inv

Descrizione: MOV copia il valore dell'Operando 2 nell'Operando 1. Il valore dell'Operando 2 rimane inalterato. Ambedue gli operandi devono risiedere nella RAM interna. Nessun flag viene modificato a meno che l'istruzione non trasferisce il valore nel registro PSW (che contiene esso stesso i flag).

** Nota: Nel caso di "MOV *iram addr*,*iram addr*" il trasferimento viene effettuato all'inverso delle altre operazioni di mov. Per esempio l'istruzione 0x85, 0x20, 0x20 va interpretata come:

"Carica il contenuto della locazione 0x20 nella locazione 0x50" e non il viceversa come si e' portati a presumere.

Vedi anche: [MOVC](#), [MOVX](#), [XCH](#), [XCHD](#), [PUSH](#), [POP](#), [Set d'istruzioni](#)

MOVC

Istruzione: MOVC

Funzione: Trasferisci il byte di codice nell'Accumulatore

Sintassi: MOVC A,@A+*Registro*

Istruzione	OpCode	N.Byte	N.Cicli	Flag
MOVC A,@A+DPTR	0x93	1	2	Inv
MOVC A,@A+PC	0x83	1	1	Inv

Descrizione: MOVC trasferisce un byte di memoria programma nell'Accumulatore. L'indirizzo del byte da trasferire e' calcolato sommando il valore dell'Accumulatore o con DPTR o con il PC (Program Counter). Nel secondo caso il valore del Program Counter viene incrementato di uno prima di essere usato.

Vedi anche: [MOV](#), [MOVX](#), [Set d'istruzioni](#)

MOVX

Istruzione: MOVX

Funzione: Trasferisci i dati alla/dalla memoria esterna (XRAM)

Sintassi: MOVX *Operando1,Operando2*

Istruzione	OpCode	N.Byte	N.Cicli	Flag
------------	--------	--------	---------	------

MOVX @DPTR,A	0xF0	1	2	Inv
MOVX @R0,A	0xF2	1	2	Inv
MOVX @R1,A	0xF3	1	2	Inv
MOVX A,@DPTR	0xE0	1	2	Inv
MOVX A,@R0	0xE2	1	2	Inv
MOVX A,@R1	0xE3	1	2	Inv

Descrizione: MOVX trasferisce un byte alla o dalla memoria esterna dal o all'Accumulatore.

Se l'Operando 1 e' @DPTR, l'Accumulatore e' trasferito nell'indirizzo a 16-bit della memoria esterna indicato da DPTR. Questa istruzione usa entrambe le porte P0 e P2 per trasferire l'indirizzo a 16-bit ed il dato verso l'esterno. Se l'operando 2 e' @DPTR il trasferimento viene eseguito dalla memoria esterna all'Accumulatore.

Se l'Operando 1 e' @R0 o @R1, l'Accumulatore e' trasferito nell'indirizzo a 8-bit della memoria esterna indicata dal registro corrispondente. Questa istruzione usa soltanto la porta P0 per trasferire l'indirizzo a 8-bit ed il dato verso l'esterno. La porta P2 non viene modificata. Se l'operando 2 e' @R0 o @R1 allora il byte viene trasferito dalla memoria esterna all'Accumulatore.

Vedi anche: [MOV](#), [MOVC](#), [Set d'istruzioni](#)

POP

Istruzione: POP

Funzione: Prendi il valore dallo Stack

Sintassi: POP

Istruzione	OpCode	N.Byte	N.Cicli	Flag
POP <i>iram addr</i>	0xD0	2	2	Inv

Descrizione: POP effettua il "pop" dallo stack all'indirizzo *iram addr* specificato. L'istruzione POP prende il valore contenuto nella RAM interna puntato dallo stack pointer e lo carica nell'indirizzo *iram addr*. Lo stack pointer viene poi decrementato di uno.

Vedi anche: [PUSH](#), [Set d'istruzioni](#)

PUSH

Istruzione: PUSH

Funzione: Metti il valore nello Stack

Sintassi: PUSH

Istruzione	OpCode	N.Byte	N.Cicli	Flag
PUSH <i>iram addr</i>	0xC0	2	2	Inv

Descrizione: PUSH effettua l'operazione di "push" del valore specificato da *iram addr* nello stack. L'istruzione PUSH, prima incrementa il valore dello stack pointer di uno, poi prende il valore contenuto nell'indirizzo *iram addr* e lo memorizza nella RAM interna all'indirizzo puntato dallo Stack Pointer.

Vedi anche: [POP](#), [Instruction Set](#)

XCH

Istruzione: XCH

Funzione: Scambia i byte

Sintassi: XCH A,*Registro*

Istruzione	OpCode	N.Byte	N.Cicli	Flag
XCH A,@R0	0xC6	1	1	Inv
XCH A,@R1	0xC7	1	1	Inv
XCH A,R0	0xC8	1	1	Inv
XCH A,R1	0xC9	1	1	Inv

XCH A,R2	0xCA	1	1	Inv
XCH A,R3	0xCB	1	1	Inv
XCH A,R4	0xCC	1	1	Inv
XCH A,R5	0xCD	1	1	Inv
XCH A,R6	0xCE	1	1	Inv
XCH A,R7	0xCF	1	1	Inv
XCH A, <i>iram addr</i>	0xC5	2	1	Inv

Descrizione: L'istruzione scambia il valore dell'Accumulatore con il valore del registro indicato nell'istruzione.

Vedi anche: [MOV](#), [Instruction Set](#)

XCHD

Istruzione: XCHD

Funzione: Scambia i digit

Sintassi: XCHD A,[@R0/@R1]

Istruzione	OpCode	N.Byte	N.Cicli	Flag
XCHD A,@R0	0xD6	1	1	Inv
XCHD A,@R1	0xD7	1	1	Inv

Descrizione: Scambia il nibble meno significato dell'Accumulatore con il nibble meno significativo della locazione di RAM interna indirizzata da R0 o R1. Il nibble piu' significativo dei registri non viene modificato.

Vedi anche: [DA](#), [Instruction Set](#)

[^ INDICE](#)
[SET D'ISTRUZIONI](#)

(C) Copyright 1997, 1998 by Vault Information Services. All Rights Reserved.

Le informazioni sono fornite senza alcuna garanzia. Per favore [vedi dettagli](#).

[Contattaci](#) per l'uso e/o il permesso di divulgazione di questo corso.

Traduzione italiana di: [Sergio Salvitti](#)

Corso 8051 – SET D'ISTRUZIONI

ISTRUZIONI SU VARIABILI A BIT

ANL

Istruzione: ANL

Funzione: AND a bit

Sintassi: ANL C, bit

Istruzione	OpCode	N.Byte	N.Cicli	Flag
ANL C, <i>bit addr</i>	0x82	2	1	C
ANL C, <i>/bit addr</i>	0xB0	2	1	C

Descrizione: Il valore del bit C e' calcolato in AND logico con il bit indirizzato e il risultato viene lasciato in C. Un simbolo "/" davanti al bit indica che verra' preso il suo valore negato. Il valore del bit indirizzato rimane in ogni caso inalterato.

Vedi anche: [ORL](#), [Set d'istruzioni](#)

CLR

Istruzione: CLR

Funzione: Poni a zero il bit

Sintassi: CLR *bit*

Istruzione	OpCode	N.Byte	N.Cicli	Flag
CLR <i>bit addr</i>	0xC2	2	1	Inv
CLR C	0xC3	1	1	C

Descrizione: CLR pone a zero il bit indicato nell'istruzione.

Vedi anche: [SETB](#), [Set d'istruzioni](#)

CPL

Istruzione: CPL

Funzione: Complementa il bit

Sintassi: CPL *bit*

Istruzione	OpCode	N.Byte	N.Cicli	Flag
CPL C	0xB3	1	1	C
CPL <i>bit addr</i>	0xB2	2	1	Inv

Descrizione: CPL complementa il valore del bit indicato nell'istruzione.

Vedi anche: [CLR](#), [SETB](#), [Set d'istruzioni](#)

JB

Istruzione: JB

Funzione: Salta se il bit e' ad uno

Sintassi: JB *bit addr, reladdr*

Istruzione	OpCode	N.Byte	N.Cicli	Flag
JB <i>bit addr,reladdr</i>	0x20	3	2	Inv

Descrizione: JB salta all'indirizzo indicato da *reladdr* se il bit indicato da *bit addr* e' da uno, altrimenti l'esecuzione del programma continua con l'istruzione successiva a quella dell'istruzione JB.

Vedi anche: [JBC](#), [JNB](#), [Set d'istruzioni](#)

JBC

Istruzione: JBC

Funzione: Salta se il bit e' ad uno e resettato

Sintassi: JB *bit addr, reladdr*

Istruzione	OpCode	N.Byte	N.Cicli	Flag
JBC <i>bit addr,reladdr</i>	0x10	3	2	Inv

Descrizione: JBC salta all'indirizzo indicato da *reladdr* se il bit indicato da *bit addr* e' ad uno e prima di saltare resetta il bit di condizione. Se il bit di condizione non e' settato l'esecuzione del programma continua con l'istruzione successiva a quella dell'istruzione JBC.

Vedi anche: [JB](#), [JNB](#), [Set d'istruzioni](#)

JC

Istruzione: JC

Funzione: Salta se il Carry e' settato

Sintassi: JC *reladdr*

Istruzione	OpCode	N.Byte	N.Cicli	Flag
------------	--------	--------	---------	------

JC <i>reladdr</i>	0x40	2	2	Inv
-------------------	------	---	---	-----

Descrizione: JC salta all'indirizzo indicato da *reladdr* se il Carry bit e' ad uno, altrimenti l'esecuzione del programma continua con l'istruzione successiva a quella dell'istruzione JC.

Vedi anche: [JNC](#), [Instruction Set](#)

JNB

Istruzione: JNB

Funzione: Salta se il bit non e' settato

Sintassi: JNB *bit addr,reladdr*

Istruzione	OpCode	N.Byte	N.Cicli	Flag
JNB <i>bit addr,reladdr</i>	0x30	3	2	Inv

Descrizione: JNB salta all'indirizzo indicato da *reladdr* se il bit indicato da *bit addr* non e' settato, altrimenti l'esecuzione del programma continua con l'istruzione successiva a quella dell'istruzione JNB.

Vedi anche: [JB](#), [JBC](#), [Set d'istruzioni](#)

JNC

Istruzione: JNC

Funzione: Salta se il Carry non e' settato

Sintassi: JNC *reladdr*

Istruzione	OpCode	N.Byte	N.Cicli	Flag
JNC <i>reladdr</i>	0x50	2	2	Inv

Descrizione: JNC salta all'indirizzo indicato da *reladdr* se il Carry bit non e' settato, altrimenti l'esecuzione del programma continua con l'istruzione successiva a quella dell'istruzione JNC.

Vedi anche: [JC](#), [Instruction Set](#)

MOV

Istruzione: MOV

Funzione: Trasferisci un bit

Sintassi: MOV *bit*

Istruzione	OpCode	N.Byte	N.Cicli	Flag
MOV C, <i>bit addr</i>	0xA2	2	1	C
MOV <i>bit addr</i> ,C	0x92	2	2	Inv

Descrizione: MOV copia il valore del secondo operando a bit nel primo. Il valore del secondo operando a bit rimane inalterato.

Vedi anche: [Set d'istruzioni](#)

ORL

Istruzione: ORL

Funzione: OR a bit

Sintassi: ORL *Operando1,Operando2*

Istruzione	OpCode	N.Byte	N.Cicli	Flag
ORL C, <i>bit addr</i>	0x72	2	2	C
ORL C, <i>/bit addr</i>	0xA0	2	1	C

Descrizione: Il valore del bit C e' calcolato in OR logico con il bit indirizzato e il risultato viene lasciato in C. Un simbolo "/" davanti al bit indica che verra' preso il suo valore negato. Il valore del bit indirizzato rimane in ogni caso inalterato.

Vedi anche: [ANL](#), [Set d'istruzioni](#)

SETB

Istruzione: SETB

Funzione: Setta il Bit

Sintassi: SETB *bit addr*

Istruzione	OpCode	N.Byte	N.Cicli	Flag
SETB C	0xD3	1	1	C
SETB <i>bit addr</i>	0xD2	2	1	Inv

Descrizione: Setta il bit sepcificato

Vedi anche: [CLR](#), [Set d'istruzioni](#)

[^ INDICE](#)
[SET D'ISTRUZIONI](#)

(C) Copyright 1997, 1998 by Vault Information Services. All Rights Reserved.

Le informazioni sono fornite senza alcuna garanzia. Per favore [vedi dettagli](#).

[Contattaci](#) per l'uso e/o il permesso di divulgazione di questo corso.

Traduzione italiana di: [Sergio Salvitti](#)

Corso 8051 – SET D'ISTRUZIONI

ISTRUZIONI DI SALTO

ACALL

Istruzione: ACALL

Funzione: Chiama una subroutine con salto assoluto in un blocco di 2K

Sintassi: ACALL *code address*

Istruzione	OpCode	N.Byte	N.Cieli	Flag
ACALL <i>page0</i>	0x11	2	2	Inv
ACALL <i>page1</i>	0x31	2	2	Inv
ACALL <i>page2</i>	0x51	2	2	Inv
ACALL <i>page3</i>	0x71	2	2	Inv
ACALL <i>page4</i>	0x91	2	2	Inv
ACALL <i>page5</i>	0xB1	2	2	Inv
ACALL <i>page6</i>	0xD1	2	2	Inv
ACALL <i>page7</i>	0xF1	2	2	Inv

Descrizione: ACALL chiama una subroutine senza condizioni all'indirizzo di codice indicato da *code address*. A questo punto viene salvato nello stack l'indirizzo dell'istruzione che segue ACALL, prima il byte meno significativo e poi quello piu' significativo. Il Program Counter viene caricato con il valore dove risiede la subroutine chiamata.

Il nuovo valore del Program Counter e' calcolato sostituendo il suo byte meno significativo con il secondo byte dell'istruzione ACALL e sostituendo i bit da 0 a 2 del byte piu' significativo del Program Counter con i 3 bit che indicano la pagina ove saltare. I bit da 3 a 7 del byte piu' significativo del Program Counter restano inalterati.

Poiche' ACALL modifica solo 11 bit del Program Counter, le chiamate possono dirette a routine locate all'interno dello stesso blocco di 2k come il primo byte che segue l'istruzione ACALL.

Vedi anche: [LCALL](#), [RET](#), [Set d'istruzioni](#)

AJMP

Istruzione: AJMP

Funzione: Salto assoluto in un blocco di 2k

Sintassi: AJMP *code address*

Istruzione	OpCode	N.Byte	N.Cicli	Flag
AJMP <i>page0</i>	0x01	2	2	Inv
AJMP <i>page1</i>	0x21	2	2	Inv
AJMP <i>page2</i>	0x41	2	2	Inv
AJMP <i>page3</i>	0x61	2	2	Inv
AJMP <i>page4</i>	0x81	2	2	Inv
AJMP <i>page5</i>	0xA1	2	2	Inv
AJMP <i>page6</i>	0xC1	2	2	Inv
AJMP <i>page7</i>	0xE1	2	2	Inv

Descrizione: AJMP salta senza condizioni all'indirizzo di codice indicato da *code address*.

Il nuovo valore del Program Counter e' calcolato sostituendo il suo byte meno significativo con il secondo byte dell'istruzione AJMP e sostituendo i bit da 0 a 2 del byte piu' significativo del Program Counter con i 3 bit che indicano la pagina ove saltare. I bit da 3 a 7 del byte piu' significativo del Program Counter restano inalterati.

Poiche' AJMP modifica solo 11 bit del Program Counter, le chiamate possono dirette a routine locate all'interno dello stesso blocco di 2k come il primo byte che segue l'istruzione AJMP.

Vedi anche: [LJMP](#), [SJMP](#), [Set d'istruzioni](#)

CJNE

Istruzione: CJNE

Funzione: Compara e salta se non uguale

Sintassi: CJNE *Operando1,Operando2,reladdr*

Istruzione	OpCode	N.Byte	N.Cicli	Flag
CJNE A,#data,reladdr	0xB4	3	2	C
CJNE A,iram addr,reladdr	0xB5	3	2	C
CJNE @R0,#data,reladdr	0xB6	3	2	C
CJNE @R1,#data,reladdr	0xB7	3	2	C
CJNE R0,#data,reladdr	0xB8	3	2	C
CJNE R1,#data,reladdr	0xB9	3	2	C
CJNE R2,#data,reladdr	0xBA	3	2	C
CJNE R3,#data,reladdr	0xBB	3	2	C
CJNE R4,#data,reladdr	0xBC	3	2	C
CJNE R5,#data,reladdr	0xBD	3	2	C
CJNE R6,#data,reladdr	0xBE	3	2	C
CJNE R7,#data,reladdr	0xBF	3	2	C

Descrizione: CJNE compara il valore dell'operando 1 con quello dell'operando 2 e salta all'indirizzo relativo indicato se essi non sono uguali. In caso contrario, il programma prosegue la sua esecuzione dell'istruzione successiva a quella dell'istruzione CJNE.

Il bit **Carry (C)** e' settato se l'operando 1 e' inferiore all'operando 2, altrimenti e' resettato.

Vedi anche: [DJNZ](#). [Set d'istruzioni](#)

DJNZ

Istruzione: DJNZ

Funzione: Decrementa e salta se il risultato non e' zero

Sintassi: DJNZ *Registro,reladdr*

Istruzione	OpCode	N.Byte	N.Cicli	Flag
DJNZ <i>iram addr,reladdr</i>	0xD5	3	2	Inv
DJNZ R0, <i>reladdr</i>	0xD8	2	2	Inv
DJNZ R1, <i>reladdr</i>	0xD9	2	2	Inv
DJNZ R2, <i>reladdr</i>	0xDA	2	2	Inv
DJNZ R3, <i>reladdr</i>	0xDB	2	2	Inv
DJNZ R4, <i>reladdr</i>	0xDC	2	2	Inv
DJNZ R5, <i>reladdr</i>	0xDD	2	2	Inv
DJNZ R6, <i>reladdr</i>	0xDE	2	2	Inv
DJNZ R7, <i>reladdr</i>	0xDF	2	2	Inv

Descrizione: DJNZ decrementa il valore del *registro* di 1. Se il valore iniziale del registro e' 0, l'operazione fara' in modo che esso contenga 255 (0xFF Esadecimale). Se il nuovo valore del Registro non e' zero, il programma saltera' all'indirizzo indicato da *relative addr*. Se invece il nuovo valore del Registro e' zero, il programma continuera' dall'istruzione che segue l'istruzione di DJNZ.

Vedi anche: [DEC](#), [JZ](#), [JNZ](#), [Set d'istruzioni](#)

JMP

Istruzione: JMP

Funzione: Salta al Data Pointer piu' l'Accumulatore

Sintassi: JMP @A+DPTR

Istruzione	OpCode	N.Byte	N.Cicli	Flag
JMP @A+DPTR	0x73	1	2	Inv

Descrizione: JMP salta senza condizioni all'indirizzo rappresentato dalla somma del valore di DPTR e il valore dell'Accumulatore.

Vedi anche: [Set d'istruzioni](#)

JNZ

Istruzione: JNZ

Funzione: Salta se l'Accumulatore non e' nullo

Sintassi: JNZ *reladdr*

Istruzione	OpCode	N.Byte	N.Cicli	Flag
JNZ <i>reladdr</i>	0x70	2	2	Inv

Descrizione: JNZ salta all'indirizzo indicato da *reladdr* se l'Accumulatore un qualsiasi valore tranne lo zero. Nell'altro caso, il programma continua con l'istruzione che segue l'istruzione JNZ.

Vedi anche: [JZ](#), [Instruction Set](#)

JZ

Istruzione: JZ

Funzione: Salta se l'Accumulatore e' nullo

Sintassi: JZ *reladdr*

Istruzione	OpCode	N.Byte	N.Cicli	Flag
JZ <i>reladdr</i>	0x60	2	2	Inv

Descrizione: JZ salta all'indirizzo indicato da *reladdr* se l'Accumulatore contiene il valore zero. Nell'altro caso, il programma continua con l'istruzione che segue l'istruzione JZ.

Vedi anche: [JNZ](#), [Instruction Set](#)

LCALL

Istruzione: LCALL

Funzione: Chiamata lunga

Sintassi: LCALL *code addr*

Istruzione	OpCode	N.Byte	N.Cicli	Flag
LCALL <i>code addr</i>	0x12	3	2	Inv

Descrizione: LCALL chiama una subroutine. Essa incrementa il Program Counter (per puntare all'istruzione seguente) ed effettua il push del PC nello stack (il byte meno significativo prima e poi quello piu' significativo). Il Program Counter viene caricato con il valore a 16-bit corrispondente ai due byte successivi al codice operativo dell'istruzione LCALL.

Vedi anche: [ACALL](#), [RET](#), [Set d'istruzioni](#)

LJMP

Istruzione: LJMP

Funzione: Salto lungo

Sintassi: LJMP *code addr*

Istruzione	OpCode	N.Byte	N.Cicli	Flag
LJMP <i>code addr</i>	0x02	3	2	Inv

Descrizione: LJMP salta senza condizioni all'indirizzo specificato da *code addr*.

Vedi anche: [AJMP](#), [SJMP](#), [JMP](#), [Set d'istruzioni](#)

NOP

Istruzione: NOP

Funzione: Nessuna operazione, attendi

Sintassi: NOP

Istruzione	OpCode	N.Byte	N.Cicli	Flag
NOP	0x00	1	1	Inv

Descrizione: NOP, come suggerisce anche il nome nonfanulla per la durata di un ciclo macchina. Essa e' generalmente usata per scopi di temporizzazione. Assolutamente, nessun Flag o registro viene alterato.

Vedi anche: [Set d'istruzioni](#)

RET

Istruzione: RET

Funzione: Ritorna da Subroutine

Sintassi: RET

Istruzione	OpCode	N.Byte	N.Cicli	Flag
RET	0x22	1	2	Inv

Descrizione: RET e' usato per tornare da una subroutine precedentemente chiamata da LCALL o da ACALL.

l'esecuzione del programma continua dall'indirizzo calcolato prendendo due byte dalla cima dello stack. Prima viene preso il byte piu' significativo e poi quello meno significativo.

Vedi anche: [LCALL](#), [ACALL](#), [RETI](#), [Set d'istruzioni](#)

RETI

Istruzione: RETI

Funzione: Ritorna da Interrupt

Sintassi: RETI

Istruzione	OpCode	N.Byte	N.Cicli	Flag
RETI	0x32	1	2	Inv

Descrizione: RETI e' usato per ritornare da un routine di servizio di un interrupt. RETI prima abilita tutti gli interrupt di priorit  uguale o inferiore a quella dell'interrupt che si sta terminando. Poi l'esecuzione del programma prosegue all'indirizzo calcolato prendendo due byte dalla cima dello stack. Prima viene preso il byte piu' significativo e poi quello meno significativo.

RETI funziona come RET che viene pero' utilizzato al di fuori delle routine di interrupt.

Vedi anche: [RET](#), [Instruction Set](#)

SJMP

Istruzione: SJMP

Funzione: Salto corto

Sintassi: SJMP *reladdr*

Istruzione	OpCode	N.Byte	N.Cicli	Flag
------------	--------	--------	---------	------

SJMP <i>reladdr</i>	0x80	2	2	Inv
---------------------	------	---	---	-----

Descrizione: SJMP salta senza condizioni all'indirizzo specificato da *reladdr*. *Reladdr* deve essere contenuto nel range da -128 a +127 byte dall'istruzione che segue SJMP stesso.

Vedi anche: [LJMP](#), [AJMP](#), [Set d'istruzioni](#)

[^ INDICE](#)
[SET D'ISTRUZIONI](#)

(C) Copyright 1997, 1998 by Vault Information Services. All Rights Reserved.

Le informazioni sono fornite senza alcuna garanzia. Per favore [vedi dettagli](#).

[Contattaci](#) per l'uso e/o il permesso di divulgazione di questo corso.

Traduzione italiana di: [Sergio Salvitti](#)

Note sulla traduzione del corso 8051

*Potete trovare la versione originale della traduzione in italiano all'indirizzo:
www.geocities.com/siliconvalley/garage/1748 ove risiede la mia home page.*

La versione originale in inglese risiede all'indirizzo: www.8052.com

Saluti

Sergio Salvitti

EMAIL: salvitti@geocities.com

--> -->