Sandro Cavalieri Foschini

Emanuele Richiardone

Microcontrollore

8051

Politecnico di Torino
A.A. 2003-04
Corso di Laurea in Ingegneria Informatica

Indice

Indice **2**

1. MICROCONTROLLORE 8051 .... **6**
1.1 Microcontrollori e l'8051 6
1.2. Package dell'8051 8

2. TIPI DI MEMORIA **12**
2.1 On-Chip 12
2.1.1 Banchi di Registri 13
2.1.2 Memoria "bit mapped" 14
2.1.3 Memoria utente e stack 15
2.1.4 Registri Speciali (SFR) 16
2.2 Memoria di programma esterna 16
2.3 RAM esterna 17

3. TIPI D'INDIRIZZAMENTO **18**
3.1 Indirizzamento Diretto 18
3.2 Indirizzamento Indiretto 19
3.3 Indirizzamento Immediato 19
3.4 Indirizzamento a registro 20
3.5 Indirizzamento indicizzato 20
3.6 Esterno Diretto 21
3.7 Esterno Indiretto 21
3.8 Indirizzamento implicito 21
3.9 Tabella riassuntiva 22

4. REGISTRI **23**
4.1 Registro accumulatore "ACC" 23
4.2 Registri "R" 23
4.3 Registro "B" 24
4.4 Data Pointer "DPTR" 24
4.5 Program Counter "PC" 24
4.5 Stack Pointer "SP" 25
4.6 Registri "SFR" 25
4.6.1 Special Function Registers 25
4.6.2 I vari tipi di SFR 26
4.6.2.1 SFR di controllo *26*
4.6.2.2 SFR generici *28*
4.6.2.3 SFR porte I/O *28*

5. ACCESSO SERIALI **30**
5.1 La porta seriale 30
5.2 Configurazione del modo di funzionamento 31
5.3 Configurazione del baud rate 33
5.4 Scrittura di un dato sulla seriale 35
5.5 Lettura di un dato sulla seriale 36

6. TIMER **37**
6.1 "Interval timer": misurare il tempo 38
6.1.1 La configurazione e inizializzazione dei timer 38

6.1.1.1 Il registro TMOD......39
6.1.1.2 Il registro TCON......41
6.2 Timer come contatori di numero eventi o durata singolo evento 42
6.3 La lettura dei timer 43
6.3.1 Modo 0 e 1 (timer a 13, 16 bit) 43
6.3.2 Modo 2 e 3 (timer a 8 bit) 43

7. INTERRUPT 44
7.1 Le sorgenti di interrupt 44
7.2 Abilitazione e configurazione degli interrupt 46
7.3 Interrupt seriali 47
7.4 Scrittura di un interrupt handler 48

8. L'ASSEMBLATORE A51 49
8.1 Le direttive dall'A51 50
8.1.1 BIT 50
8.1.2 DB 51
8.1.3 DS 51
8.1.4 DW 51
8.1.5 END 51
8.1.6 EQU 51
8.1.7 IF, ELSE, ENDIF 52
8.1.8 INCL 52
8.1.9 ORG 52
8.1.10 REG 52
8.1.11 SET 53

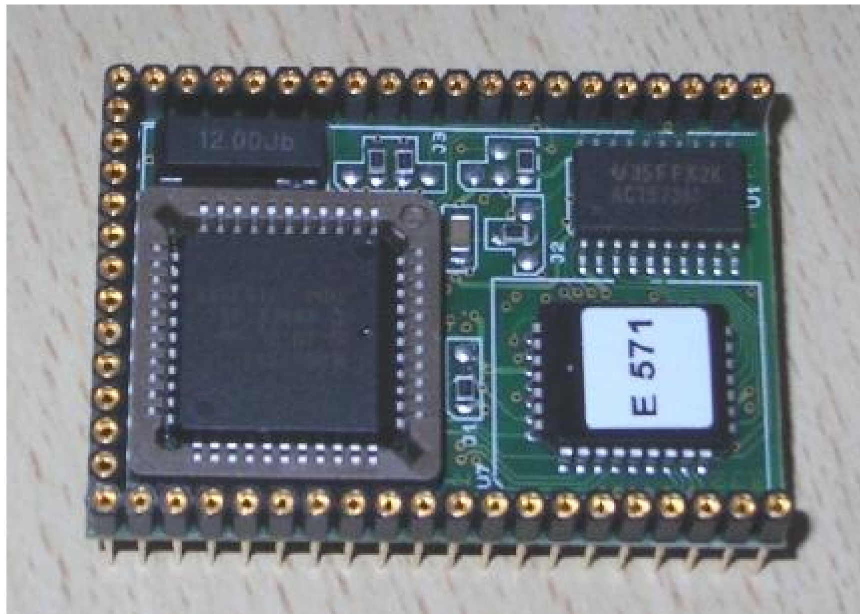
9. SET D'ISTRUZIONI DELL'8051 54
9.1 Istruzioni aritmetiche 55
9.2 Istruzioni logiche 55
9.3 Istruzioni trasferimento dati 56
9.3.1 RAM interna 56
9.3.2 RAM esterna 57
9.3.3 Istruzioni per la gestione delle tabelle dati 57
9.4 Istruzioni Booleane 58
9.5 Istruzioni di salto 59
9.5.1 Istruzioni di salto incondizionato 59
9.5.2 Istruzioni di salto condizionato 61
9.6 Tabella riassuntiva 61

10. PROGRAMMAZIONE ASSEMBLER 8051 63
10.1 Semplici programmi 63
10.1.1 blink.asm 63
10.1.2 hello.asm 66
10.1.3 name.asm 68
10.1.4 tabellina.asm 72
10.1.5 bubblesort.asm 76
10.1.6 twosort.asm 83
10.2 Riepilogo funzioni 91
10.2.1 Impostazione porta seriale in modalità a 8 bit con *auto reload* 91
10.2.2 Stampa di numeri, caratteri, stringhe su `SBUF` 91
10.2.3 Input con echo di numeri, caratteri, stringhe da `SBUF` 92
10.2.4 Implementazione *bubble sort* 93
10.2.5 Implementazione *counting sort* 94

10.3 Note sul software utilizzato 95
10.3.1 Utilizzo di Keil µVision2 95
10.3.2 Il debug 96
10.3.3 Utilizzo di Phytec Flashtools98 96

11. BIBLIOGRAFIA 97
11.1 L'hardware 8051 97
11.2 Programmazione Assembler 8051 97

12. INDICE DELLE FIGURE E TABELLE 98
12.1 Indice delle figure 98
12.2 Indice delle tabelle 98



Il seguente testo è articolato in due parti, nella prima si illustrano le specifiche tecniche e il funzionamento del microcontrollore 8051.
Nella seconda si analizzano alcuni esempi di programmi scritti nel linguaggio Assembler specifico di questa architettura.

1.
MICROCONTROLLORE 8051

1.1 Microcontrollori e l'8051

Il microcontrollore è un dispositivo che raggruppa su un unico chip tutto il necessario per un sistema a microprocessore.

- CPU
- Memoria RAM
- Memoria EPROM o EEPROM
- Porte I/O
- Timer e contatori
- UART o porte di comunicazione seriale speciali
- Eventuali convertitori A/D

L'8051 è uno di questi microcontrollori single-chip, progettato e prodotto dalla Intel a partire dal 1980.

Attualmente Intel non è l'unico produttore del microcontrollore: infatti diverse case costruttrici indipendenti producono sia il modello standard che versioni migliorate con periferiche aggiuntive incorporate (porte I/O aggiuntive, watch-dog, ADC, DAC, driver PWM, interfacce I2C ecc.).

L'8051 è il più diffuso nel mercato mondiale dei microcontrollori ed è disponibile in più di cento versioni.

Le caratteristiche principali di questo microcontrollore si possono riassumere in:

- **CPU** a **8 bit** ottimizzata per le applicazioni di controllo con capacità estesa di elaborazione booleana (su singoli bit);
- **128 byte** di memoria **RAM interna** per i dati al chip
- massimo **64 kbyte** di **memoria** indirizzabile per i **dati**
- **4 kbyte** di **memoria** per i **programmi interna** al chip
- massimo **64 kbyte** di **memoria** indirizzabile per i **programmi**
- **32 linee I/O** bidirezionali indirizzabili separatamente;
- **Generatore di clock** incorporato
- **5 interrupt** di cui 2 esterni e tutti con due livelli di priorità
- **2 timer/contatori** a 16 bit
- **Porta seriale full-duplex** (il baud rate è prodotto da uno dei due timer).

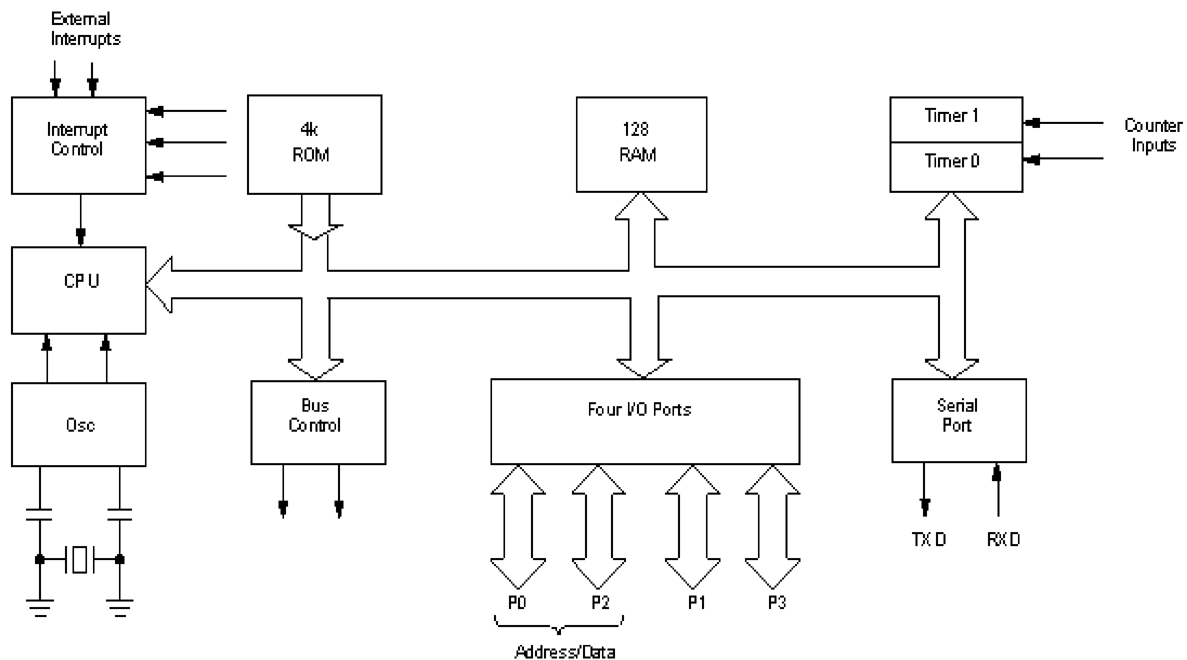


Fig. 1.1 – Architettura interna 8051: schema a blocchi

1.2. Package dell'8051

L'8051 utilizzato per la seguente trattazione è l'OKI M80C154S, a 12 MHz (frequenza imposta da un quarzo esterno).

Si tratta di un microcontrollore standard (non offre funzionalità aggiuntive solitamente introdotte da altri costruttori) e si presenta con un *package* QFJ a 44 pin.

Il microcontrollore è inserito su una basetta microMODUL-8051 di produzione Phytec, che presenta i seguenti componenti:

- l'OKI M80C154S
- un oscillatore a 12.000 MHz
- un latch ACT573M
- una memoria FLASH 29F040 (128kB, memoria di programma)
- una memoria RAM volatile (32kB, memoria RAM esterna)
- un Programmable Logic Device EPM7032S

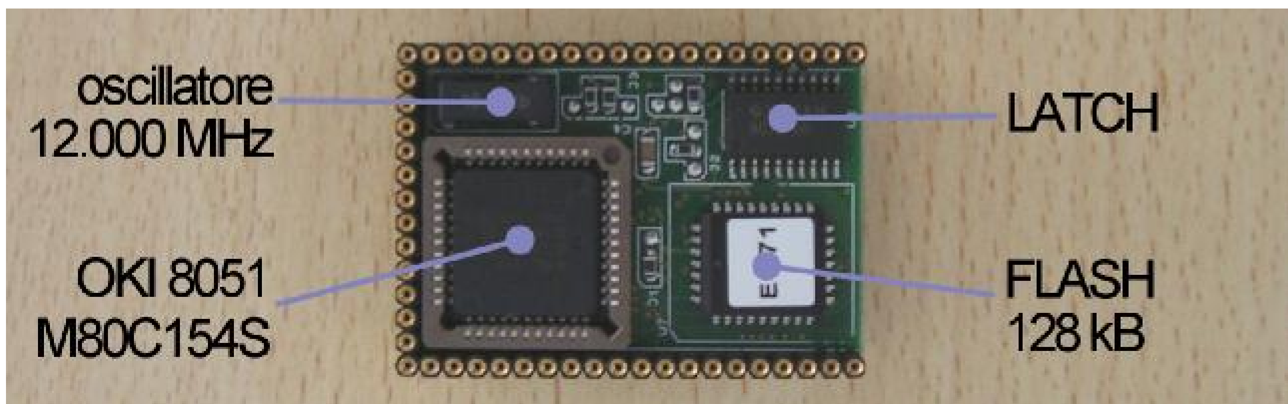


Fig. 1.2 – microMODUL-8051: vista anteriore del modulo

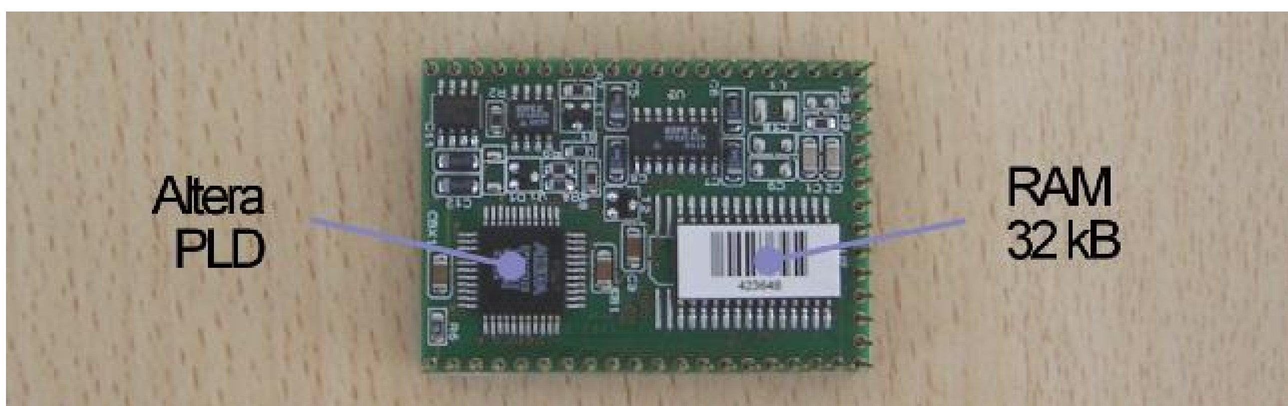


Fig. 1.3 – microMODUL-8051: vista posteriore del modulo

Questo modulo è montato su una basetta sempre di produzione Phytec; essa fornisce l'alimentazione, è provvista di due porte seriali, ha due led e due interruttori (*boot* e *reset*).

Fig. 1.4 – Basetta Phytec

Nello specifico la piedinatura del microcontrollore è la seguente.

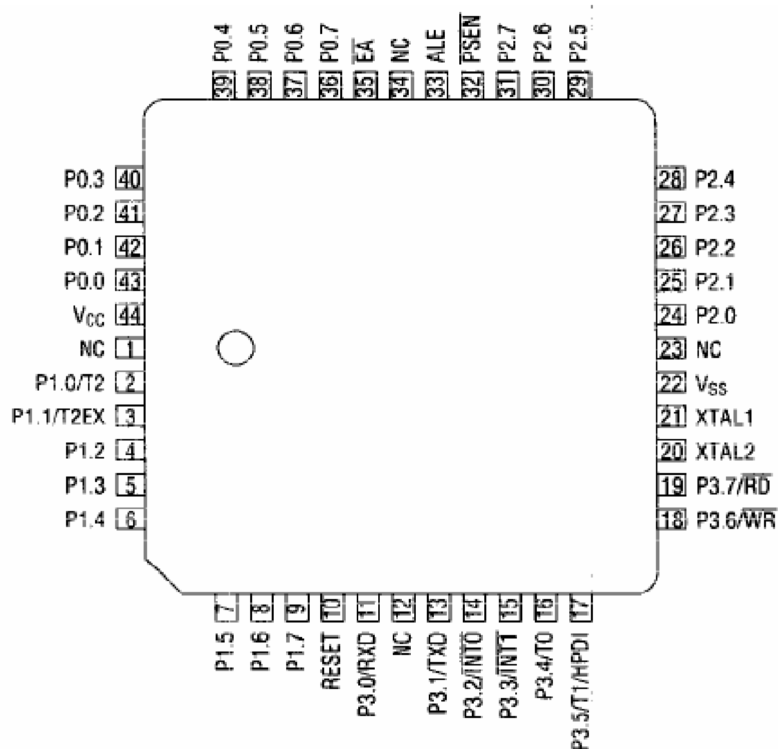


Fig. 1.5 – Disposizione PIN

Pin	Nome	Tipo	Descrizione
1-8	**P1.0 - P1.7**	I/O	**Port 1**: porta bidirezionale I/O ad 8 bit. Questa porta serve anche per ricevere la parte bassa dell'indirizzo durante la programmazione della EPROM interna (8751H) e durante la verifica della ROM (8051) o della EPROM (8751)
9	**RST**	I	**Reset**: un livello alto in questo piedino per due cicli macchina resetta il dispositivo. Un resistore interno a massa permette di ottenere il reset all'accensione aggiungendo un condensatore tra Vcc ed il piedino
10-17	**P3.0 - P3.7**	I/O	**Port 3**: porta bidirezionale I/0 ad 8 bit. Può assolvere funzioni particolari, come mostrato di seguito
10	**P3.0**	I	**Rxd**: Ingresso porta seriale
11	**P3.1**	U	**Txd**: Uscita porta seriale
12	**P3.2**	I	**#INT0**: Ingresso interrupt esterno 0
13	**P3.3**	I	**#INT1**: Ingresso interrupt esterno 1
14	**P3.4**	I	**T0**: Ingresso esterno timer/contatore 0
15	**P3.5**	I	**T1**: Ingresso esterno timer/contatore 1
16	**P3.6**	O	**WR**: Segnale di scrittura per la memoria RAM esterna
17	**P3.7**	O	**RD**: Segnale di lettura per memoria RAM esterna
18	**XTAL2**	O	**Crystal 2**: Uscita amplificatore dell'oscillatore
19	**XTAL1**	I	**Crystal 1**: Ingresso amplificatore invertente dell' oscillatore
20	**Vss**	I	**MASSA**
21-28	**P2.0 - P2.7**	I/O	**Port 2**: porta bidirezionale I/0 ad 8 bit. Questa porta serve anche per emettere la parte alta dell' indirizzo (A8-A15) durante la fase di fetch nella memoria esterna per i programmi, durante l'accesso ai nella memoria dei dati quando si utilizza l'indirizzamento a 16 bit. Inoltre, ha la funzione di ricevere la parte alta dell'indirizzo durante la programmazione della EPROM interna (8751)
29	**#PSEN**	O	**Program Store ENable**: è il segnale per la lettura per la memoria ROM o EPROM esterna (memoria per i programmi)
30	**ALE/#PROG**	O	**Address Latch Enable/Program Pulse**:segnale di abilitazione del latch che deve memorizzare la parte bassa dell'indirizzo durante l'accesso alla memoria esterna. Questo piedino viene utilizzato anche per fornire l'impulso di scrittura mentre si programma la EPROM (8751)
31	**#EA/Vpp**	I	**External Access Enable/Programming Supply Voltage:** se viene forzata a 0 permette di utilizzare solo la memoria ROM esterna escludendo i 4 k interni. Se viene forzata a 1 permette di indirizzare i 4 k interni ed eventualmente anche una memoria ROM o EPROM esterna per i programmi. Questo terminale riceve inoltre i 21 V di tensione durante la fase di programmazione dei 4 k di EPROM interna nella versione 8751H
32-39	**P0.0 - 0.7**	I/O	**Port 0**: porta di I/O bidirezionale ad 8 bit. La porta 0 serve anche come bus multiplato per trasferire la parte bassa dell'indirizzo e i dati durante un accesso alla memoria esterna. Inoltre ha la funzione di ricevere il dato durante la programmazione della EPROM interna (8751H) e di emettere il dato durante la verifica della ROM (8051) o della EPROM (8751)
Vcc	**+5V**	+5V	+5V

Tab. 1.1 – Descrizione piedinatura 8051

Tabella delle funzioni dei pin a seconda del modo di funzionamento			
	Modo di funzionamento		
Nome	**Accesso a memoria interna**	**Accesso a memoria esterna**	**Programmazione EPROM interna**
P1.0 - P1.7	Porta bidirezionale I/O ad 8 bit	Porta bidirezionale I/O ad 8 bit	Ricevere la parte bassa dell'indirizzo della cella di memoria da programmare
RST	Reset del microprocessore	Reset del microprocessore	Nulla, deve essere a 1
P3.0 - P3.7	Porta bidirezionale I/O ad 8 bit o funzioni speciali	Porta bidirezionale I/O ad 8 bit o funzioni speciali. P3.6 P3.7 in caso di accesso a memoria dati esterna vengono usati come RD e WR	Nulla ad eccezione di P3.6 P3.7 che devono essere a 1
XTAL2	Uscita amplificatore dell'oscillatore.	Uscita amplificatore dell' oscillatore.	Uscita amplificatore dell'oscillatore.
XTAL1	Ingresso amplificatore invertente dell' oscillatore .	Ingresso amplificatore invertente dell' oscillatore .	Ingresso amplificatore invertente dell' oscillatore .
Vss	Massa	Massa	Massa
P2.0 - P2.7	Porta bidirezionale I/0 ad 8 bit	Emette la parte alta dell' indirizzo (A8-A15) della cella di memoria	Riceve la parte alta dell'indirizzo (A8-A15) della cella di memoria da programmare
#PSEN	Sempre a 1	Segnale di Read per la memoria di programma esterna	Nulla, deve essere a 0
ALE/#PROG	Indica l'inizio del fetch di una istruzione	Abilita il latch che deve memorizzare la parte bassa (A0-A7) dell'indirizzo della memoria esterna	Ingresso impulso di programmazione (100uS)
#EA/Vpp	Sempre a 1	Se a 0 permette di utilizzare solo la memoria ROM esterna escludendo i 4 k interni	Tensione di programmazione della EPROM (21V o 12.75V)
P0.0 - 0.7	Porta di I/O bidirezionale ad 8 bit	Emette la parte bassa dell'indirizzo (A0-A7) della cella di memoria o funziona come bus dati bidirezionale (D0-D7)	Riceve il dato da programmare
Vcc	+5V	+5V	+5V

Tab. 1.2 – Funzioni dei PIN a seconda dell'impostazione del modo di funzionamento del microcontrollore.

2.
TIPI DI MEMORIA

L’8051 ha tre tipi di memoria

- **On-Chip** (RAM interna e SFR, 128 Byte)
- Memoria di **programma esterna** (64 kByte)
- **RAM esterna** (max 64 kByte)

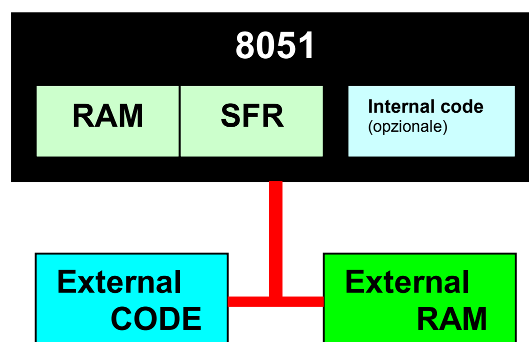


Fig. 2.1 – Tipi di memoria dell’8051

2.1 On-Chip

Risiede fisicamente nel microcontrollore, infatti l’8051 include internamente una certa quantità di memoria. In effetti la memoria interna è di due tipi

- RAM interna (banco registri, memoria a bit, memoria utente e stack)
- Area registri speciali (*SFR memory*)

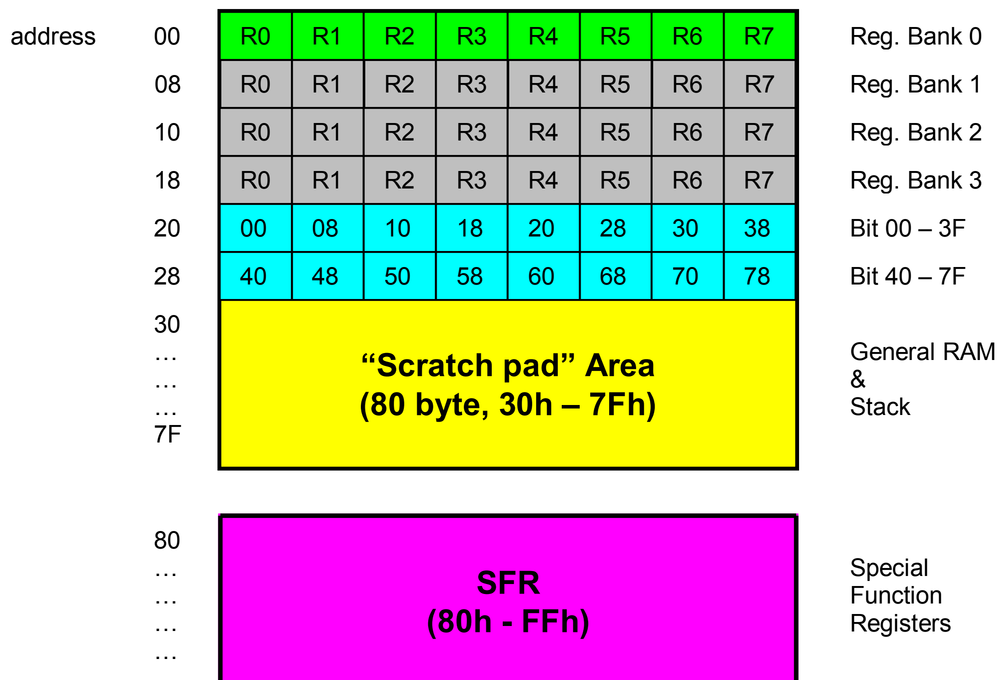


Fig. 2.2 – Organizzazione della memoria On-Chip

L'8051 ha un banco di 128 byte di RAM interna, che risulta essere il tipo di memoria (volatile) più veloce e più flessibile in lettura e scrittura.

I 128 byte di RAM interna sono suddivisi in tre zone a funzionalità specifica:

- **Banco di registri 0** (**00h-07h**, i primi 8 byte). L'eventuale utilizzo dei banchi 1, 2 e 3 (indirizzi da 08h a 1Fh) si imposta modificando il valore di alcuni registri SFR.
- **Memoria a bit** (**20h-2Fh**, i primi 8 byte)
- **80 byte** (**30h-7Fh**) di parte libera che possono essere usati come variabili utente che richiedono accessi frequenti e veloci.

2.1.1 Banchi di Registri

L'8051 è dotato di 8 registri *general purpose*: i registri "R", numerati da 0 a 7 (R0, R1, R2, R3, R4, R5, R6, e R7) che sono utilizzati da molte delle sue istruzioni assembler.

L'uso principale di questi registri è la memorizzazione di dati temporanei, o il trasferimento di dati tra le locazioni di memoria.

Esempio

L'addizione del valore di contenuto nel registro **R4** con il valore del registro accumulatore **A** si scrive

```
ADD A, R4
```

Ed il risultato è contenuto sempre in **A**.

Il registro **R4** è realmente parte della RAM interna, in particolare il suo indirizzo è 04h.

Di conseguenza l'istruzione

```
ADD A, 04h
```

porta allo stesso risultato della precedente, infatti si addiziona il valore contenuto nella locazione di memoria d'indirizzo 04h con il valore dell'accumulatore e si e carica il risultato nell'accumulatore stesso.

I banchi di registri risiedono nei primi 32 byte della RAM interna.

L'8051 dispone di 4 distinti banchi di registri, selezionabili dall'utente. Quindi sebbene dal bootstrap del microcontrollore il banco 0 (indirizzi da 00h a 07h) sia impostato per default, è possibile rimappare i registri richiedendo all'8051 di usare un banco di registri diverso. Ad esempio selezionando il banco 3 il registro **R4** diventa sinonimo di RAM interna alla locazione 1Ch.

Nota

Usando solamente il primo banco di registri (banco 0), le locazioni di RAM interna da 08h a 1Fh possono essere usate per altri scopi. Ma usando anche i banchi di registri 1, 2, 3, accedendo ad indirizzi di memoria al di sotto di 20h si sovrascrive il valore di un registro R!

2.1.2 Memoria "bit mapped"

L'8051 permette di accedere a 128 *variabili a bit* (che possono assumere o il valore 1 o il valore 0), numerate da 00h a 7Fh.

La memoria a bit non è realmente un nuovo tipo di memoria ma solo una parte di 16 byte della RAM interna, con indirizzi compresi da 20h a 2Fh. Poiché l'8051 fornisce delle istruzioni particolari per accedere a questa area effettuando operazioni bit a bit, è utile ritenerla come se fosse una memoria separata.

I comandi **SETB** e **CLR** permettono all'utente un immediato accesso a queste variabili.

Esempio

L'istruzione che pone a 1 il bit numero 20 (esadecimale) e azzera il 24 è:

```
SETB 20h

CLR 24h
```

Nota

Scrivendo il valore FFh nell'indirizzo 20h in effetti si impostano tutti i bit da 00h a 07h.
Quindi le istruzioni

```
MOV 20h,#0FFh
```

e

```
SETB 00h
SETB 01h
SETB 02h
[…]
SETB 07h
```

sono equivalenti.

È lecito utilizzare le locazioni di RAM interna da 20h a 2Fh come ulteriore area RAM nel caso il programma non richieda l'uso di variabili a bit. In caso contrario prestare attenzione a non sovrascrivere i bit con le variabili utente.

Le variabili a bit di indirizzo oltre 80h sono utilizzate per accedere ad alcuni registri SFR su base bit.

2.1.3 Memoria utente e stack

Questa area di memoria è utilizzabile per memorizzare le variabili del programma assembler che richiedono accessi frequenti e veloci. Ma questa stessa area è utilizzata dall'8051 per memorizzare lo *stack*.
Poiché in realtà gli 80 byte sono anche usati dalle variabili impostate dall'utente l'ampiezza dell'area dello stack è molto limitata.

Per *default* il microcontrollore dal *bootstrap* inizializza SP (lo Stack Pointer) al valore 08h.
Ciò significa che lo stack inizierà da questo valore e crescerà verso l'alto.

Nota

Per evitare la sovrascrittura dei banchi di registri 1, 2 e 3 (nel caso siano utilizzati dal programma assembler) lo stack pointer deve essere inizializzato sopra l'indirizzo del più alto banco di registri utilizzato.

Anche se si ipotizza l'uso della memoria a bit SP va inizializzato ad un valore più alto di 2Fh per garantire che le variabili a bit siano suddivise dallo stack.

2.1.4 Registri Speciali (SFR)

I registri speciali (SFR) sono aree di memoria che controllano specifiche funzionalità dell'8051, quali l'accesso alle 32 linee di I/O, l'accesso in lettura e scrittura sulla porta seriale (e l'impostazione del suo baud rate), l'accesso ai timer e la configurazione del sistema degli interrupt.

Utilizzando il metodo di accesso alla memoria di 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.

Esempio

L'istruzione

```
MOV 45h, #01h
```

scrive il valore "1" nella locazione 45 (esadecimale) della memoria interna;

Allo stesso modo l'istruzione

```
MOV 99h, #01h
```

scrive il valore "1" sulla porta seriale dell'8051. Infatti l'indirizzo 99 (esadecimale) corrisponde al registro SFR denominato **SBUF** che permette l'I/O seriale.

Nota

L'area dei registri SFR (memoria ad indirizzi superiori di 80h) è riservata e non può essere utilizzata come RAM addizionale.

Non tutte le locazioni dell'area SFR sono utilizzate per contenere i registri. Le rimanenti locazioni sarebbero teoricamente utilizzabili per contenere dati, ma la Intel raccomanda di non utilizzarle per questo scopo poiché il risultato sarebbe la lettura di dati casuali e la scrittura in tali locazioni non avrebbe alcun effetto. Queste locazioni sono riservate alla ditta costruttrice del chip per futuri sviluppi.

2.2 Memoria di programma esterna

In genere si tratta di una EPROM esterna, limitata a 64K, che contiene le istruzioni che l'8051 deve eseguire. Infatti la CPU può effettuare il *fetch* delle istruzioni solo da questo tipo di memoria.

Ma, in funzione della versione di 8051 usata, possono essere usate diverse combinazioni di tipi di memoria, per esempio è possibile avere 4K di memoria di codice nella PROM interna del chip e 64K di memoria di codice esterna in una EPROM.

Nel modo di funzionamento a memoria interna (EA=1) la CPU nel primi 4K di indirizzamento, esegue il fetch nella memoria interna, per indirizzi superiori a 0FFFh il fetch è effettuato nella memoria esterna.

Nel modo di funzionamento a memoria esterna la CPU esegue sempre il fetch nella memoria esterna per tutta l'area di indirizzamento (0000h - FFFFh).

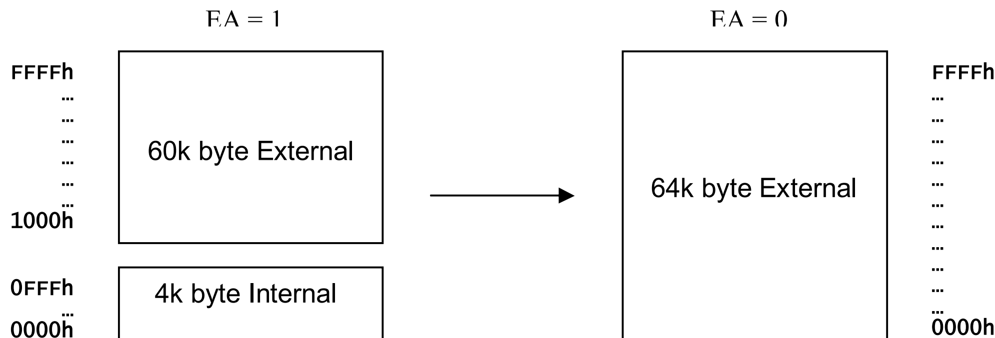


Fig. 2.3 – Metodi di funzionamento memoria di programma

Nota

In genere poiché la memoria programma è limitata a 64k la dimensione dei programmi per l'8051 non dovrebbe superare tale limite. Infatti solo utilizzando particolari assemblatori su basette con microcontrollori 8051 e componenti esterne EPROM organizzate a banchi è possibile superare a tale limite.

Durante l'accesso alla memoria esterna la CPU sfrutta P0 come parte bassa del bus indirizzi o bus dati e P2 come parte alta del bus indirizzi, di conseguenza non è più possibile sfruttarle come porte I/O.

2.3 RAM esterna

La RAM esterna in genere è implementata come una FLASH o una RAM statica ad accesso casuale situata fuori dal chip.

Risulta ovviamente più lenta della RAM interna. Ad esempio incrementare una variabile nella RAM *interna* richiede una sola istruzione ed un solo ciclo d'istruzione, mentre se la variabile risiede nella RAM *esterna* l'operazione risulta 7 volte più lenta poiché sono necessarie 4 istruzioni per un totale di 7 cicli d'istruzione.

La RAM interna è limitata a 128 byte (256 byte nell'8052), quella esterna può arrivare fino a 64K.

Nota

L'8051 può indirizzare soltanto 64 kByte di RAM. Per espandere tale capacità è richiesta un'apposita basetta ed un compilatore che supporti la gestione della RAM a banchi.

3.
TIPI D'INDIRIZZAMENTO

3.1 Indirizzamento Diretto

Si specifica *direttamente* nell'istruzione l'**indirizzo** della cella di memoria che contiene il **valore** dell'operando.

L'operando è specificato nell'istruzione da un campo indirizzo ad 8 bit.

L'indirizzamento diretto è possibile solo con i seguenti valori:

- **00h** a **7Fh** che sono riferiti alla **memoria interna**
- **80h** a **FFh** che sono riferiti ai **registri SFR** (*)

(*) Non si può usare l'indirizzamento diretto per trattare i 128 byte superiori della RAM interna dell'8051: è necessario usare l'indirizzamento indiretto.

Esempio

```
        MOV A, 20h
```

Il registro accumulatore A è caricato con il valore contenuto nella locazione di RAM interna all'indirizzo 20 esadecimale. Solo i dati della RAM interna e dell'area SFR possono essere indirizzati direttamente.

Vantaggi

- Velocità (il valore da caricare è contenuto nella memoria interna dell'8051, che è molto veloce)
- Elevata flessibilità (il valore è contenuto in un determinato indirizzo modificabile).

3.2 Indirizzamento Indiretto

Si specifica nell'istruzione l'indirizzo di una cella di memoria che contiene l'**indirizzo** dell'operando.

Esempio

```
MOV A, @R0
```

Con questa istruzione il microcontrollore 8051 analizza il valore contenuto nel registro interno R0. Quindi caricherà nel registro A il valore contenuto nella RAM interna alla locazione il cui indirizzo è contenuto in R0.

Sia la RAM interna sia quella esterna possono essere indirizzate indirettamente. I registri d'indirizzo per indirizzi ad 8 bit possono essere R0 o R1 del banco di registri selezionato, oppure lo stack pointer SP. Il registro d'indirizzo per gli indirizzi a 16 bit può essere solo il registro DPTR (Data PoinTeR).

Vantaggi

- Elevata flessibilità (il valore è contenuto in un determinato indirizzo modificabile)
- Possibilità di indirizzare ai 128 byte extra della RAM interna dell'8051.

Nota

L'indirizzamento indiretto non è mai relativo a un registro SFR, ma sempre alla RAM interna o esterna (quindi, ad esempio, non si può utilizzare per scrivere sulla porta seriale).

3.3 Indirizzamento Immediato

Si specifica nell'istruzione il valore dell'operando. Il codice operativo riporta *immediatamente* il valore da memorizzare. Infatti in alcune istruzioni il byte che segue il codice operativo (identificato con il prefisso "#") può rappresentare il valore di una costante e non di un indirizzo.

Esempio

```
MOV A, #10h
```

Il registro accumulatore A è caricato con il valore (in questo caso la costante esadecimale 10) che segue immediatamente.

Vantaggi

- Elevata velocità (il valore da caricare è gia contenuto nell'istruzione stessa)

Svantaggi

- Nessuna flessibilità (il valore è *hard-coded*, stabilito alla compilazione).

3.4 Indirizzamento a registro

Alcune istruzioni possono accedere ai banchi di registri che contengono i registri R0 - R7. Questo è possibile grazie a 3 bit che selezionano uno degli 8 registri (R0 - R7) interni al codice operativo. Il banco utilizzato dipende dallo stato dei bit RS0 e RS1 del registro di stato PSW.

Esempio

```
        ADD A, R7
```

L'istruzione effettua la somma tra il contenuto del registro A e il contenuto del registro R7. Il risultato è riposto in A.

Vantaggi

- Elevata efficienza dal punto di vista del codice macchina (elimina il byte del campo indirizzo)

3.5 Indirizzamento indicizzato

È possibile utilizzare l'indirizzamento indicizzato solo con la memoria di codice (memoria a sola lettura). Questo metodo di indirizzamento è stato previsto per permettere l'accesso a tabelle di ricerca nella memoria dei programmi. Un registro a 16 bit (DPTR oppure il *program counter*) punta alla base della tabella e l'accumulatore viene è utilizzato come indice all'interno della tabella. L'indirizzo effettivo del dato nella tabella ottenuto sommando a DPTR o PC il contenuto dell'accumulatore. Un altro tipo di indirizzamento indicizzato è utilizzato nell'istruzione di salto con selezione multipla. In questo caso l'indirizzo di salto è calcolato sommando alla base il contenuto dell'accumulatore.

Esempio

```
        MOVC A, @A+DPTR
```

L'istruzione pone nel registro A il contenuto della locazione di memoria di programma ricavata da A+DPTR.

3.6 Esterno Diretto

Questo tipo di indirizzamento è simile all'indirizzamento diretto, ma tratta la memoria esterna anziché quella interna.

È utilizzabile solamente con queste due istruzioni

```
MOVX A, @DPTR

MOVX @DPTR, A
```

L'indirizzo della memoria esterna alla quale si vuole accedere deve essere caricato con nel registro DPTR.
La prima istruzione carica il valore della memoria nel registro accumulatore A; la seconda – viceversa – registra nella memoria esterna il valore dell'accumulatore.

3.7 Esterno Indiretto

Anche questo modo d'indirizzamento agisce sulla memoria esterna.

Esempio

```
MOVX @R0, A
```

Il contenuto di R0 è utilizzato come indirizzo di memoria esterna dove caricare il valore dell'accumulatore, che in verità è a sua volta un indirizzo di memoria.

Nota

Poiché il valore di @R0 può soltanto essere incluso nell'intervallo da 00h a FFh, questo metodo permette di indirizzare soli 256 byte (e quindi di usufruire una così piccola quantità di RAM esterna).

3.8 Indirizzamento implicito

Alcune istruzioni sono specifiche di alcuni registri per cui nel codice operativo è implicitamente specificato a quali registri si farà l'accesso.

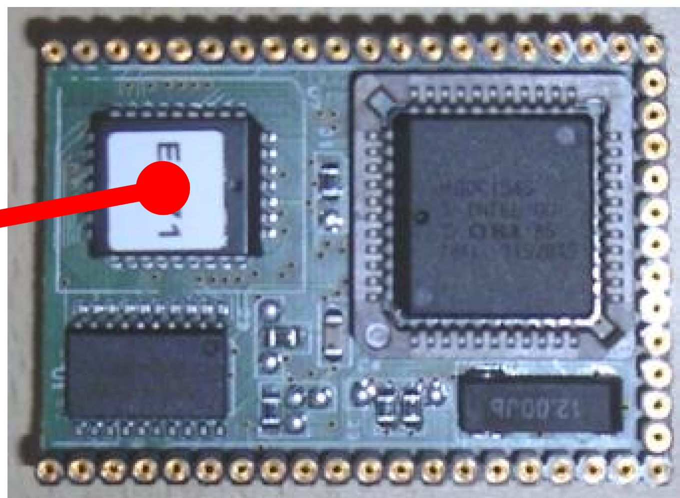
Esempio

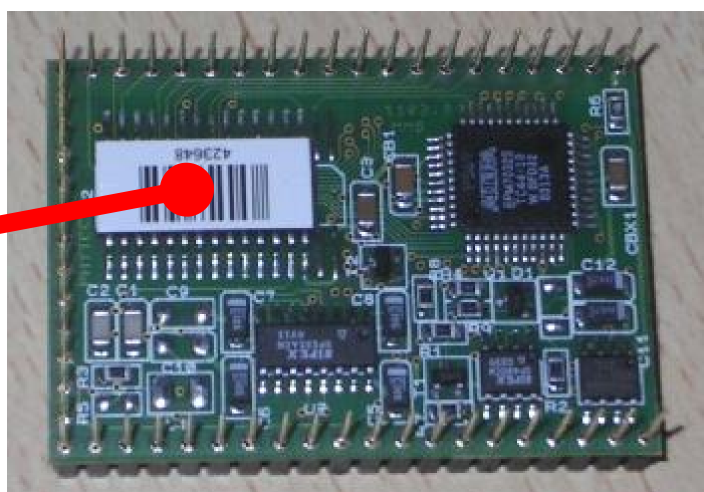
```
DIV AB
```

3.9 Tabella riassuntiva

TIPO	ESEMPIO
Diretto	`MOV A, 20h`
Indiretto	`MOV A, @R0`
Immediato	`MOV A, #10h`
A registro	`ADD A, R7`
Indicizzato	`MOVC A, @A+DPTR`
Esterno diretto	`MOVX A, @DPTR` oppure `MOVX @DPTR, A`
Esterno indiretto	`MOVX @R0, A`
Implicito	Indirizzo contenuto nell'istruzione specifica

Tab. 3.1 – Sintesi dei modi d'indirizzamento





4.
REGISTRI

4.1 Registro accumulatore "ACC"

Il registro ACC (indirizzabile a bit) può contenere un valore di 8 bit, ed è un registro di uso generale per accumulare i risultati di un gran numero di istruzioni (operazioni logico-aritmetiche e di trasferimento): infatti è utilizzato da più della metà delle 255 istruzioni dell'8051. È indicato semplicemente con A nei codici mnemonici delle istruzioni. Risiede all'indirizzo E0h.

4.2 Registri "R"

I registri R sono un set di 8 registri denominati: R0, R1, …, R7, usati come registri ausiliari in molte operazioni.

Ad esempio, per eseguire la somma 1 + 2, supponendo che il valore iniziale 1 risieda nell'accumulatore mentre il valore 2 risieda nel registro R5 si esegue l'istruzione

```
        ADD A, R5
```

e al termine dell'istruzione l'accumulatore conterrà il valore 3.

Il registro accumulatore da solo sarebbe poco utile se non avesse il supporto di questi registri ausiliari, usati anche per memorizzare valori temporanei.

4.3 Registro "B"

Anche il registro B (indirizzabile a bit) può memorizzare una parola di 8 bit.

Esso è usato soltanto in due istruzioni dell'8051: MUL AB e DIV AB, utilizzate per moltiplicare o dividere velocemente un numero con un altro. Nel registro B si può memorizzare il secondo numero (quindi o un fattore o il divisore) dell'operazione.

Oltre che alle due istruzioni MUL e DIV, il registro B è spesso utilizzato come un ulteriore registro temporaneo come se fosse il nono registro R. Risiede all'indirizzo F0h.

4.4 Data Pointer "DPTR"

Il registro DaTa PoinTer (DPTR) è un registro dell'8051 accessibile soltanto a 16 bit, mentre gli altri registri A, B ed R sono tutti ad un singolo byte.

Il DPTR - come suggerisce il nome (puntatore dati) - è adoperato per indirizzare i dati. Esso è 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 utilizza l'indirizzo contenuto nel registro DPTR. Essendo un registro a 16 bit, risultano quindi indirizzabili fino a 64k di memoria. Esso è inoltre utilizzato da alcune istruzioni che riguardano il codice di programma.

Anche se lo scopo iniziale del DPTR è 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.

In realtà il valore a 16 bit è ottenuto leggendo assieme i due registri (da 8 bit) DPL e DPH che compongono il valore del DPTR. È un numero intero senza segno compreso tra 0000h e FFFFh (da 0 a 65535 decimale). Risiedono all'indirizzo 82h (DPL) o 83h (DPH).

Nota

Mentre l'8051 è dotato di un'istruzione per l'incremento del DPTR che tratta i due registri da un byte come se fossero uno solo registro da 16 bit non esiste l'operazione opposta di decremento, questa – se necessaria – va scritta manualmente nel programma.
Anche l'operazione di push nello stack del DPTR va fatta in due tempi: ricordarsi di eseguire sia il push del registro DPL che quello di DPH.

4.5 Program Counter "PC"

Il Program Counter (PC - Contatore di Programma) è un indirizzo a 2 byte che punta alla locazione di memoria programma dove l'8051 prenderà la prossima istruzione da eseguire. Allo *startup*, il microcontrollore inizializza questo registro al valore 0000h e lo incrementa ogni volta che è eseguita una istruzione. È necessario notare che non sempre il PC viene incrementato di uno. Poiché alcune istruzioni richiedono 2 o 3 byte il PC sarà incrementato di 2 o di 3 opportunamente.

Non esiste un modo diretto per modificare il valore del Program Counter, eseguendo un'ipotetica operazione del tipo PC=2430h. Solo attraverso l'istruzione di salto è modificato il valore in modo non incrementale.

4.5 Stack Pointer "SP"

Lo Stack Pointer (SP - Puntatore dello Stack) è un indirizzo usato per indicare la cima dello stack, e può contenere un valore a 8 bit, dunque può indirizzare uno stack delle dimensioni massime di 256 byte.

Effettuando il *push* di un valore nello stack l'8051 prima incrementa il valore di SP e poi memorizza il valore nella locazione di memoria puntata da esso. Viceversa con l'operazione di *pop* di un dato dallo stack il microcontrollore prende il valore puntato dallo stack pointer e dopo decrementa il suo valore.

L' ordine con cui sono effettuate le operazioni di stack è importante. Allo startup, l'8051 inizializza automaticamente SP al valore 07h. Se immediatamente si richiede un push, il valore sarà memorizzato all'indirizzo 08h della RAM interna. Ciò è giustificato da quello detto precedentemente. Prima viene incrementato SP (da 07h a 08h) e poi viene memorizzato il valore nella locazione di memoria all'indirizzo (08h). *Lo stack cresce verso gli indirizzi alti della memoria RAM interna*.

La modifica di SP è effettuata in maniera automatica dall'8051 quando esegue le sei istruzioni seguenti: PUSH, POP, ACALL, LCALL, RET, e RETI, oltre che nelle operazioni di interrupt. Risiede all'indirizzo 81h.

4.6 Registri "SFR"

4.6.1 Special Function Registers

L'8051 standard è dotato di 21 registri SFR che permettono, mediante lettura o modifica del loro contenuto attraverso il programma utente, di impostare i diversi modi operativi del microcontrollore.

L'accesso ai registri, ognuno identificato con un proprio nome ed indirizzo, avviene come se questi fossero parte della normale RAM interna, ricordando che la RAM interna è compresa fra gli indirizzi da 00h a 7Fh, mentre i registri SFR risiedono nell'area dall'indirizzo 80h all'indirizzo FFh.

Anche se questo *range* di indirizzi offre 128 possibili byte, ci sono solo 21 registri SFR nell'8051 standard, come illustrato nella seguente tabella che riporta graficamente la disposizione dei registri SFR.

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

SFR di controllo
SFR porte I/O
Altri SFR

Fig. 4.1 – Tipi di registri SFR

Tutti gli altri indirizzi sono considerati non validi, e la loro lettura o può produrre un valore indefinito, oltre che una eventuale incompatibilità del programma con altri derivati dell’8051 che usano quei registi per l’implementazione di altre funzionalità specifiche di quel modello.

4.6.2 I vari tipi di SFR

4.6.2.1 SFR di controllo

Sono i registri segnati in blu in fig. 4.1, che permettono il controllo del modo operativo o della configurazione dell’8051.

PSW – D0h Program Status Word: Il registro PSW (indirizzabile a bit) contiene alcuni bit importanti che vengono aggiornati nell’esecuzione delle istruzioni dell’8051, quali il carry flag, il carry flag ausiliario, il flag di overflow ed il flag di parità e i flag utilizzati per selezionare il banco di registri “R” attivo.

CY	AC	F0	RS1	RS0	OV	-	P

Fig. 4.2 – La PSW

- **CY** (PSW.7): Carry Flag: Flag di riporto. Questo bit viene utilizzato come accumulatore dalle istruzioni che operano sui singoli bit. In tal caso il bit suddetto viene indicato con la lettera C.
- **AC** (PSW.6): Auxiliary Carry. Flag di riporto ausiliario. Utilizzato nelle operazioni in aritmetica BCD.
- **FO** (PSW.5): Flag 0 utilizzabile dall’utente.
- **RS1** (PSW.4): Register bank Selector bit 1.

- **RS0** (PSW.3): Register bank Selector bit 0. Insieme a RS1 seleziona il banco di 8 registri. R0 - R7 nella memoria RAM interna da 128 byte:

RS1	RS1	banco registri	indirizzi
0	0	0	00H-07H
0	1	1	08H-0FH
1	0	2	10H-17H
1	1	3	18H-1FH

Tab. 4.1 – Register bank selector RS1 e RS2

- **OV** (PSW.2): OVerflow; flag che indica il superamento dei limiti dell'operando.
- **-** (PSW.1): Flag definibile dall'utente.
- **P** (PSW.0): Parity flag. Flag di parità. Il suo valore rispecchia direttamente il numero di "1" presenti nell'accumulatore. P=1 se il numero di bit ad 1 nell'accumulatore è dispari; P=0 se il numero di bit ad 1 nell'accumulatore è pari.

Nota

Nelle routine di interrupt è necessario preservare, salvandolo nello stack, il contenuto del registro PSW all'inizio della procedura per permettere di ripristinarne il contenuto iniziale prima del ritorno nel programma principale. Il codice dell'*interrupt handler* non deve infatti influenzare lo svolgimento del programma principale cambiandone lo stato del PSW.

TCON – 88h Timer Control: Il registro TCON (indirizzabile a bit) abilita o disabilita il funzionamento dei due timer dell'8051 e contiene un flag per indicare l'overflow di un timer. Inoltre 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 l'eventuale attivazione di un interrupt esterno.

TMON – 89h Timer Mode: Il registro TMON imposta il modo di funzionamento di dei due timer: contatore a 16 bit, un timer a 8-bit con auto-caricamento, come un timer a 13 bit o come due timer separati. Altre possibili configurazioni sono quelle di "contatore di eventi" segnalati da un apposito pin esterno.

SCON – 98h Serial Control: Il registro SCON (indirizzabile a bit) imposta il comportamento della porta seriale dell'8051 impostandone l'abilitazione della ricezione e la sua velocità (baud rate). Inoltre contiene dei flag per indicare quando l'operazione di trasmissione o ricezione di un byte è terminata con successo.

Nota

L'uso della porta seriale è vincolato all'impostazione dei registri TCON e TMOD oltre al sopraindicato SCON, poiché sebbene il controllo della seriale sia compito di questo registro, nella maggior parte dei casi il programma prevede di usare uno dei due timer come generatore di baud rate configurandone TCON e TMOD.

PCON – 87h Power Control: Il registro PCON è usato per controllare il power mode dell'8051 permettendone il passaggio allo stato *sleep* che richiede per il funzionamento minor potenza. In più, uno dei bit di PCON è impiegato per raddoppiare il baud rate effettivo della porta seriale dell'8051.

IE – A8h Interrupt Enable: Il registro IE (indirizzabile a bit) permette l'abilitazione o la disabilitazione degli interrupt. Il bit più significativo posto a 0 disabilita globalmente la

ricezione di tutti i segnali di interrupt dal parte del microcontrollore; i seguenti 7 bit meno significativi di IE sono usati per abilitare o disabilitare gli interrupt individualmente (ma l'impostazione del bit più significativo a zero ha la precedenza).

IP – B8h Interrupt Priority: Il registro IP permette di cambiare la priorità di ciascun interrupt, impostandola a bassa (0) oppure alta priorità (1). Un interrupt può interrompere solo un altro interrupt a priorità più bassa. Per esempio, configurando l'8051 in modo tale che tutti gli interrupt siano a bassa priorità ed eccezione di quello relativo alla porta seriale, esso può sempre sospendere il sistema anche se c'è un altro interrupt attivo. Inoltre nel momento in cui è attivo l'interrupt della porta seriale nessun altro interrupt è in grado di sospendere la routine in corso.

4.6.2.2 SFR generici

Sono i registri segnati in verde in fig. 4.1

Timer 0 (TL0/TH0 - 8Ah/8Bh); Timer 1 (TL1/TH1 – 8ch/8dh): Sono i registri di conteggio a 16 bit (coppie a 8 bit) dei contatori/timer interni. Lo stato dei due timer 0 e 1 è leggibile in questi registri. Il registro TMOD configura il loro comportamento (modo operativo, ovvero quando devono incrementare il loro valore).

Nota

I timer dell'8051 standard contano sempre in avanti (anche se la funzionalità di conteggio all'indietro è spesso disponibile)

SBUF – 99h Serial Buffer: L'invio o la ricezione di dati dalla porta seriale del microcontrollore avviene semplicemente scrivendo o leggendo su questo registro. Ogni valore scritto nel registro SBUF è inviato sulla porta seriale dal pin TXD. Allo stesso modo, ogni valore ricevuto dalla seriale attraverso il pin RXD è 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.

4.6.2.3 SFR porte 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 allo stato alto/basso o leggerne il suo valore sono usati i registri SFR segnati in verde in fig. 4.1.

L'accesso a questi registri può avvenire con operazioni bit a bit (ovvero con le istruzioni **SETB** e **CLR**)

P0 – B8h Porta 0; P1 – 90h Porta 0; P2 – A0h Porta 0; P0 – B0Porta 0: Questi registri rappresentano le quattro porte P*n* di input/output. Decisa una porta, a ciascun bit del registro SFR corrisponde un pin del microcontrollore. Per la porta 0, il bit 0 è il pin P0.0, il bit 7 è il pin P0.7. L'impostazione a "1" di un bit di questo registro equivale a forzare ad un livello alto il corrispondente pin di I/O, mentre portarlo a "0" equivale a forzarlo al livello basso.

Nota

Sebbene l'8051 disponga di 4 porte di I/O se l'impianto hardware permette l'uso di una RAM esterna oppure della ROM per il codice, le porte `P0` a `P2` non possono essere utilizzate poiché in questa modalità di funzionamento il microcontrollore "occupa" queste porte per indirizzare la memoria esterna. Rimangono libere solo le porte `P1` e `P3`.

5.
ACCESSO SERIALI

5.1 La porta seriale

Il microcontrollore 8051 ha la importante caratteristica di implementare al suo interno una UART, conosciuta anche come porta seriale, che permette di effettuare comunicazioni *full duplex*.

La parte di ricezione contiene un buffer che permette di ricevere un secondo byte mentre il primo non è stato ancora letto. I registri di ricezione e trasmissione sono entrambi mappati all'indirizzo 98H (registro SBUF). Una scrittura in SBUF carica il registro di trasmissione, mentre una lettura in SBUF accede al registro di ricezione.

L'uso di tale periferica di I/O è molto semplice: l'unica operazione da effettuare è quella di configurare la porta seriale con il modo operativo ed il baud rate. Una volta configurata la porta è sufficiente leggere o scrivere il registro SFR SBUF per ricevere o inviare dei dati in linea.

Inoltre il microcontrollore segnalerà automaticamente se la trasmissione o la ricezione di un byte sarà terminata con successo, svincolando il programmatore dal compito della trasmissione a livello di bit con un evidente risparmio di codice e tempo di elaborazione.

5.2 Configurazione del modo di funzionamento

La configurazione della porta seriale (numero bit della parola da trasmettere, sorgente del baud rate) avviene agendo sul registro SCON *"Serial Control*", in particolare agendo sui suoi primi 4 bit (bit da 4 a 7).

Il significato di ogni suo singolo bit è riportato nella seguente tabella:

n#	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 è stato trasmesso.
0	**RI**	98h	Receive Flag. Settato quando un byte è stato ricevuto.

Tab. 5.1 – Registro SCON: funzione dei singoli bit

Inoltre le possibili combinazioni dei bit SM0 e SM1 sono:

SM0	SM1	Modo seriale	Funzione	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 (*)

Tab. 5.2 – Registro SCON: funzione dei bit SM0/SM1

Nota

(*) Il baud rate indicato in questa tabella è raddoppiato se PCON.7 (SMOD) è settato.

I bit SM0 e SM1 permettono di impostare i quattro modi seriali supportati, selezionando sia il modo operativo che la maniera con la quale viene calcolato il baud rate. Nei modi 0 e 2 il baud rate è fisso e basato sulla frequenza dell'oscillatore principale. Nei modi 1 e 3 il baud rate è variabile e dipende dalla frequenza con la quale scatta l'overflow del timer 1.

Il successivo bit SM2 è un flag che permette la "comunicazione multiprocessore", utile in alcune applicazioni che usano funzioni avanzate di comunicazione seriale. Normalmente è opportuno porre questo bit a 0, in modo che alla ricezione di un byte l'8051 attivi il flag "RI" (Receive Interrupt) che segnala al programma che un dato è stato ricevuto e quindi può essere processato. Nel caso SM2 sia impostato al valore 1, il flag "RI" si attiva soltanto se il nono bit ricevuto ha il valore pari ad "1". In tal caso se SM2 è attivo e il nono bit ricevuto è zero il flag "RI" non viene settato.

Il seguente bit REN “Receiver Enable” abilita la ricezione della porta seriale. È necessario quindi impostarlo a “1” per poter utilizzare questa periferica di I/0.

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 è 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 è preso da TB8.

Il funzionamento del bit RB8 è la controparte del bit TB8 usata 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” è usato per segnalare la fine della trasmissione di un byte. Ciò si rende necessario per il fatto che, quando viene scritto un byte in SBUF, la porta seriale impiega un certo periodo di tempo per inviare il dato. È opportuno un controllo di questo bit prima di inviare un nuovo dato sulla seriale per evitare che il programma lo scriva prima che il precedente sia stato completamente inviato, con il rischio di inviare dati non validi. Il bit TI serve a segnalare al programma quando il byte è stato completamente trasmesso e il buffer di trasmissione è di nuovo pronto per inviare un altro byte.

Il bit RI “Receive Interrupt” è la controparte del bit TI usata per la ricezione. Indica che un byte è stato ricevuto dalla porta seriale. In questo caso il programma applicativo deve leggere immediatamente il valore da SBUF prima che un altro byte possa essere ricevuto.

La porta seriale può operare in quattro modi distinti:

- **Modo 0** – I dati seriali sono trasferiti in ingresso e in uscita tramite la linea RXD. La linea TXD emette il clock utilizzato dal registro a scorrimento per leggere/emettere i dati seriali. I dati trasmessi sono ad 8 bit; il primo bit ad essere inviato è l’LSB. La velocità in baud è fissata ad 1/12 della frequenza dell’oscillatore.

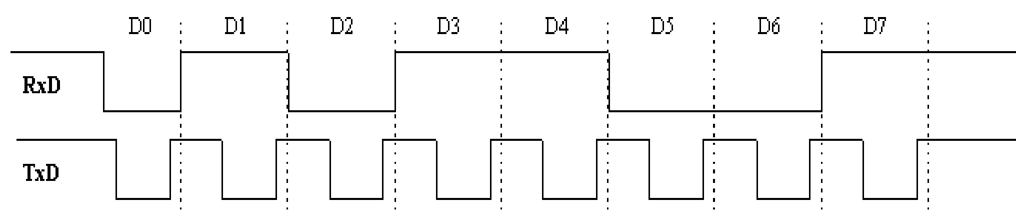


Fig. 5.1 – Funzionamento seriale: modo 0

- **Modo 1** – Comunicazione a 10 bit, 1 bit di start (0), 8 bit di dati (l’LSB è sempre il primo) ed 1 bit di stop (1). La trasmissione avviene tramite la linea TXD, la ricezione tramite la linea RXD. In ricezione il bit di stop viene scritto nel bit RB8 del registro SCON. La velocità di comunicazione è variabile.

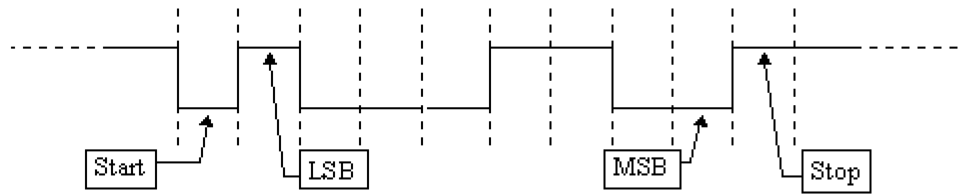


Fig. 5.2 – Funzionamento seriale: modo 1

- **Modo 2** – Comunicazione ad 11 bit, 1 bit di start (0), 8 bit di dati (LSB per primo), 1 bit programmabile dall'utente (ad esempio per la parità) ed 1 bit di stop (1). La trasmissione avviene tramite la linea TXD, la ricezione tramite la linea RXD. In trasmissione il nono bit viene letto dal bit TB8 di SCON. In ricezione il nono bit viene scritto nel bit RB8 del registro SCON. La velocità in baud è programmabile ad 1/32 o 1/64 della frequenza dell'oscillatore.

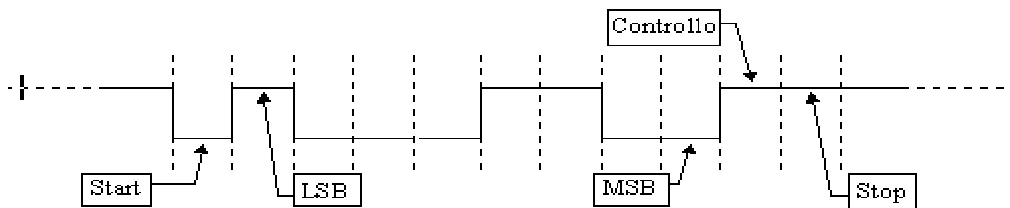


Fig. 5.3 – Funzionamento seriale: modo 2

- **Modo 3** – Comunicazione a 11 bit, 1 bit di start (0), 8 bit di dati (LSB per primo). 1 bit programmabile dall'utente (ad esempio per la parità) ed 1 bit di stop (1). La trasmissione avviene tramite la linea TXD, la ricezione tramite la linea RXD.

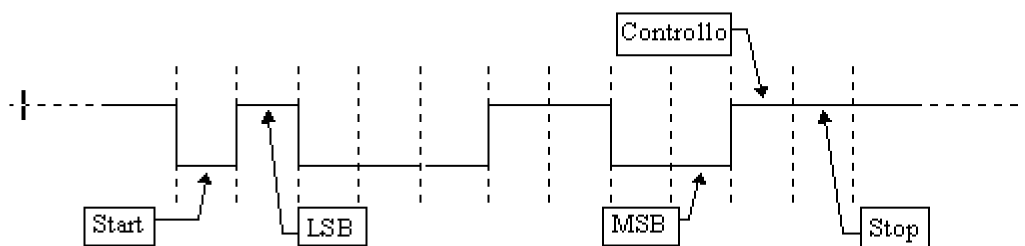


Fig. 5.4 – Funzionamento seriale: modo 3

5.3 Configurazione del baud rate

Questa operazione è necessaria solo nei modi di funzionamento 1 e 3, poiché nei modi 0 e 2 il baud rate è fisso, data la frequenza di oscillazione del quarzo del microcontrollore.

Modo O e **2** – Il baud rate è sempre ricavato dividendo la frequenza di clock principale per 12. Questo significa che se usate un quarzo a 11,059 MHz il baud rate sarà pari a 921.583 baud. Nel modo 2 invece la il baud rate è sempre ricavato dividendo la frequenza di clock principale per 32, e quindi utilizzando un quarzo come quello dell'esempio precedente il baud rate sarà pari a 345.594 baud.

Modo 1 e **3** – Il baud rate dipende dalla frequenza con la quale il timer 1 raggiunge l'overflow. Il modo più comune per ottenere ciò è quello di configurare il timer 1 in modo 2 (8 bit con *auto-reload*) ed impostare il valore di reload in `TH1`. In questo modo il timer 1 andrà periodicamente in overflow generando il baud rate.

La seguente equazione permette di calcolare il valore che deve essere scritto in `TH1` per un dato baud rate:

$$TH1 = 256 - \frac{fq/384}{baud}$$

in cui fq è la frequenza del microcontrollore e $baud$ è il baud rate desiderato.

Nel caso sia impostato `PCON.7`, il baud rate raddoppia e l'equazione diventa:

$$TH1 = 256 - \frac{fq/192}{baud}$$

Esempio

Impostare il baud rate a 19.200 baud con un quarzo a 11,059 MHz

`TH1` = 256 – [(fq / 384) / $baud$]
= 256 – [(11.059.000 / 384) / 19.200]
= 256 – [28.799 / 19.200]
= 256 – 1.5 = 254.5

Per ottenere 19.200 baud dovremmo impostare `TH1` a 254.5, infatti settandolo a 254 si otterranno 14.400 baud, mentre a 255 il baud rate sarà di 28.800 baud.

Si rende quindi necessario impostare il bit `PCON.7` (`SMOD`) a 1. In questo modo il baud rate raddoppia e usando la seconda equazione:

`TH1` = 256 – [(fq / 192) / $baud$]
= 256 – [(11059000 / 192) / 19200]
= 256 – [(57699/ 19200]
= 256 – 3 = 253

ottenendo un numero intero utilizzabile.

In tabella viene mostrato l'utilizzo del timer 1 per generare baud rate usati comunemente nelle trasmissioni seriali.

Timer 1				
Baud rate	Clock (MHz)	SMOD	Modo	Valore reload
1M (Modo 0)	12	-	-	-
375 k (Modo 2)	12	1	-	-
62,5 k (Modo 1,3)	12	1	2	FFh
19.2 k	11,059	1	2	FDh
9,6 k	11,059	0	2	FDh
4,8 k	11,059	0	2	FAh
2,4 k	11,059	0	2	F4h
1,2 k	11,059	0	2	E8h
137,5	11,986	0	2	1Dh
110	6	0	2	72h
110	12	0	1	FEh

Tab. 5.3 – Generazione dei principali baud rate

5.4 Scrittura di un dato sulla seriale

Per inviare un byte sulla porta seriale è sufficiente scriverlo sul registro SBUF (99h).

Esempio

Per inviare la lettera “A” useremo la seguente istruzione

```
        MOV SBUF, #'A'
```

e l’8051 inizia a trasmettere il carattere attraverso la porta seriale. La trasmissione non è istantanea ma richiede una quantità di tempo prestabilita. Poiché l’8051 non dispone di un buffer di trasmissione aggiuntivo prima di tentare l’invio del prossimo carattere, è necessario un automatismo di segnalazione che confermi che il carattere sia stato inviato.

L’8051 segnale quando la trasmissione è stata completata settando il bit TI di SCON.

Un codice più completo quindi prevede alcuni controlli in più:

```
        CLR TI              ; Azzera TI (TI=0)

        MOV SBUF, #?A?      ; Invio carattere

        JNB TI, $           ; Attendi che TI sia settato (TI=1)
```

Con l’ultima istruzione che indica al microcontrollore di rimanere all’interno dell’istruzione stessa finché la condizione si verifica. Il simbolo “$” è utilizzato nella maggior parte dei compilatori assembler (compreso l’A51) per indicare l’indirizzo dell’istruzione in corso.

5.5 Lettura di un dato sulla seriale

Per leggere un byte sulla porta seriale è sufficiente leggere il valore dal registro SBUF (99h) quando il flag RI di SCON è stato attivato.

Esempio

Per leggere un carattere si usa la seguente istruzione

```
JNB RI, $          ; Attendi che RI sia settato (RI=1)

MOV A, SBUF        ; Lettura carattere
```

con la prima istruzione il microcontrollore rimane all'interno dell'istruzione stessa finché l'8051 provvede (automaticamente) a settare RI a 1 non appena il carattere sia completamente ricevuto. Appena RI si attiva, la condizione diventa falsa e il programma procede con l'istruzione successiva.

6.
TIMER

L'8051 standard dispone di due registri timer/contatori a 16 bit, utilizzati generalmente per svolgere tre funzioni:

- Tenere traccia del tempo totale trascorso e/o calcolare il tempo trascorso tra due eventi
- Contare il numero degli eventi
- Generare il baud rate per la porta seriale (caso trattato nel paragrafo precedente).

I timer possono essere configurati, controllati e letti in maniera indipendente: la funzione di conteggio avviene poiché il microcontrollore automaticamente incrementa il loro valore.

In funzionamento *timer* il registro è incrementato ogni ciclo macchina. Poiché un ciclo macchina consiste di 12 periodi di clock, il conteggio avviene ad 1/12 della frequenza dell'oscillatore che al massimo può essere di 12 MHz.

In funzionamento *contatore*, il registro è incrementato per ogni transizione da 1 a 0 del corrispondente terminale di ingresso (T0 e T1). Poiché il riconoscimento di una transizione richiede due cicli macchina, la massima frequenza di conteggio è di 1/24 della frequenza del clock.

6.1 “Interval timer”: misurare il tempo

Quando un timer è usato per misurare il tempo è anche chiamato “interval timer” poiché esso misura l’intervallo temporale tra due eventi. Esso viene incrementato di uno ogni ciclo macchina, ovvero ogni 12 impulsi di clock.

Perciò un timer attivo è incrementato (ipotizzando un quarzo a 11.059 MHz):

11 059 000 / 12 = 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 si possono calcolare i secondi trascorsi:

50 000 / 921 583 = 0.0542 secondi

Supponendo di voler sapere quante volte il timer è incrementato ogni 0,05 secondi è possibile calcolare:

0.05 * 921 583 = 46 079

Il risultato indica che si impiegano 0.05 secondi (un ventesimo di secondo) per contare da 0 a 46 079.

Volendo eseguire un evento ogni secondo è sufficiente che il timer conti da 0 a 46.079 per venti volte; a questo punto eseguite le azioni richieste dall’evento, si resetta il timer e si aspetta che completi di nuovo il conteggio altre venti volte. In questa maniera si esegue effettivamente l’evento una volta al secondo con una precisione al ventesimo di secondo.

6.1.1 La configurazione e inizializzazione dei timer

Due registri SFR, TCON e TMOD, sono condivisi dai due timer per la loro regolazione. Inoltre ogni timer dispone di due SFR dedicati (TH0/TL0 e TH1/TL1) che contengono il valore attuale del contatore. Tutti i registri sono anche raggiungibili tramite gli indirizzi riportati nella seguente tabella.

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

Tab. 6.1 – I registri SFR per l’uso dei timer

Prendendo ad esempio il Timer 0 il suo valore è contenuto nei due registri TH0 e TL0, che si possono ritenere rispettivamente il byte più 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 contiene il valore 3 (decimale) e TL0 il valore 232 (decimale), infatti

valore del timer 0 = TH0 * 256 + TL0

Il massimo valore che un timer di 16 bit può assumere è 65.535 (decimale). Nel momento in cui viene raggiunto tale valore un ulteriore impulso di clock lo farà giungere in *overflow*.

6.1.1.1 Il registro TMOD

Il modo operativo (timer oppure contatore) di ambedue i timer si configura agendo sul registro TMOD. I 4 bit più 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:

Registro TMOD (89h)			
Bit	Nome	Spiegazione della funzione	Timer
7	**GATE1**	Quando questo bit è settato, il timer 1 è attivo solo quando il pin P3.3 è nello stato alto. Se tale bit è resettato il timer 1 sarà svincolato dallo stato del pin P3.3.	1
6	**C/T1**	Quando questo bit è settato il timer 1 conterà il numero degli eventi sul pin T1 (P3.5). Se il bit è resettato il timer 1 verrà 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 è settato, il timer 0 è attivo solo quando il pin P3.2 è nello stato alto. Se tale bit è resettato il timer 0 sarà svincolato dallo stato del pin P3.2.	0
2	**C/T0**	Quando questo bit è settato il timer 0 conterà il numero degli eventi sul pin T1 (P3.5). Se il bit è resettato il timer 0 verrà 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

Tab. 6.2 – I timer: registro TMOD

Le quattro combinazioni dei modi operativi avvengono agendo sui bit T0M0 e T0M1 per il Timer 0; T1M0 e T1M1 per il Timer 1 come riportato dalla seguente tabella:

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

Tab. 6.3 – I timer: registro TMOD, bit TxMx

- **Modo 0** – *Timer a 13 bit*. Generalmente questo modo di funzionamento non è più utilizzato in nuovi sviluppi: è un retaggio del passato che è stato mantenuto per ottenere la compatibilità tra l'8051 con il suo predecessore: l'8048. In questa modalità di funzionamento, TLx conta da 0 a 31. Quando TLx è incrementato da 31, esso si resetta e incrementa THx. Perciò 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 può contare solo fino ad un valore pari a 8192. Se si carica il timer con il valore 0, esso va in overflow e quindi a 0 dopo 8192 cicli macchina.

- **Modo 1** – *Timer a 16 bit*. Questo è un modo molto utilizzato. TLx è incrementato da 0 a 255. All'overflow di TLx, THx viene incrementato di 1. Tenuto conto che è un timer a 16 bit, esso può assumere 65536 valori distinti. Se si carica il timer con il valore 0, esso va in overflow e quindi a 0, dopo 65536 cicli macchina.

- **Modo 2** – *Timer ad 8 bit con auto-reload*. Quando il timer è configurato in modo 2, THx contiene il valore che deve essere caricato in TLx quando va in overflow. TLx inizia a contatore 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. Il modo 2 è usato molto spesso per generare il baud rate della porta seriale.

  Esempio

  Se TH0 contiene il valore FDh e TL0 il valore FEh il conteggio procede 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

Tab. 6.4 – I timer: esempio di conteggio

Si noti 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. Il vantaggio di usare tale modo di funzionamento è che l'hardware del microcontrollore permette ad esempio che il timer assuma dei valori compresi tra 100 e 255 senza essere costretti a verificare da programma quando il timer va in overflow e quindi settarlo al valore 100, operazione che richiederebbe tempo e non garantirebbe una elevata accuratezza.

- **Modo 3** – *Timer in splitted mode* (timer separati). Se il timer 0 è configurato per lavorare in questa modalità, si trasforma in due timer a 8 bit separati; 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 è configurato in modo split, il vero Timer 1 (cioè TH1 e TL1) può essere configurato nei modi 0, 1, 2 come sempre, ma non può essere abilitato/disabilitato poiché i suoi bit di controllo sono assegnati a TH0. Allora, il vero Timer 1 funziona in maniera libera e si incrementa ad ogni ciclo macchina senza condizioni. L'unico vero uso di questa modalità è 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.

6.1.1.2 Il registro TCON

Solo quattro bit del registro TCON sono relativi all'uso dei timer. Gli altri sono relativi agli alla gestione degli interrupt sono discussi in seguito. Questo registro può essere indirizzato a bit (tramite le istruzioni SETB e CLR): l'indirizzo di ognuno è specificato nella seguente tabella:

Registro TCON (88h)				
Bit	Nome	Indirizzo a Bit	Spiegazione della funzione	Timer
7	**TF1**	8Fh	**Timer 1 Overflow**. Questo bit è settato quando il timer 1 è andato in overflow	1
6	**TR1**	8Eh	**Timer 1 Run**. Quando questo bit viene settato il timer 1 è abilitato, altrimenti è fermo.	1
5	**TF0**	8Dh	**Timer 0 Overflow**. Questo bit è settato quando il timer 0 è andato in overflow	0
4	**TR0**	8Ch	**Timer 0 Run**. Quando questo bit viene settato il timer 0 è abilitato, altrimenti è fermo.	0

Tab. 6.5 – I timer: registro TCON

Generalmente infatti, quando si abilita o disabilita un timer, non si vuole modificare il valore degli altri bit di TCON, allora è possibile sfruttare il fatto che tale registro è indirizzabile a bit.

Per scoprire se un timer ha raggiunto il massimo del conteggio e quindi scatta l'overflow si sfrutta il fatto che in questa situazione il microcontrollore lo riporta automaticamente a zero e setta il bit TFx corrispondente nel registro TCON. Per cui quando il bit TF0 è settato vuol dire che il timer 0 ha raggiunto l'overflow e alla stessa maniera se TF1 è settato vuol dire che il timer 1 è tornato a zero.

Esempio

Impostazione del Timer 0 in Modo 1 (timer a 16 bit).

Per la configurazione del timer 0 si devono trattare i 4 bit meno significativi di TMOD: resettando T0M1 e settando T0M0 si seleziona il timer 0 in modo1; i primi due bit GATE0 e C/T0 vanno ambedue posti a 0 poiché si vuole che il timer sia indipendente da pin esterni. La seguente istruzione pone a 1 il bit 0 di TMOD, eseguendo la configurazione esaminata:

```
MOV TMOD,#01h      ; TMOD = 000000001
```

Ora è necessario abilitare il timer, agendo sul registro TCON, indirizzabile a bit:

```
SETB TR0           ; Abilita il timer
```

Il timer inizia a contare in avanti incrementando il suo valore ad ogni ciclo macchina (ogni 12 impulsi di clock).

6.2 Timer come contatori di numero eventi o durata singolo evento

I timer dell'8051 possono essere utilizzati non solo per tener traccia del tempo trascorso ma anche come contatori di eventi semplicemente configurando un bit addizionale del registro TCON: C/T che seleziona la funzione contatore (C/T=1) oppure timer (C/T=0).

Settando il bit 2 di TCON (TCON.2) chiamato C/T0, (ovviamente la controparte per il timer 1 è TCON.6 chiamato C/T1) il timer 0, invece di incrementarsi ad ogni ciclo macchina, effettua il *monitoring* della linea P3.4. In questa maniera si incrementa ogni volta che la linea passa dal valore alto a quello basso. A questa linea chiaramente è possibile collegare sensori o trasduttori che monitorano ciò che desideriamo conteggiare.

Si noti che il microcontrollore è in grado di controllare 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 è troppo elevata, l'8051 non riesce a conteggiare il numero di eventi in maniera corretta. Più precisamente, l'8051 riesce a contare un numero di eventi ad un massimo di un ventiquattresimo della frequenza di clock. Ciò 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 MHz * 1/24 = 500.000).

L'8051 può efficacemente essere utilizzato per misurare la durata di un evento. Utilizzando a questo scopo il timer 0 si agisce sul bit chiamato GATE0 (o GATE1 per il timer 1) del registro TMOD.

Impostando questo bit a zero il timer continua a girare indipendentemente dallo stato di eventuali segnali su pin dedicati del microcontrollore esterni. Tuttavia settandolo il microcontrollore controllo lo stato del pin INT0 (P3.2) ed esegue il conteggio solo quando il pin P3.2 si trova nello stato alto, bloccandosi in caso contrario.

6.3 La lettura dei timer

6.3.1 Modo 0 e 1 (timer a 13, 16 bit)

La lettura di un timer in modo a 13 o 16 bit la lettura è complessa poiché il valore è contenuto in due registri da 8 bit e si potrebbe verificare il caso che -mentre si sta leggendo il valore del byte meno significativo del timer e questo valore è pari a 255- si legge un valore errato del byte più significativo che si sta incrementando in quel momento.

Le soluzioni proponibili sono 2. La prima, più semplice, è quella di congelare momentaneamente il timer (CLR TR0), leggerne il valore e riabilitarlo subito dopo (SETB TR0). In questo modo la lettura del timer fermo non comporta problemi, tuttavia questo metodo è utilizzabile solo se è tollerabile per il nostro programma un conteggio del tempo non precisissimo (il timer rimane fermo per alcuni colpi di clock).

La seconda soluzione - riportata nel codice seguente - consta nel procedimento di lettura prima del byte più significativo seguita da quello meno significativo e poi una rilettura del byte alto confrontandolo con quello letto precedentemente: solamente se i due valori sono coincidenti il valore letto è esatto.

```
LOOP: MOV A, TH0
      MOV R0, TL0
      CJNE A, TH0, LOOP     ; confronta A e TH0, ripete il codice se non uguali
```

6.3.2 Modo 2 e 3 (timer a 8 bit)

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

7.
INTERRUPT

7.1 Le sorgenti di interrupt

L'*interrupt* è un evento che interrompe la normale esecuzione di un programma. Più che deviare il flusso sequenziale del programma stesso esso piuttosto effettua un "congelamento" richiamando una subroutine – l'*interrupt handler* – che è eseguita soltanto quando si manifesta l'avvenimento che ha scatenato la richiesta di interrupt.

Questa caratteristica permette all'8051 di fornire l'illusione del *multi-tasking* anche se esso esegue una sola istruzione alla volta.

Un interrupt è attivato quando il corrispondente evento si presenta. L'8051 prevede cinque sorgenti di interruzione distinte. Esse sono>

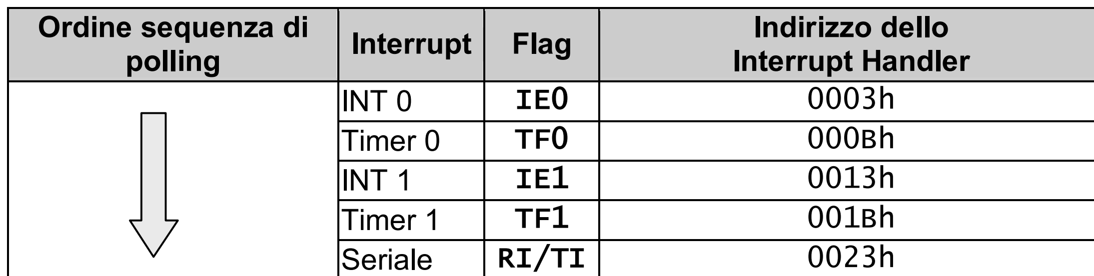
- Overflow del Timer 0
- Overflow del Timer 1
- Ricezione/trasmissione di un carattere dalla seriale
- Evento esterno 0 (INT 0).
- Evento esterno 1 (INT 1).

Le ultime due sono le linee di richiesta interruzione a disposizione dell'utente.

L'8051 può essere configurato per gestire un interrupt handler per ognuno di questi eventi. Se gli interrupt sono abilitati l'8051 sospende momentaneamente la normale esecuzione del programma utente ed esegue la subroutine di interrupt che esegue le funzioni richieste dall'evento e restituisce il controllo al microcontrollore dal punto in cui era stato fermato, in maniera tale che l'esecuzione del programma possa proseguire come se non fosse mai stato interrotto.

Gli interrupt risultano molto utili nel caso della gestione di processi asincroni. Inoltre essi evitano che si sprechino dei cicli macchina ad aspettare un evento che non avviene molto frequentemente. Infatti liberano il programmatore dal compito di controllare la condizione che scatena l'interrupt stesso. Il microcontrollore esegue il controllo automaticamente e quando la condizione risulterà vera, richiamerà l'interrupt handler, eseguirà il codice in esso contenuto e ritornerà al programma principale.

All'attivazione della richiesta di interruzione il microcontrollore salta a degli indirizzi di memoria predefiniti differenti per ogni tipo di interrupt, come mostrato nella tabella che segue



Ordine sequenza di polling	Interrupt	Flag	Indirizzo dello Interrupt Handler
	INT 0	IE0	0003h
	Timer 0	TF0	000Bh
	INT 1	IE1	0013h
	Timer 1	TF1	001Bh
	Seriale	RI/TI	0023h

Tab. 7.1 – Interrupt: indirizzi e polling

Ad esempio in caso di overflow del timer 0 il programma principale sospende momentaneamente il programma e salta all'indirizzo 0003h dove avremmo dovuto scrivere l'interrupt handler opportuno.

La sequenza con cui sono stati scritti gli interrupt nella prima colonna della tabella rispecchia inoltre l'ordine in cui l'8051 esegue la sequenza di polling sulle cinque sorgenti di interrupt. Ad esempio se l'interrupt del Timer 1 si attiva nello stesso momento di quello esterno INT 0, è quest'ultimo che viene servito per primo e quello relativo alla porta seriale immediatamente dopo.

Nel momento in cui giunge la richiesta di interruzione, l'8051 automaticamente segue queste istruzioni:

- Salvataggio del valore del Program Counter nello stack (`PUSH PC`), a partire dal byte meno significativo.
- Blocco di tutti gli interrupt della stessa o più bassa priorità
- In caso l'interrupt riguarda o un timer o una richiesta esterna, il corrispettivo flag è disattivato (quindi è superfluo resettarlo nel codice del programma!)
- L'esecuzione del programma salta all'indirizzo dell'interrupt handler corrispondente.

Il termine dell'interrupt handler è segnato dall'istruzione `RETI` (RETurn from Interrupt), la quale fa compiere al microcontrollore le seguenti operazioni:

- Preleva due byte dallo stack (`POP PC`) per collocarli nel Program Counter per riprendere l'esecuzione del programma principale.
- Ripristina lo stato dell'interrupt alla sua condizione precedente la chiamata.

Quindi l'istruzione RETI si differenzia dall'istruzione RET semplicemente per il fatto che questa informa la CPU che il sottoprogramma di gestione interruzione è terminato ed è quindi possibile accettare ulteriori richieste di interruzione. Dal punto di vista dell'utilizzo dello stack (recupero dell'indirizzo di ritorno) RET e RETI si comportano in modo identico.

7.2 Abilitazione e configurazione degli interrupt

Per default all'accensione dell'8051 tutti gli interrupt sono disabilitati: benché si verifichi uno dei cinque eventi che possono essere sorgente di interrupt questo non sarà accettato.

Il programmatore deve specificatamente configurare il registro di mascheramento interruzioni IE (A8h) che indica quali interrupt sono abilitati ad essere serviti: ciascun tipo interruzione supportata dispone di un suo bit nel registro che ne permette l'abilitazione.

Registro IE (A8h)			
Bit	Nome	Indirizzo a Bit	Funzione
7	**EA**	AFh	Abilita globalmente gli interrupt
6	**-**	AEh	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

Tab. 7.2 – Gli interrupt: registro IE

Esempio

Abilitazione dell'interrupt del Timer 1.

```
        SETB EA     ; (*)
        SETB ET1
```

(*) Al fine di abilitare effettivamente un qualsiasi interrupt è necessario settare anche EA (bit 7 di IE), oltre all'interrupt specifico che intendiamo attivare. Infatti questo bit che abilita globalmente il microcontrollore alle richieste di interruzione ha la precedenza su quelli di selezione della sorgente. Ovviamente nel caso sia settato saranno servite tutte le richieste proveniente da tutte le sorgenti poste che hanno il proprio biposto a 1.
Se necessario è possibile disabilitare EA prima di entrare in una porzione di codice che ha stringenti criticità temporali e non deve essere interrotta per nessun motivo. Usciti da tale sezione critica il bit può essere riabilitato.

Il programmatore può cambiare l'ordine standard con il quale l'8051 serve gli interrupt avendo a disposizione due livelli di priorità (alto e basso) delle interruzioni.
La priorità degli interrupt è controllata dal registro Interrupt Priority **IP** (B8h).

Registro **IP** (B8h)			
Bit	Nome	Indirizzo a Bit	Funzione
7	-	-	Non definito
6	-	-	Non definito
5	-	-	Non definito
4	**PS**	BCh	Priorità Interrupt Seriale
3	**PT1**	BBh	Priorità Interrupt Timer 1
2	**PX1**	BAh	Priorità Interrupt esterno INT 1
1	**PT0**	B9h	Priorità Interrupt Timer 0
0	**PX0**	B8h	Priorità Interrupt esterno INT 0

Tab. 7.3 – Gli interrupt: registro IP

I principi per l'attuazione di un ordine di priorità ad una richiesta di interrupt sono:

- Nel caso si verifichino contemporaneamente due richieste di interrupt è servito prima quello a priorità più alta; nel caso avessero la stessa priorità sono processati secondo l'ordine della sequenza di polling
- Un'interruzione ad alta priorità non può essere interrotta (nemmeno da un'altra dello stesso tipo)
- Un'interruzione a bassa priorità può essere interrotta da una ad alta priorità
- Un'interruzione a bassa priorità può essere iniziata soltanto quando nessun altro interrupt è attivo.

7.3 Interrupt seriali

Vi sono due flag in grado di attivare gli interrupt seriali: RI e TI. L'interrupt handler deve verificare lo stato di tutti e due per sapere quale dei due (o eventualmente se entrambi) l'ha richiamato. Inoltre sarà compito dello stessa subroutine di interruzione resettare questi flag, operazione che non è automaticamente eseguito dall'8051.

Esempio

```
INT_SERIAL: JNB RI, CHECK_TI  ; Se il flag RI non e' attivo vai a CHECK_TI

            MOV A, SBUF       ; Leggi il buffer in ricezione

            CLR RI            ; Cancella il flag RI

CHECK_TI:   JNB TI, EXIT_INT  ; Se il flag TI non è attivo vai a EXIT_TI

            CLR TI            ; Cancella TI prima di inviare un altro carattere
```

```
                MOV SBUF, #'A'      ; Copia nuovo carattere da inviare nel buffer TX

EXIT_INT:       RETI                ; Ritorna al programma
```

7.4 Scrittura di un interrupt handler

Tutti i vari interrupt sono eseguiti senza preavviso, in maniera trasparente al programma principale. Ogni interrupt deve lasciare il processore nel medesimo stato che esso aveva al momento di iniziare l'interrupt stesso.

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

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

affinché il programma principale trovi la situazione dei suoi registri immutata e quindi non si accorga che è avvenuta l'interruzione.

È indispensabile salvare sempre il contenuto di PSW (a meno che non si sia *assolutamente* sicuri di non modificare nessuno dei suoi bit con le operazioni contenute nell'interrupt handler) e in genere tutti i registri utilizzati nella subroutine di mediante le operazioni di push e pop all'inizio e alla fine dell'interrupt handler.

Nota

Poiché l'indirizzo di R0 dipende dal banco di registri "R" selezionato (potrebbe riferirsi alla RAM interna in posizione 00h o 08h o 10h oppure 18h) l'assembler non permette di eseguire la seguente istruzione:

```
PUSH R0
```

Perciò l'operazione di push è solo possibile facendo riferimento all'indirizzo assoluto del registro stesso. Ad esempio supponendo di usare il set di registri di default l'operazione va scritta:

```
PUSH 00h
```

8.
L’ASSEMBLATORE A51

L’A51 è propriamente un *cross-assembler* in quanto assembla codice sorgente scritto per architettura 8051 in codice oggetto 8051, ma si esegue sui diffusissimi PC della famiglia x86. La suite di sviluppo comprende – oltre all’assemblatore – un Linker (con capacità di *bank interleave* per generare programmi con più di 64kB di codice) e un Object to Hex Converter in grado di convertire i file oggetto assoluti così generati nel formato “Intel Hexadecimal” contenente il codice macchina direttamente interpretabile dall’8051.

Questo file “.hex” può essere poi caricato, attraverso una linea seriale, direttamente nella memoria di programma FLASH presente sulla basetta del microcontrollore. Il programma che permette ciò è FlashTools98, che è suddiviso in due parti distinte.
Una parte (permettetemi di chiamarla impropriamente “client”) è eseguibile su PC Windows e permette la selezione del file .HEX che deve essere caricato nella FLASH. Questa parte si interfaccia con il programma di monitor (il “server”) presente sul primo banco della stessa memoria di programma, che riceve via seriale in programma inviato e lo memorizza nel banco numero 2. Quindi il programma utente è caricato su *secondo* banco, in quanto nel primo risiede il programma monitor, che non deve *assolutamente* essere cancellato, pena l’impossibilità di caricare altri programmi sulla basetta. L’utilizzo approfondito di FlashTools98 verrà trattato inseguito, nella sezione “10.3 Note sul software utilizzato”.

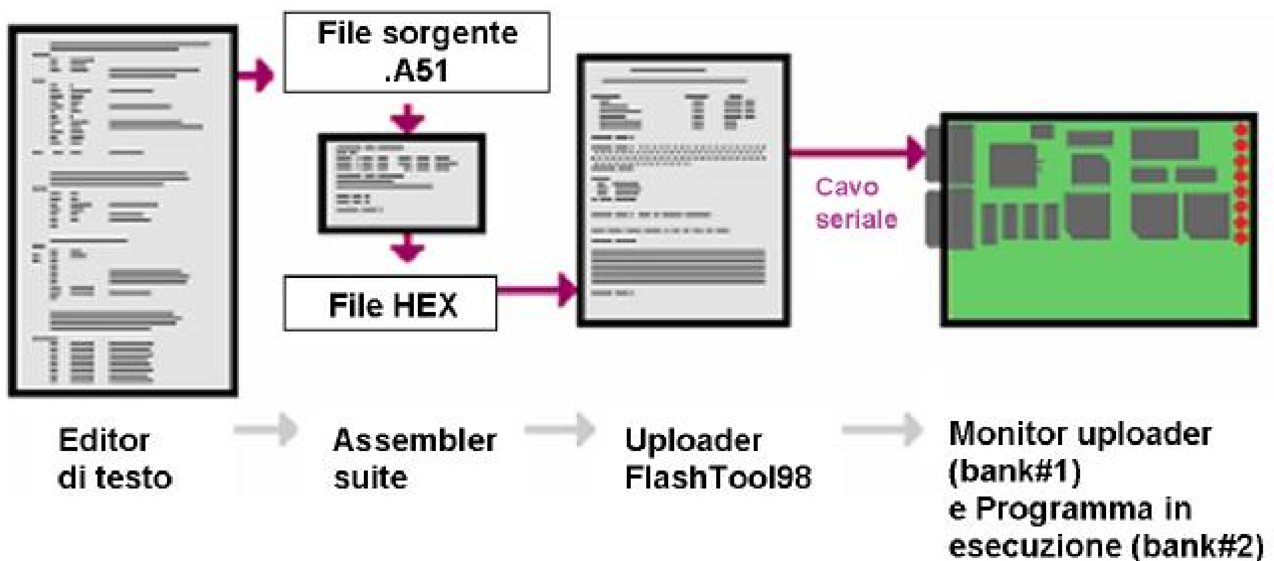


Fig. 8.1 – Il processo di scrittura di un programma

Un programma Assembler è composto da istruzioni (che generano un'istruzione macchina, e saranno trattate nella sezione "Set d'istruzioni dell'8051") e direttive o *pseudo-istruzioni*. Questi sono dei comandi per l'assemblatore, utili per la definizione di variabili e costanti, etc…

Il formato delle istruzioni supportate dall'A51 nel codice sorgente è del tipo:

```
[label]  [opcode[argumenti]]                [;commenti]
```

La `label` è qualsiasi sequenza di caratteri del tipo

! $ % & : ? [\ ] ^ _ ` { | } ~ A-Z a-z

(ricordando che l'A51 è case sensitive).
L'**opcode** può essere o un'istruzione Assembler 8051 o una direttiva per l'assemblatore.

8.1 Le direttive dall'A51

8.1.1 BIT

La direttiva permette la creazioni di variabili di tipo bit. È espressa nella seguente forma:

```
label BIT bit_expression
```

È necessario specificare una label che identificherà in seguito la variabile. In caso contrario si riceve un errore "L" (*label error*).

8.1.2 DB

La direttiva permette la definizione di strutture dati costituite da un byte. I suoi argomenti sono una serie di zero o più espressioni con valore (in decimale) da -128 a 255, separate da una virgola.
Può essere utilizzata per memorizzare stringhe di caratteri, ad esempio

```
myString DB "CAT"
```

corrisponde alla memorizzazione di 3 byte con valore 043H, 041H e 054H.

8.1.3 DS

La direttiva è utilizzabile per riservare un blocco di memoria per la memorizzazione di variabili o altri dati. Questo blocco non è inizializzato in alcun modo, e una lettura a *run-time* senza aver prima proceduto a una inizializzazione mostra in genere numeri casuali.
Ad esempio la seguente istruzione riserva 10 byte chiamandoli "STORAGE".

```
STORAGE DS 10
```

8.1.4 DW

La direttiva permette la definizione di strutture dati costituite da un word (due byte).
La word è posta in memoria in formato H:L (il byte più significativo della word nell'indirizzo più basso; il byte meno significativo della word nell'indirizzo più alto), l'opposto rispetto allo standard Intel. In altre parole, un'espressione tipo

```
DW 01234H
```

piazza 12H all'indirizzo "n" e 34H all'indirizzo "n+1".
Questa direttiva può essere utilizzata per la memorizzazione di uno o due caratteri, ma non per memorizzare sequenze di caratteri.

8.1.5 END

La direttiva segna la fine del programma sorgente. Eventuali righe del programma successive non saranno prese in considerazione.

8.1.6 EQU

La direttiva permette l'assegnazione di uno specifico valore ad una certa label (che deve necessariamente essere specificata). Le costanti definite con questa pseudo-istruzione non possono essere cambiate di valore nel corso del programma. Ad esempio per definire una costante simbolica con nome TWO e valore "2":

```
TWO EQU 1+1
```

8.1.7 IF, ELSE, ENDIF

Le direttive permettono all'assemblatore la scelta di compilare o meno certi blocchi di codice a seconda del risultato di un'espressione di valutazione. Il pseudo-codice `IF` segnala l'inizio di un blocco condizionale. Esso richiede un argomento, il quale se ha valore diverso da zero permette la compilazione del blocco sottostante, la cui fine è segnata dalla direttiva `ENDIF`.
Inoltre grazie al pseudo-codice `ELSE` è possibile assemblare uno dei due blocchi impostati, ma non entrambi.
Le seguenti due sequenze di codice risultano quindi equivalenti:

```
IF EXPRESSION
[…]   ;istruzioni
ELSE
[…]   ;altre istruzioni
ENDIF

TEMP_LAB SET EXPRESSION
IF TEMP_LAB NE 0
[…]   ;istruzioni
ENDIF
IF TEMP_LAB EQ 0
[…]   ;altre istruzioni
ENDIF
```

8.1.8 INCL

La direttiva permette di INCLudere il contenuto di un altro file nel file corrente durante la procedura di assembramento. Il nome del file da includere è specificato come una normale stringa costante. Ad esempio per includere il file `const.def`:

```
INCL “const.def”
```

8.1.9 ORG

La direttiva permette di assegnare il program counter (PC) ad un particolare valore, sovrascrivendo il valore di default 0000H.

8.1.10 REG

La direttiva ha un funzionamento simile a `EQU` ad eccezione del fatto che l'argomento deve essere un registro (`R0` … `R7`). L'espressione permette quindi di "rinominare" un registro, permettendone anche i rispettivi metodi di indirizzamento.
Ad esempio l'espressione "`MOV @R0, A`" può essere espressa anche come:

```
TEMP REG R0
MOV @TEMP, A
```

8.1.11 SET

La direttiva ha un funzionamento simile a EQU ad eccezione del fatto che permette il cambiamento del valore, durante il corso del programma, di una variabile simbolica assegnata con questa pseudo-istruzione. Ciò permette di scrivere un'espressione (a solo scopo esemplificativo, visto che è chiaramente inutile…) del tipo:

```
COUNT SET 1
COUNT SET 2
COUNT SET 3
```

9.
SET D'ISTRUZIONI DELL'8051

Tutti i chip della famiglia 8051 supportano un set base di 111 istruzioni, tra istruzioni aritmetiche, logiche, di trasferimento dati e di salto.

L'esecuzione di una di queste istruzioni avviene in 1 o 2 cicli macchina, le (più complicate) istruzioni di divisione (`DIV`) e di moltiplicazione (`MUL`) richiedono ben 4 cicli macchina.

Un ciclo macchina è rigorosamente costituito da 12 periodi del segnale di clock. Ipotizzando una frequenza di clock di 12 MHz un'istruzione che richiede 1 ciclo macchina è eseguita esattamente in 1 µS.

Notazioni

- **A** - Accumulatore (Registro "A")
- **AB** - Tra l'Accumulatore ed il registro "B"
- **DPTR** - Data Pointer
- **C** - Carry bit
- **R*n*** - Registro (R7-R0) del banco di registri "R" attualmente selezionato
- **@R*n*** - 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 più A
- **@A+PC** - Indirizzamento indiretto tramite la somma di PC più A
- **addr** - Indirizzo della RAM interna
- **#n** - Indirizzamento immediato
- **#ind16** - Indirizzamento immediato (su 16 bit)
- **bit**- Indirizzo del bit
- **/bit** - Indirizzo del bit (prendi il bit negato)
- **rel** - Indirizzo relativo (da +128 a -127)

9.1 Istruzioni aritmetiche

Le istruzioni aritmetiche sono mostrate in tabella, che indica anche le modalità di indirizzamento che possono essere usate per accedere agli operandi.

Cod. mnemonico	Operazione	Dir	Ind	Reg	Imm	Imp
ADD A, <byte>	A = A + <byte>	X	X	X	X	
ADDC A, <byte>	A = A + <byte> + C	X	X	X	X	
SUBB A, <byte>	A = A - <byte> - C	X	X	X	X	
INC A	A++					X
INC <byte>	<byte>++	X	X	X		
INC DPTR	DPTR++					X
DEC A	A--					X
DEC <byte>	<byte>--	X	X	X		
MUL AB	BA = B·A					X
DIV AB	A = A div B					
	B = A mod B					X
DA	Decimal Adjust (BCD)					X

Tab. 9.1 – Instruction set: istruzioni aritmetiche

Le istruzioni **INC** <byte> e **DEC** <byte> permettono di incrementare e decrementare direttamente il contenuto della RAM interna senza passare attraverso l'accumulatore.

L'istruzione **MUL AB** effettua la moltiplicazione fra il contenuto dell'accumulatore e quello del registro B; la parte più significativa del risultato viene scritta nel registro B, la parte meno significativa nell' accumulatore.

L'istruzione **DIV AB** divide il contenuto dell'accumulatore per il contenuto del registro AB; il quoziente viene scritto in accumulatore e il resto nel registro B.

L'istruzione **DA A** serve quando si utilizza l'aritmetica in BCD. Utilizzando la rappresentazione BCD le istruzioni ADD e ADDC dovrebbero essere sempre seguite da un'istruzione DA A; ciò assicura che il risultato sia ancora in BCD. DA A non deve essere utilizzata per convertire un numero da binario a BCD.

9.2 Istruzioni logiche

Le istruzioni logiche sono illustrate in tabella

Cod mnemonico	Operazione	Dir	Ind	Reg	Imm	Imp
ANL A, <byte>	A = A and <byte>	X	X	X	X	
ANL <byte>, A	<byte> = <byte> and A	X				
ANL <byte>, #n	<byte> = <byte> and n	X				
ORL A, <byte>	A = A or <byte>	X	X	X	X	
ORL <byte>, A	<byte> = <byte> or A	X				

ORL <byte>, #n	<byte> = <byte> or n	X				
XRL A, <byte>	A = A xor <byte>	X	X	X	X	
XRL <byte>, A	<byte> = <byte> xor A	X				
XRL <byte>, #n	<byte> = <byte> xor n	X				
CLR A	Azzera ACC					X
CPL A	Complementa ACC					X
RL A	Ruota ACC a sinistra					X
RLC A	Ruota ACC a sinistra utilizzando CY					X
RR A	Ruota ACC a destra					X
RRC A	Ruota ACC a destra utilizzando CY					X
SWAP A	Scambia i due semibyte (nibble) di ACC					X

Tab. 9.2 – Instruction set: istruzioni logiche

Le operazioni booleane possono essere effettuate su qualunque byte della RAM interna o della SFR area, utilizzando l'indirizzamento diretto senza passare attraverso l'accumulatore.

Esempio

Effettuare un AND tra il contenuto della locazione 20h della RAM interna e il numero 80h:

```
ANL 20h, #80h
```

Il risultato è posto ancora nella locazione 30h, senza utilizzare l'accumulatore.

9.3 Istruzioni trasferimento dati

9.3.1 RAM interna

Le istruzioni di trasferimento dati con la RAM interna sono illustrate in tabella.

Cod.mnemonico	Operazione	Dir	Ind	Reg	Imm
MOV A, <sorg>	A = <sorg>	X	X	X	X
MOV <dest>, A	<dest> = A	X	X	X	
MOV <dest>, <sorg>	<dest> = <sorg>	X	X	X	X
MOV DPTR, #ind16	DPTR = ind16				X
PUSH <sorg>	Incrementa SP e inserisci <sorg> nello stack	X			
POP <dest>	Preleva <dest> dallo stack e decrementa SP	X			
XCH A, <byte>	Scambia A con <byte>	X	X	X	
XCHD A, @R*n*	Scambia i semibyte bassi di A e del dato puntato da R*n* (R*n* può essere RO o R1)		X		

Tab. 9.3 – Instruction set: istruzioni trasferimento dati RAM interna

Le indicazioni <dest> e <sorg> specificano rispettivamente le locazioni destinazione e sorgente. Si tenga presente che la SFR area può essere indirizzata solo in modo diretto. L'istruzione **MOV** <dest>, <sorg> permette di effettuare un trasferimento dati senza passare attraverso l'accumulatore.

Per le operazioni con lo stack, lo stack pointer SP dopo un reset punta alla locazione 07H della RAM interna mentre il banco di registri R0-R7, che viene automaticamente selezionato, è il banco numero 0.

Se si vuole utilizzare uno degli altri tre banchi di registri è consigliabile far puntare lo stack pointer SP oltre la zona di memoria occupata dai banchi dei registri (si tenga presente che nell'8051 lo stack cresce verso gli indirizzi alti della memoria).

9.3.2 RAM esterna

Può essere soltanto utilizzata la modalità di indirizzamento indiretto. Si possono utilizzare i registri ad 8 bit R1 ed R0 o il registro a 16 bit DPTR. La tabella mostra le istruzioni che permettono di accedere alla memoria esterna per i dati.

Cod.mnemonico	Operazione
MOVX A, @R*n*	Il dato puntato da R*n* (R1 o R0) è trasferito in A
MOVX @R*n*, A	Il dato contenuto in A è trasferito nella locazione puntata da R*n*
MOVX A, @DPTR	Il dato puntato da DPTR (indirizzo a 16 bit) è trasferito in A
MOVX @DPTR, A	Il dato contenuto in A è trasferito nella locazione puntata da DPTR

Tab. 9.4 – Instruction set: istruzioni trasferimento dati RAM esterna

Utilizzando i registri R0 ed R1 si possono indirizzare solo i primi 256 byte della RAM esterna, con il registro DPTR tutti i 64k di memoria RAM esterna.

Si tenga presente che il microprocessore 8051 distingue tra memoria interna ed esterna grazie al fatto che il codice operativo dell'istruzione MOV è diverso dal codice operativo di MOVX (X sta per eXternal). Quando viene utilizzata l'istruzione MOVX, le linee WR e RD vengono abilitate in modo da attivare la memoria esterna.

Se l'istruzione MOVX non viene utilizzata, queste due linee rimangono inattive e sono utilizzabili come linee di I/O.

9.3.3 Istruzioni per la gestione delle tabelle dati

L'8051 dispone di istruzioni con modalità di indirizzamento indicizzato per la gestione di tabelle nella memoria per i programmi. Queste tabelle di ricerca possono essere solo lette e non modificate e possono risiedere nella memoria per i programmi esterna (nel qual caso ci pensa l'8051 ad attivare il terminale PSEN).

Le istruzioni per la gestione di tabelle sono mostrate in tabella

Cod. mnemonico	Operazione
MOVC A, @A + DPTR	Il dato puntato da (A + DPTR) viene trasferito in A
MOVC A, @A + PC	Il dato puntato da (A + PC) viene trasferito in A

Tab. 9.5 – Instruction set: istruzioni gestione tabelle dati

La prima istruzione MOVC può gestire una tabella a 256 punti di ingresso (256 elementi) numerati da O a 255. La base della tabella deve essere caricata in DPTR mentre l'accumulatore funge da indice nella tabella stessa. L'altra istruzione MOVC opera in modo analogo con la differenza che questa volta è il PC a puntare alla base della tabella. La tabella in questo caso è consultata per mezzo di un sottoprogramma.

Esempio

```
        MOV A, #5

        CALL tabella
```

Il sottoprogramma "tabella" può essere il seguente:

```
        MOVC A, @+PC

        RET

        .............     ; elementi della tabella
```

In questo caso la tabella può avere 255 punti di ingresso (255 elementi) numerati da 1 a 255. Il numero 0 non può essere utilizzato perché coincide con la posizione occupata dall'istruzione RET (indirizzo contenuto in PC durante l'esecuzione di MOVC) e quindi il valore ritornato dalla lettura della tabella sarebbe proprio il codice operativo dell'istruzione RET.

9.4 Istruzioni Booleane

L'8051 contiene un processore completo per le operazioni booleane (cioè su singoli bit). La RAM interna contiene una zona indirizzabile bit a bit (locazioni di memoria da 20H a 2FH, 16 byte per un totale di 128 bit utilizzabili separatamente). Anche i bit delle porte di I/O e di alcuni registri sono indirizzabili separatamente. Per le operazioni booleane il bit di riporto C del registro PSW ha la funzione di accumulatore.

Le istruzioni booleane sono mostrate in tabella

Cod. mnemonico	Operazione
ANL C, bit	C =C and bit
ANL C, /bit	C = C and not bit
ORL C, bit	C = C or bit
ORL C, /bit	C = C or not bit
MOV C, bit	C = bit

MOV bit, C	bit = C
CLR C	C = O
CLR bit	bit = 0
SETB C	C = 1
SETB bit	bit = 1
CPL C	C = not C
CPL bit	bit = not bit
JC rel	Salta se C =1
JNC rel	Salta se C=O
JB bit, rel	Salta se bit = 1
JNB bit, rel	Salta se bit = 0
JBC bit, rel	Salta se bit = 0 e azzera bit

Tab. 9.6 – Instruction set: istruzioni booleane

Gli indirizzamenti sono effettuati solo in modo diretto sia nella RAM per i dati sia nella SFR area (registri). Gli indirizzi per bit vanno da OOH a 7FH per i 128 bit posizionati nelle locazioni di memoria RAM interna da 20H a 2FH.

L'indicazione "rel" nelle istruzioni di salto condizionato indica che il salto viene effettuato in modo relativo rispetto al valore contenuto nel PC. L'offset rispetto al PC ha il formato di un byte contenente un numero con segno. Il campo di valori di salto va quindi da -128 a +127.

9.5 Istruzioni di salto

9.5.1 Istruzioni di salto incondizionato

Le istruzioni di salto incondizionato sono riassunte nella tabella

Cod. mnemonico	Operazione
JMP addr	Salta all'indirizzo addr
JMP @A + DPTR	Salta all'indirizzo che si ottiene sommando A con DPTR
CALL addr	Salta al sottoprogramma che inizia all'indirizzo addr. L'indirizzo corrente di PC viene salvato nello stack.
RET	Ritorna da sottoprogramma
RETI	Ritorna da interruzione
NOP	Nessuna operazione

Tab. 9.7 – Instruction set: istruzioni salto incondizionato

L'istruzione **JMP** esiste in tre formati diversi: **SJMP, LJMP e AJMP**.

JMP è un codice mnemonico generico che è automaticamente tradotto dall'assemblatore nella variante più adeguata a seconda della posizione e distanza dell'indirizzo a cui saltare.

L’istruzione **SJMP** (Short JuMP) codifica l’indirizzo di salto con un offset relativo al PC; l’istruzione è lunga 2 byte di cui uno è utilizzato per l’offset. L’offset è dunque un numero con segno che va da -128 a + 127.

L’istruzione **LJMP** (Long JuMP) è lunga 3 byte di cui due sono utilizzati per l’offset. L’offset è dunque un numero con segno che va da -32.768 a = +32.767 che permette di saltare in qualunque posizione della memoria.

L’istruzione **AJMP** (Absolute JuMP) utilizza 11 bit per l’indirizzo. L’istruzione è lunga 2 byte; 3 bit del codice operativo e il rimanente byte vengono utilizzati per costruire l’indirizzo. L’indirizzo viene costruito sfruttando i 5 bit più significativi del PC e gli 11 bit sopra menzionati. È perciò possibile effettuare un salto entro 2 k oltre l’istruzione AJMP (è quindi possibile solo un salto "in avanti").

Anche l’istruzione CALL dispone di due varianti **ACALL** e **LCALL** che dal punto di vista dell’indirizzo di salto si comportano analogamente alle istruzioni AJMP e LJMP.

L’istruzione JMP @A + DPTR permette di effettuare un salto con selezione a seconda del valore contenuto nell’accumulatore. L’indirizzo di salto viene calcolato sommando il contenuto del registro `DPTR` a quello dell’accumulatore.

Generalmente `DPTR` viene impostato con l’indirizzo di inizio di una tabella di salti e l’accumulatore viene utilizzato come indice di questa tabella.

Esempio

Si supponga di avere una tabella di salti (tabella_salti) a 5 elementi; il codice per utilizzare questa tabella potrebbe essere il seguente:

```
MOV DPTR, #tab_salti
MOV A, #numero salto
RL A
JMP @A+DPTR
```

La parte “tab_salti” può essere la seguente:

```
tab_salti:

AJMP ind_1
AJMP ind_2
AJMP ind_3
AJMP ind_4
AJMP ind_5
```

L’istruzione `RL A` serve per convertire l’indice che va da 0 a 4 in un numero pari che va da 0 a 8; infatti ogni elemento della tabella di salti è lungo 2 byte.

9.5.2 Istruzioni di salto condizionato

Le istruzioni di salto condizionato sono riassunte nella tabella

Cod. mnemonico	Operazione	Dir	Ind	Reg	Imm	Imp
JZ rel	Salta se A = 0					X
JNZ rel	Salta se A <> 0					X
DJNZ <byte>, rel	Decrementa <byte> e salta se <byte> <> 0	X		X		
CJNE A, <byte>, rel	Salta se A <> <byte>	X			X	
CJNE <byte>, #n, rel	Salta se A <> n		X	X		

Tab. 9.9 – Instruction set: istruzioni salto condizionato

Benché non esista un flag di zero nel registro di stato PSW, le istruzioni JZ e JNZ funzionano controllando direttamente il contenuto dell'accumulatore.

9.6 Tabella riassuntiva

La seguente tabella riporta -in ordine alfabetico- l'instruction set completo dell'8051.

Cod. mnemonico	Operazione
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 è nullo
INC	Incrementa il registro
JB	Salta se il bit è a uno
JBC	Salta se il bit è a uno e resettalo
JC	Salta se il Carry è a uno
JMP	Salta all'indirizzo
JNB	Salta se il bit non è a uno
JNC	Salta se il Carry non è a uno
JNZ	Salta se l'Accumulatore non è nullo
JZ	Salta se l'Accumulatore è 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
MOVX	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

Tab. 9.10 – Instruction set completo

10.
PROGRAMMAZIONE ASSEMBLER 8051

10.1 Semplici programmi

Ora analizzeremo dei sorgenti assembler di semplici programmi per 8051; nel sottocapitolo 10.2 riepilogheremo listando frammenti di codice utili alla programmazione.

In allegato sono presenti i listati interi dei programmi qui commentati, con relativi output su seriale di esempio.

10.1.1 blink.asm

Argomenti: timer, porta a bit

Questo breve programma spegne e accende il led collegato alla primo pin della porta P1; a dire il vero nelle diverse varianti e rielaborazioni del package, i led possono essere vari ed essere collegati su porte e pin differenti (esistono anche versioni che dispongono di un cicalino). I cicli di accensione e spegnimento hanno la stessa durata e sono di un secondo; la temporizzazione è regolata dal timer 0.

Iniziamo con una direttiva dell’A51 definendo dove inizia il nostro *code segment*; se non è presente l’assemblatore parte automaticamente da 0.

```
        CSEG AT 0H
```

Segue il ciclo principale del programma: è un loop infinito. Per prima cosa si aspetta un secondo con la subroutine `waitonesec`, quindi si spegne il led con la

subroutine `blinkclr`, e si esegue di nuovo la successione con `blinkset` che accende il led.
Le subroutine sono chiamate con `CALL`, ovvero delle `ACALL`, e dovranno terminare con `RET`.

```
main:
CALL waitonesec
CALL blinkclr

CALL waitonesec
CALL blinkset

JMP main
```

Il `JMP` salta incondizionatamente all'indirizzo `main`, realizzando così il ciclo infinito.

Segue la definizione delle due funzioni di accensione e spegnimento del led.
Si precisa che per accendere il led si setta il bit a 0, invece per spegnerlo lo si setta a 1.

```
blinkset:
MOV P1, #0H
RET

blinkclr:
MOV P1, #1H
RET
```

È importante notare come le porte sono utilizzate: tutta la memoria è indirizzabile a bit, e le porte sono mappate sulla memoria. Quindi P1 è come se fosse un indirizzo di memoria qualsiasi su cui si copia un valore.
Il bit della porta che ci interessa è il primo, di conseguenza si possono sostituire i due `MOV` con una coppia di `CLR` e `SETB` che agiscono sul singolo bit `P1.0`.

```
blinkset:
CLR P1.0
RET

blinkclr:
SETB P1.0
RET
```

Inoltre già come detto prima le porte corrispondo ad indirizzi in RAM, e il loro indirizzo parte da `80H` che corrisponde a `P0.0`; `P1.0` corrisponde quindi a 88H.

```
blinkset:
CLR 88H
RET

blinkclr:
```

```
        SETB 88H
        RET
```

Passiamo ora alla subroutine più importante di questo programma: `waitonesec`. Per prima cosa è necessario calcolare i valori da impostare al timer per una corretta attesa di 1 secondo. Si parte dal clock del 8051, che nel nostro caso è 12 MHz; esso impiega 12 colpi di clock per eseguire un ciclo di macchina (e quindi un incremento del timer). Quindi il timer verrà incrementato

12 000 000 / 12 = 1 000 000

di volte al secondo. Siccome tale valore è troppo elevato per essere contato dalla modalità a 16 bit del timer (16 bit sono massimo 65 536 conteggi), spezziamo il ciclo in 100 conteggi da 10 000 incrementi. `R5` sarà il nostro contatore.

```
        waitonesec:
        MOV R5, #100 ; ripeto 100 volte

        restart:
```

Il timer 0 a 16 bit è composto da due registri a 8 bit, `TH0` e `TL0` (high e low); ogni volta che `TL0` raggiunge 256 (ovvero FF), `TH0` si incrementa. Per contare 10 000 incrementi, il timer deve partire quindi da un valore conveniente, che è dato da

65 535 – 10 000 = 55 535

Tale valore in esadecimale è D8EF, e quindi carichiamo il byte più significativo con D8 (216) e il meno significativo con EF (239). Potevamo anche dividere 55 535 per 256 e prendere il modulo ed il resto con medesimi risultati.

```
        MOV TH0, #216
        MOV TL0, #239
```

A questo punto carichiamo `TMOD` con la modalità a 16 bit del timer 0, settando il primo bit detto T0M0 e lasciamo a zero il successivo bit T0M1.
Settiamo `TR0` che fa partire il timer 0. L'opcode che bisogna eseguire dopo è l'attesa dell'overflow, che indica il raggiungimento dei 10 000 cicli di clock: esso è ottenuto con un jump condizionale JNB che aspetta il bit a 1 di TF0 saltando continuamente al proprio indirizzo.

```
        MOV TMOD, #01
        SETB TR0 ; go!

        JNB TF0, $ ; aspetto l'overflow
```

Sono passati quindi

10 000 * 12 / 12 000 000 = 0.01

secondi.

Dobbiamo ripetere quest'operazione 100 volte per avere 1 secondo, senza dimenticare di resettare ogni volta il bit `TR0`.

```
        CLR TR0
        DJNZ R5, restart

        RET
```

La subroutine termina con `RET` che ritorna all'indirizzo successivo alla CALL chiamante.

Non bisogna dimenticare l'`END` finale che avverte l'A51 della fine del programma.

```
        END
```

10.1.2 hello.asm

Argomenti: seriale, indirizzamento indiretto a codice memoria interna

Hello.asm è il classico programma che stampa "`Hello, World`"; nell'8051 l'unico dispositivo per l'output di stringhe di testo è la seriale, e quindi per comunicare con l'esterno bisogna capire la messa a punto della porta seriale.
Inoltre hello.asm presenta la gestione delle stringhe sull'8051, che è molto semplice.

Qui decidiamo di spostare il blocco del programma dopo l'indirizzo 30H nella memoria interna; inoltre il nostro `main` (dal punto di visto logico chiaramente, per l'A51 è una serie di istruzioni come le altre) è situato verso la fine del codice.

```
        CSEG AT 0H
        JMP start
        CSEG AT 30H ; sposto il codice a 30H
```

Si noti come tramite la direttiva `CSEG` è possibile definire l'indirizzo dove risiederà l'istruzione successiva: per esempio qui si passa all'indirizzo 30H dove si definisco importanti funzioni di I/O che utilizzeremo spesso.

```
        putc:
        JNB TI, putc
        CLR TI
        MOV SBUF, A
        RET
```

`putc` stampa un carattere, identificato dal suo codice ASCII, presente in `A`:
- aspetta che `SBUF` (il serial buffer) sia libero, ovvero quando il bit `TI` passa da 0 a 1
- si assicura di resettare `TI` per occupare il buffer
- copia il valore in `A` dentro `SBUF`.

```
    puts:
    CLR A
    MOVC A, @A+DPTR
```

`puts` utilizza `putc` per stampare su seriale una stringa, cioè un vettore di codici ASCII terminati da uno 0 (nota: valore 0, non il codice ASCII di 0). Si inizia con l'azzerare `A` che potrebbe contenere valori non voluti.
L'istruzione successiva è un esempio di indirizzamento a 16 bit della memoria di programma (detto indiretto a codice); `DPTR` contiene l'indirizzo di inizio della stringa ed è, per così dire, il dato in input della subroutine. `MOVC` copia in `A` ciò che è presente all'indirizzo dato dalla somma tra `A` e `DPTR`.
Il risultato è che all'inizio `A` contiene il primo carattere. Segue la subroutine `more` che chiama la `putc` e incrementa `DPTR`, passando quindi al byte successivo.

```
    JNZ more
    RET

    more:
    CALL putc
    INC DPTR
    JMP puts
```

Il `JNZ more` ci assicura che la `puts` termina se `A` (l'accumulatore) è uguale a 0, il carattere terminale di una stringa.

Passiamo al corpo del codice: il salto incondizionato all'indirizzo 0x00 salta qui.
Esso è composto da due parti: il setup della seriale e la chiamata a `puts`.
Con l'istruzione `ORL TMOD, #20H` imposto con una maschera la modalità del timer 1, che utilizzeremo per regolare il baud rate della porta seriale.
20H in binario corrisponde a 100 000, ovvero setto il bit 5 di `TMOD`, detto `T1M1`, senza interferire con il timer 0. `T1M0` viene lasciato a 0, selezionando così la modalità a 8bit con *auto reload*: in `TH1` copiamo il valore che deve essere caricato in `TL1` quando raggiunge 255 e viene ulteriormente incrementato. Quindi `TL1` conta continuamente dal valore contenuto in `TH1` fino a FF. Si fa partire il timer settando `TR1`.
Nel registro serial control `SCON` copiamo il numero binario 1010000 che setta `REN` (*receive enabled*) e `SM1`, impostando la modalità UART a 8bit e abilitando la ricezione.
`PCON` serve per il power control del 8051, cosa che non ci interessa, ma l'ultimo bit `PCON.7` (detto `SMOD`) se settato, serve per raddoppiare il baud rate della seriale; è proprio ciò che facciamo.

per ottenere 2 400 baud impostiamo `TH1` con 230, valore ottenuto con la formula

`TH1` = 256 - ((frequenza clock / 192) / baud rate)

con in nostri valori

`TH1` = 256 - ((12 000 000 / 192) / 2400) = 229,95833

```
start:
ORL TMOD, #20H
SETB TR1
MOV SCON, #50H
MOV TH1, #230 ; 2400 baud
ORL PCON, #80H

SETB TI
```

Per ultimo settiamo `T1` per segnale che SBUF è libero.

Ora che la seriale è configurata possiamo copiare in `DPTR` l'indirizzo di dove si trova la stringa voluta e chiamare `puts`. Questa operazione è eseguita infinite volte.

```
stampo:
MOV DPTR, #HELLO
CALL puts

JMP stampo
```

Definiamo per convenienza l'"a capo" con un paio di direttive `EQU`, e all'indirizzo `HELLO` definiamo un vettore di byte terminato da uno 0.

```
CR EQU 0DH
LF EQU 0AH

HELLO:
DB 'Hello World', CR, LF, 0

END
```

10.1.3 name.asm

Argomenti: seriale, indirizzamento diretto memoria esterna

Ora vediamo come fare per memorizzare un dato generico ottenuto dall'input nella memoria RAM esterna e riprodurlo in output; questo esempio è un'estensione del precedente *hello.asm*.

Come prima riproponiamo la `puts`.

```
CSEG AT 0H
JMP start

putc:
JNB TI, putc
CLR TI
```

```
MOV SBUF, A
RET

puts:
CLR A
MOVC A, @A+DPTR
JNZ more
RET

more:
CALL putc
INC DPTR
JMP puts
```

Definiamo un'ulteriore subroutine putrs che stampa una stringa in memoria esterna. Questo poiché per prelevare un byte dalla RAM esterna è necessario utilizzare un opcode diverso, l'istruzione MOVX: questa copia il valore contenuto nell'indirizzo presente in DPTR. Gli indirizzi partono da 0x00 che corrisponde al primo byte di memoria RAM esterna.
In DPTR dovremmo quindi in questo caso caricare l'indirizzo del vettore in memoria esterna.

```
putrs:
MOVX A, @DPTR
JNZ morer
RET

morer:
CALL putc
INC DPTR
JMP putrs
```

Introduciamo anche un'utilissima subroutine che accetta una stringa da seriale e la salva in memoria esterna ad un indirizzo prestabilito: gets.
Inversamente al caso della scrittura su seriale, per prima cosa si aspetta che SBUF sia occupato a causa di un byte in arrivo, controllando il passaggio del bit RI da 0 a 1. Quando questo avviene preleviamo con un MOV il codice ASCII del carattere immesso e lo poniamo in A.
Quindi resettiamo RI visto che non vi è più necessità del dato in attesa su SBUF.

```
gets:
JNB RI, gets
MOV A, SBUF ; legge il carattere
CLR RI
CALL putc
```

Notiamo un dettaglio visivo molto comodo: a questo punto chiamiamo putc che ci presenterà sul terminale seriale l'*echo* del carattere appena immesso.

Successivamente l'istruzione `CJNE` ("Compare and Jump if Not Equal") controlla se `A` è un carriage return `CR`, primo byte di un a capo `CR LF`; se così fosse passa al successivo `JMP` ed esce ritornando all'indirizzo chiamante.

```
CJNE A, #CR, NoCr ; controlla se CR
SJMP finegets
```

Se invece `A` non è un `CR`, copiamo il valore di `A` nella memoria esterna puntata da `DPTR`, e incrementiamo quest'ultimo affinché la prossima volta scriviamo nella cella successiva.

```
NoCr:
MOVX @DPTR, A
INC DPTR
DJNZ R1, gets
```

Un `DJNZ R1, gets` ci permette di ripetere il ciclo di memorizzazione il numero di volte presente in `R1`: esso contiene quindi il valore della lunghezza massima della stringa (essa può comunque essere interrotta prima da un invio).

Segue la fine della subroutine, che delimita la stringa con uno 0.

```
finegets:

MOV A, #0
MOVX @DPTR, A
RET
```

Ora passiamo all'indirizzo dove salta l'inizio del codice. Anche qui come prima cosa settiamo le modalità della porta seriale e la velocità a 2 400 baud.

```
start:
ORL TMOD, #20H
SETB TR1
MOV SCON, #50H
MOV TH1, #230
ORL PCON, #80H

SETB TI
```

Il primo I/O che eseguiamo è la scrittura su seriale della stringa presente all'indirizzo `#question`, mettendo in `DPTR` tale indirizzo e chiamando poi la `puts`.

```
MOV DPTR, #question
CALL puts

MOV DPTR, #inbuff
MOV R1, #inlen
CALL gets
```

Dopodichè configuriamo la `gets` impostando l'indirizzo dove scrivere in memoria esterna (presente in `#inbuff`) ed il numero di celle che possono comporre la stringa (`#inlen`). Segue la `CALL` a `gets` per prelevare la stringa.

Ora il più è fatto, si tratta di riscrivere la stringa memorizzata dalla `gets` con una `putrs`, circondata da altre due stringhe memorizzate in memoria interna, "*Hi* " e il " *!*".

```
    MOV DPTR, #answer1
    CALL puts

    MOV DPTR, #inbuff
    CALL putrs

    MOV DPTR, #answer2
    CALL puts

    JMP $
```

Per finire l'esecuzione del codice sull'8051 si utilizza la tecnica dell'*end dinamico*, ovvero si continua a saltare con un `JMP` incondizionato al proprio indirizzo (`$`).

Dobbiamo ancora definire dove scrivere in memoria esterna, impostando con due direttive dell'A51 `#inbuff` e `#inlen`; non avendo particolari esigenze memorizziamo il nome nei primi 11 byte.

```
    inbuff EQU 0
    inlen EQU 10

    CR EQU 0DH
    LF EQU 0AH
```

E finiamo con la definizione delle stringhe fisse nel codice utilizzate nel programma.

```
    question:
    DB 'What is your name? ', 0
    answer1:
    DB CR, LF, 'Hi ', 0
    answer2:
    DB ' !', CR, LF, 0

    END
```

10.1.4 tabellina.asm

Argomenti: operazioni con i numeri

Per passare da codice ASCII a numero decimale è sufficiente sottrarre al carattere il valore ASCII di 0; nel caso di un numero composto da più cifre si deve moltiplicare la cifra per il suo ordine e sommare il tutto. *tabellina.asm* prende in input un numero da una cifra e ne presenta i multipli fino al ventottesimo, per utilizzare un solo registro da 8 bit (28 * 9 = 252).

Impostiamo la seriale a 2 400 baud e poniamo in output la richiesta del numero da moltiplicare.

```
CSEG AT 0H

ORL TMOD, #20H
SETB TR1
MOV SCON, #50H
MOV TH1, #230 ; 2400 baud
ORL PCON, #80H

SETB TI

inizio:
MOV DPTR, #input
CALL puts
```

Chiamiamo la subroutine `getnumber` che vedremo successivamente: essa mette in `R3` il numero immesso dall'utente (il numero vero e proprio, non l'ASCII).
Decidiamo di tenere in `R1` il contatore da utilizzare nella moltiplicazione, ed in `R7` il contatore dei cicli che stiamo facendo; quest'ultimo viene decrementato e parte da 29 per il motivo detto prima: se l'utente vuole i multipli di 9, il risultato dopo 252 supera il byte e quindi si potrebbe tenere il numero in due registri.

```
CALL getnumber

MOV R1, #0
MOV R7, #29
```

Il ciclo principale inizia con lo stampare l'operazione da eseguire:
un invio, il numero in `R1`, una stringa " ` * ` ", l'operando, ed un " ` = ` ". L' effetto è questo:

```
001 * 4 =
```

```
main:
MOV DPTR, #acapo
CALL puts

MOV A, R1
CALL putnumber
```

```
MOV DPTR, #per
CALL puts

MOV A, R3
CALL putonenumber

MOV DPTR, #uguale
CALL puts
```

Notiamo che per stampare una cifra utilizziamo `putonenumber`, mentre per un numero a massimo 3 cifre `putnumber`: vedremo il codice dopo.

Ora eseguiamo l'operazione di moltiplicazione tra l'operando ed il contenuto del contatore `R1`; `MUL` utilizza quest'unica sintassi possibile. `putnumber` stamperà poi il valore di `A`.

```
MOV A, R3
MOV B, R1
MUL AB

CALL putnumber

INC R1
DJNZ R7, main
```

Incrementiamo il contatore `R1` e decrementiamo `R7`: se non siamo a 0 riparte il ciclo principale.
Se siamo giunti a 0 la routine principale riparte, chiedendo un nuovo numero da moltiplicare.

```
MOV DPTR, #acapo
CALL puts

JMP inizio
```

Ora passiamo alla solita subroutine per l'output di stringhe.

```
putc:
JNB TI, putc
CLR TI
MOV SBUF, A
RET

puts:
CLR A
MOVC A, @A+DPTR
JNZ more
RET

more:
```

```
        CALL putc
        INC DPTR
        JMP puts
```

Vediamo la subroutine `putnumber`. Essa stampa numeri decimali fino a un massimo di tre cifre; inoltre stampa il contenuto di `A`, quindi massimo 255. Inizia come al solito ad aspettare che il buffer sia libero controllando `TI`.
In A c'è il numero che si vuole mandare sulla seriale: tale numero viene diviso due volte, la prima volta per 100 e la seconda per 10, per ottenere il valore delle centinaia e delle decine, per rappresentarlo in notazione decimale.

```
        putnumber:
        JNB TI, putnumber
        CLR TI

        MOV B, #100
        DIV AB ; A quoziente, B resto
```

Iniziamo con le centinaia dividendo `A`, il numero, per 100, che viene quindi copiato in `B`. Il modulo sarà la parte intera dell'operazione, il resto servirà invece per le decine e le unità: per esempio se abbiamo il numero 123 in `A`, dopo la `DIV A` contiene 1 e `B` contiene 23.

A questo punto vogliamo liberarci di `A` contenente le centinaia e lo stampiamo subito. Per stampare il valore ASCII del numero, dobbiamo prima sommare il valore ASCII di 0, 48. Nel nostro esempio `A` (che conteneva 1), dopo la `ADD` contiene 49 (in decimale).

```
        ADD A, #'0'
        MOV SBUF, A
```

Prepariamo la prossima divisione delle decine muovendo il resto in `A`, ed aspettiamo che la cifra delle centinaia sia rimossa da `SBUF`; inoltre in `B` copiamo 10. Dopo la `DIV AB`, `A` contiene la decina e `B` l'unità; nell'esempio `A` conterrà 2 e `B` 3.

```
        MOV A, B
        JNB TI, $
        CLR TI
        MOV B, #10

        DIV AB
```

Procediamo quindi come prima a mandare su seriale il valore ASCII, prima della decina, e poi dell'unità (che va copiata in A per l'elaborazione).

```
        ADD A, #'0'
        MOV SBUF, A

        JNB TI, $
        CLR TI
```

```
MOV A, B
ADD A, #'0'
MOV SBUF, A

RET
```

Una subroutine decisamente meno complicata è quella che stampa una sola cifra.

```
putonenumber:
JNB TI, putonenumber
CLR TI

ADD A, #'0'
MOV SBUF, A

RET
```

La routine di input è semplice ed accetta un numero ASCII che converte in numero vero e proprio e lo depone in `R3`.

```
getnumber:
JNB RI, getnumber
MOV A, SBUF
CLR RI
CALL putc ; l'echo

SUBB A, #'0'
MOV R3, A

RET
```

Il codice termina con la memorizzazione in memoria interna delle stringhe utilizzate.

```
CR EQU 0DH
LF EQU 0AH

input: DB 'number: ', 0
per: DB ' * ', 0
uguale: DB ' = ', 0
acapo: DB CR, LF, 0

END
```

10.1.5 bubblesort.asm

Argomenti: vettore, indirizzamenti, algoritmo

Analizziamo ora un programma di ordinamento che si basa sull'algoritmo *bubble sort*. Avendo un vettore di *n* elementi da ordinare, si eseguono *n* cicli, ed a ogni ciclo si confrontano a due a due i gli elementi: in questo caso decidiamo di ordinare gli elementi (numeri) in modo crescente, quindi si scambiano se non rispettano questo ordine. Il *bubble sort* ha un ordine di grandezza molto elevato, in ogni caso circa $O(n^2)$.
Per la gestione del vettore l'utilizzo del registro a 16 bit `DPTR` è più elaborata; inoltre si presenta un utilizzo dello stack.

La prima parte del codice è il main, che termina con l'*end dinamico*.
Dopo aver settato la porta seriale, chiediamo all'utente il numero di elementi che vorrà immettere e lo leggiamo; successivamente chiediamo gli elementi uno ad uno con la subroutine `chiediamoli`. Quindi invochiamo `bubblesort` per ordinare gli elementi ormai memorizzati nella RAM esterna, e li stampiamo.

```
CSEG AT 0H

ORL TMOD, #20H
SETB TR1
MOV SCON, #50H
MOV TH1, #230 ; 2400 baud
ORL PCON, #80H

SETB TI

MOV DPTR, #input
CALL puts
CALL getnum ; numero di elementi
MOV A, R1
CALL chiediamoli

CALL bubblesort
PUSH ACC
MOV DPTR, #incorder
CALL puts
POP ACC

CALL vediamoli

JMP $
```

Per comprensione riportiamo le routine partendo da quelle più semplici e già descritte nel sorgenti precedenti.

```
putc:
JNB TI, putc
CLR TI
MOV SBUF, A
```

```
RET

puts:
CLR A
MOVC A, @A+DPTR
JNZ more
RET

more:
CALL putc
INC DPTR
JMP puts
```

`putbignum` stampa in ASCII un numero decimale su di due cifre, ovvero al massimo il numero 99, con la tecnica della divisione vista prima.

```
putbignum:
JNB TI, putbignum
CLR TI

MOV B, #10
DIV AB

ADD A, #'0'
MOV SBUF, A

wait:
JNB TI, wait
CLR TI

MOV A, B
ADD A, #'0'
MOV SBUF, A

RET
```

`getnum` lavora in modo analogo a quanto visto prima, ma deve preoccuparsi come in *name.asm* di riconoscere il `CR`, segno che il numero è terminato: essendo però di due cifre la distinzione fra decine ed unità è semplice. Se dopo la prima cifra abbiamo un invio, il nostro numero è una unità. Se abbiamo una seconda cifra, la prima deve essere moltiplicata per 10 e sommata a questa.
Iniziamo con il leggere la prima cifra e scrivere l'echo all'utente.

```
getnum:
JNB RI, $
MOV A, SBUF
CLR RI
CALL putc
```

Quindi lo memorizziamo come numero in `R1`.

```
        SUBB A, #'0'
        MOV R1, A
```

Passiamo al secondo input dell'utente.

```
        JNB RI, $
        MOV A, SBUF
        CLR RI
        CALL PUTC
```

Controlliamo se il secondo carattere (che è in `A`) è un `CR`; se così fosse saltiamo alla fine con la successiva `SJMP fine`, senza dimenticare che in `R1` c'è la nostra unità e che sarà il dato in uscita dalla routine. Altrimenti saltiamo alla subroutine che formerà il numero a partire dalle due cifre.

```
        CJNE A, #CR, NoCr
        SJMP fine
```

Questa inizia con il convertire il nostro secondo carattere che abbiamo controllato da ASCII in numero. Quindi salviamo la cifra in `R2` e riprendiamo la decina che era in `R1`: si deve assegnare il giusto ordine alla decina, con una moltiplicazione per 10.

```
        NoCr:
        SUBB A, #'0'
        MOV R2, A
        MOV A, R1
        MOV B, #10
        MUL AB
```

Il nostro ultimo passo è quello di sommare il risultato di tale moltiplicazione (che è in `A`) alla unità presente in `R2`. Il risultato infine va posto in `R1`.

```
        ADD A, R2 ; sommo decine ed unita', in A risultato
        MOV R1, A

        fine:
        RET
```

Vediamo ora la routine di memorizzazione degli elementi `chiediamoli`. Essa vuole in `A` il numero di elementi da chiedere, che viene salvato nello stack per le successive routine. Inoltre `R7` sarà il nostro contatore.

```
        chiediamoli:
        MOV R7, A
        PUSH ACC

        MOV DPTR, #vectbuff
```

Ripristiniamo in `DTPR` l'indirizzo a 16 bit del vettore di numeri che andiamo a scrivere in memoria esterna.

La routine entra quindi nel ciclo dove chiede ad uno ad uno gli elementi con la `getnum`, e li memorizza.
Per ragioni di chiarezza si presenta inoltre l'indice del vettore. L'output all'utente sarà questo:

element 01 :

Per poter scrivere una stringa e memorizzare in RAM è necessario salvare l'indice dell'indirizzo del vettore `DPTR`, per esempio nello stack. Per fare questa operazione di `PUSH`, si deve spezzare `DPTR` nei suoi due registri costitutivi da 8 bit: `DPL` e `DPTR`.
`R0` servirà poi per scrivere l'indice.

```
unaltro:
MOV R0, DPL
PUSH DPL
PUSH DPH
```

Segue quindi la scrittura di "*element* " e di " *:* ", con l'indice interposto. Quest'ultimo deve essere regolato in modo che non stampi l'indirizzo alla memoria, e che parta da 1.
Per la prima cosa è quindi necessario sottrarre l'indirizzo di partenza del vettore; con questo meccanismo non possiamo quindi scrivere i dati in RAM dalla `255 – 99 = 156°` cella a causa degli 8 bit di limite di `R0`.
Per il secondo dettaglio basta incrementare `R0` di uno.

```
MOV DPTR, #elemento
CALL puts

MOV A, R0
SUBB A, #vectbuff
INC A
CALL putbignum

MOV DPTR, #duepunti
CALL puts
```

A questo punto possiamo finalmente chiedere l'elemento con `getnum`.

```
CALL getnum
MOV A, R1
```

Ora abbiamo l'elemento in `A` e lo salviamo con l'indirizzamento esterno diretto; non dimentichiamo però di ripristinare `DPTR` salvato nello stack con due `POP` corrispondenti alle precedenti `PUSH`.

```
POP DPH
POP DPL

MOVX @DPTR, A
```

```
    INC DPTR
    DJNZ R7, un altro

    POP ACC
    RET
```

Il puntatore a 16 bit DPTR deve essere incrementato di uno prima di passare ad un altro elemento.
E come ultima istruzione ripristiniamo l'accumulatore che contiene il numero di elementi.

Prima di passare alla subroutine di ordinamento, descriviamo brevemente quella che stampa sulla porta seriale gli elementi del vettore.

```
    vediamoli:
    MOV R7, A
    MOV DPTR, #vectbuff

    vedo:
    MOV R0, DPL

    PUSH DPL
    PUSH DPH

    MOV DPTR, #elemento
    CALL puts
    MOV A, R0
    SUBB A, #vectbuff
    INC A
    CALL putbignum
    MOV DPTR, #duepunti
    CALL puts
```

È molto simile a `chiediamoli`. Qui però prima si legge il valore in memoria esterna e poi lo si stampa.

```
    POP DPH
    POP DPL

    MOVX A, @DPTR
    CALL putbignum

    INC DPTR

    DJNZ R7, vedo
    RET
```

Vediamo la routine `bubblesort`, Per prima cosa ci teniamo due copie di `A`, `R5` e `R6`, che serviranno per i contatori dei due cicli principali.

```
bubblesort:
MOV R5, A
MOV R6, A
PUSH ACC
```

Con `reinizio` parte uno dei due cicli principali: `R6` è il suo contatore (lo decrementiamo alla fine), e `R5` serve per ricaricare ad ogni ciclo `R7` con il numero degli elementi.

```
reinizio:
MOV B, R5
MOV R7, B
```

Ripristiniamo l'indirizzo del vettore da ordinare in memoria, e come prima cosa del secondo ciclo (chiamato `ciclo`) preleviamo l'elemento posto nella casella corrente. Quindi lo copiamo in `R2` e copiamo in `R3` l'elemento successivo incrementando `DPTR` di uno.

```
MOV DPTR, #vectbuff
ciclo:
MOVX A, @DPTR
MOV R2, A
INC DPTR
MOVX A, @DPTR
MOV R3, A
```

Per ogni ciclo abbiamo in `R2` e in `R3` (ma anche in `A`) le coppie di elementi successivi del vettore.

Ora dobbiamo confrontare questa coppia di elementi, per stabilire se il primo è minore oppure maggiore del secondo. Per farlo sottraiamo al secondo valore ancora presente in `A` il primo (in `R2`). Se il contenuto di `A` è maggiore del contenuto di `R2`, il risultato è negativo e di conseguenza viene generato un carry: per verificare la sua presenza, controllo se l'ultimo bit della Process Status Word (`PSW.7`) è settato. Se così è, viene resettato e saltiamo all'indirizzo `eminore` grazie all'opcode `JBC`. Se invece così non è, rimettiamo in A e B i due elementi senza invertirli e saltiamo il caso in cui siano da invertire.

```
SUBB A, R2
JBC PSW.7, eminore
MOV A, R3
MOV B, R2
SJMP emaggiore
eminore:
MOV A, R2
MOV B, R3
emaggiore:
```

Se sono da scambiare, li copiamo invertiti in `A` e `B`.

Il passo successivo è riscrivere la coppia ordinata nella memoria. Copiamo prima il secondo elemento letto, e dopo aver decrementato il puntatore, il primo; `DEC` non può decrementare il registro a 16 bit, ma decrementiamo solo gli 8 bit meno significativi.

```
    MOVX @DPTR, A

    DEC DPL ; agisco su meno di 255

    MOV A, B
    MOVX @DPTR, A
```

Passiamo alla coppia successiva incrementando `DPTR` e saltando a `ciclo`; questo viene fatto per ogni coppia del vettore (se il vettore è lungo *n* elementi, il ciclo va fatto *n - 1* volte). Inoltre questo ciclo di *n - 1* volte, va rifatto *n* volte per assicurare al primo elemento di poter raggiungere l'ultima cella se necessario.

```
    INC DPTR

    DEC R7
    CJNE R7, #1, ciclo
    DJNZ R6, reinizio

    POP ACC
    RET
```

Il codice si conclude con i soliti dati in memoria di programma.

```
    CR EQU 0DH
    LF EQU 0AH

    input: DB 'number of element (max 99, each max 99): ',
      0
    elemento: DB CR, LF, 'element ', 0
    duepunti: DB ' : ', 0
    acapo: DB CR, LF, 0
    incorder: DB CR, LF, 'increasing order: ', 0

    vectbuff EQU 0

    END
```

10.1.6 twosort.asm

Argomenti: vettori, algoritmi

Abbiamo visto un semplice algoritmo di ordinamento. Ora vediamone uno più complesso, per esempio il *count sort*. Esso ha una complessità inferiore al *bubble* (circa *O(3n)* con i nostri valori), ma occupa una maggiore quantità di memoria: si contano le presenze degli elementi nel vettore da ordinare, riportandole su un vettore di supporto lungo quanto il massimo valore da ordinare. Quindi questo secondo vettore viene elaborato riportando sul primo gli elementi. Non è un algoritmo rigoroso, in quanto si perde l'informazione nel caso di due elementi con la stesa chiave: essi non saranno più distinguibili.

Il codice di *twosort.asm* è quello di *bubblesort.asm* con l'aggiunta del *counting sort*: l'utente può scegliere l'algoritmo voluto.

```
        CSEG AT 0H

        ORL TMOD, #20H
        SETB TR1
        MOV SCON, #50H
        MOV TH1, #230 ; 2400 baud
        ORL PCON, #80H

        SETB TI

        MOV DPTR, #input
        CALL puts
        CALL getnum
        MOV A, R1
        PUSH ACC

        CALL chiediamoli
```

A questo punto del *main*, stampiamo la scelta sull'algoritmo desiderato, e quindi ordinamo con la subroutine corrispondente: in verità la scelta è basata sulla lettera '`b`', se viene immesso un carattere qualsiasi parte il `countsort`.

```
        MOV DPTR, #algo
        CALL puts
        MOV DPTR, #bubble
        CALL puts
        MOV DPTR, #counting
        CALL puts
        CALL getc
        POP ACC

        CJNE R1, #'b', algo_count
        CALL bubblesort
        JMP algo_end

        algo_count:
```

```
CALL countsort
algo_end:

CALL vediamoli

JMP $
```

Vediamo ora come è implementato il *counting sort*.
Si inizia con il tenere una copia del numero di elementi in `R6`.

```
countsort:
PUSH ACC ; salvo il numero di elementi
MOV R6, A
```

Esso è organizzato in tre parti: per prima cosa azzeriamo il vettore di supporto `presenze`. Quindi contiamo gli elementi presenti nel vettore da ordinare, e per terzo passiamo in rassegna `presenze` formando il vettore ordinato.

Il codice per azzerare il vettore di presenze è molto semplice: passiamo in rassegna la parte di memoria tra l'indirizzo `presenze` e `finepresenze`, ed in ogni cella copiamo `A` che contiene 0.

```
MOV A, #0
MOV DPTR, #presenze
cleanp:
MOVX @DPTR, A
INC DPTR
MOV R0, DPL
CJNE R0, #finepresenze, cleanp
```

La seconda parte, come la terza, deve gestire due puntatori in locazioni di memoria differenti: il vettore da ordinare e quello del *counting*. Questo viene fatto utilizzando lo stack per memorizzare il primo, e un registro per tenere il byte basso dell'altro (il byte alto non cambia essendoci meno di 256 elementi).
`costruisco` è la seconda subroutine, che parte copiando in `A` l'elemento presente nella cella puntata dal corrente `DPTR`; esso punta agli elementi del vettore da ordinare `vectbuff`, e sarà incrementato ad ogni ciclo. I due `PUSH` lo salvano nello stack.

```
MOV DPTR, #vectbuff
costruisco:
MOVX A, @DPTR
PUSH DPL
PUSH DPH
```

Ora che in `A` abbiamo l'elemento, dobbiamo registrare la sua presenza sul vettore `presenze`. Per farlo, incrementiamo la cella di memoria corrispondente al numero che abbiamo trovato (il vettore essendo stato precedentemente azzerato).

Per generare tale indirizzo, copiamo in `DPTR` l'indirizzo di inizio del vettore di supporto, e sommiamo al byte meno significativo il valore trovato prima nel vettore da ordinare (che è in `A` ed è al massimo 99) con la `ADD A, #presenze`.

```
MOV DPTR, #presenze
ADD A, #presenze
MOV DPL, A
```

Ora basta leggere il numero che c'è in quella cella di memoria esterna e riscriverlo incrementato di uno.

```
MOVX A, @DPTR
INC A
MOVX @DPTR, A
```

Questo secondo ciclo termina con il ripristino dallo stack dell'indirizzo della cella corrente del vettore da ordinare ed il suo incremento. Quando tutti gli elementi del vettore sono stati registrati nel vettore delle presenze, esce.

```
POP DPH
POP DPL
INC DPTR
DJNZ R6, costruisco
```

La terza subroutine è la più complessa delle tre, e prima di entrare nel ciclo inizializza: `R0` che sarà l'indice del numero che stiamo valutando, `DPTR` all'indirizzo d'inizio del vettore `presenze`, e `R2` che conterrà l'indice del vettore che riscriviamo ordinato.

```
MOV R0, #0
MOV DPTR, #presenze
MOV R2, #vectbuff
```

Il ciclo può iniziare, caricando in `A` il contenuto della cella corrente delle presenze. Questo valore rappresenta il numero di elementi con il valore di `R0` che sono stati trovati nel vettore d'origine (non dimentichiamo che `R0` sarà incrementato ad ogni ciclo). Se non ne abbiamo trovati, esso sarà chiaramente uguale al valore di inizializzazione, ovvero 0: in questo caso saltiamo la parte di scrittura sul vettore da ordinare, saltando all'indirizzo `nonconto`.

```
aggiorno:
MOVX A, @DPTR ; numero di elementi di valore R0

CJNE A, #0, conto
SJMP nonconto
```

Se invece `A` non è a zero, ci teniamo tale valore in `R1` e salviamo nello stack l'indirizzo corrente del vettore `presenze`.

```
conto:
MOV R1, A
PUSH DPL
PUSH DPH
```

Entriamo quindi nel `sottociclo` che aggiunge tanti elementi quanti erano quelli trovati in `R1`.
Per farlo ripristiniamo in `DPTR` l'indirizzo della cella del vettore del elementi, tenendo come abbiamo detto l'informazione sulla cella corrente in `R2` (esso viene incrementato ogni volta che scriviamo un elemento su `vectbuff`).
Non possiamo scrivere direttamente `R0` su `@DPTR`, e quindi `A` fa da tramite.

```
sottociclo:
MOV DPTR, #vectbuff
MOV DPL, R2
MOV A, R0
MOVX @DPTR, A

INC R2
DJNZ R1, sottociclo
```

Ora che abbiamo aggiornato questa parte di vettore degli elementi, prima di incontrare l'indirizzo `nonconto` dobbiamo ancora ripristinare il `DPTR` salvato prima nello stack.
Concludiamo il secondo ciclo passando alla cella successiva di `presenze`; controlliamo se abbiamo raggiunto l'ultimo degli elementi permessi, 99.

```
POP DPH
POP DPL
nonconto:

INC DPTR
INC R0
CJNE R0, #99, aggiorno ; 99 presenze
```

Il `count sort` è fatto, non ci resta che ripristinare A e ritornare all'indirizzo chiamante.

```
POP ACC
RET
```

Seguono le routine identiche a quelle di *bubblesort.asm*.

```
bubblesort:
MOV R5, A
MOV R6, A
PUSH ACC ; salvo il numero di elementi
reinizio:
MOV B, R5
MOV R7, B
```

```
MOV DPTR, #vectbuff
ciclo:
MOVX A, @DPTR
MOV R2, A
INC DPTR
MOVX A, @DPTR
MOV R3, A

SUBB A, R2
JBC PSW.7, eminore ; c'e' carry, risultato negativo
MOV A, R3
MOV B, R2
SJMP emaggiore
eminore:
MOV A, R2
MOV B, R3
emaggiore:
MOVX @DPTR, A

DEC DPL ; meno di 255

MOV A, B
MOVX @DPTR, A

INC DPTR ; la coppia successiva

DEC R7
CJNE R7, #1, ciclo
DJNZ R6, reinizio

POP ACC
RET

chiediamoli:
MOV R7, A ; R7 contatore
PUSH ACC

MOV DPTR, #vectbuff ; per il vettore

unaltro:
MOV R0, DPL
PUSH DPL
PUSH DPH

MOV DPTR, #elemento
CALL puts
MOV A, R0
SUBB A, #vectbuff
INC A ; parto da uno
CALL putbignum
MOV DPTR, #duepunti
CALL puts
```

```
CALL getnum
MOV A, R1

POP DPH
POP DPL

MOVX @DPTR, A
INC DPTR

DJNZ R7, unaltro

POP ACC
RET

vediamoli:
MOV R7, A
MOV DPTR, #vectbuff

vedo:

MOV R0, DPL

PUSH DPL
PUSH DPH

MOV DPTR, #elemento
CALL puts
MOV A, R0
SUBB A, #vectbuff
INC A ; parto da uno
CALL putbignum
MOV DPTR, #duepunti
CALL puts

POP DPH
POP DPL

MOVX A, @DPTR
CALL putbignum

INC DPTR

DJNZ R7, vedo
RET

putc:
JNB TI, putc
CLR TI
MOV SBUF, A
RET
```

```
puts:
CLR A
MOVC A, @A+DPTR
JNZ more
RET

more:
CALL putc
INC DPTR
JMP puts

putbignum:
JNB TI, putbignum
CLR TI

MOV B, #10
DIV AB

ADD A, #'0'
MOV SBUF, A

wait:
JNB TI, wait
CLR TI

MOV A, B
ADD A, #'0'
MOV SBUF, A

RET

getnum:
JNB RI, $
MOV A, SBUF ; leggo il char
CLR RI
CALL PUTC ; l'echo

SUBB A, #'0'
MOV R1, A
JNB RI, $
MOV A, SBUF ; leggo il char
CLR RI
CALL PUTC ; l'echo

cjne A, #CR, NoCr ; e' un CR oppure un numero?
SJMP fine ; e' un CR, ho finito

NoCr:
SUBB A, #'0'
MOV R2, A
MOV A, R1
MOV B, #10
```

```
MUL AB

ADD A, R2
MOV R1, A
fine:
RET

;;;;;
getc:
JNB RI, $
MOV A, SBUF
CLR RI
CALL PUTC
MOV R1, A
RET

;;;;;;;;;;; code data ;;;;;;;;;;;;;
CR EQU 0DH
LF EQU 0AH

input: DB 'number of element (max 99, each max 99): ',
    0
elemento: DB CR, LF, 'element ', 0
duepunti: DB ' : ', 0
acapo: DB CR, LF, 0
algo: DB CR, LF, 'what algorithm you choose? ', CR,
    LF, 0
bubble: DB 'with [b]ubble sorting (less memory, O(n2))
    ', CR, LF, 0
counting: DB 'with [c]ount sorting (more memory,
    O(3n), default) ', CR, LF, 0

vectbuff EQU 0
presenze EQU 100 ; per il countsort
finepresenze EQU 199 ; per il countsort

END
```

10.2 Riepilogo funzioni

Vediamo alcuni frammenti di codice incontrati negli esempi, utili alla programmazione in assembler 8051.

10.2.1 Impostazione porta seriale in modalità a 8 bit con *auto reload*

Si utilizza la prima porta seriale utilizzando il timer 1. Il valore di `TH1` dipende dalla velocità desiderata e dal clock utilizzato dall'8051; in questo caso 230 corrisponde a 2 400 baud con un clock di 12 MHz.
La formula è:

`TH1` = 256 - ((frequenza clock / 192) / baud rate)

```
        ORL TMOD, #20H
        SETB TR1
        MOV SCON, #50H
        MOV TH1, #230
        ORL PCON, #80H
        SETB TI
```

10.2.2 Stampa di numeri, caratteri, stringhe su `SBUF`

Stampa di un numero in `A`.

```
        JNB TI, $
        CLR TI
        ADD A, #'0'
        MOV SBUF, A
        RET
```

Stampa di un numero a tre cifre in `A`.

```
        JNB TI, $
        CLR TI
        MOV B, #100
        DIV AB
        ADD A, #'0'
        MOV SBUF, A
        MOV A, B
        JNB TI, $
        CLR TI
        MOV B, #10
        DIV AB
        ADD A, #'0'
```

```
        MOV SBUF, A
        JNB TI, $
        CLR TI
        MOV A, B
        ADD A, #'0'
        MOV SBUF, A
```

Stampa di un carattere in `A`.

```
putc: JNB TI, putc
        CLR TI
        MOV SBUF, A
        RET
```

Stampa di una stringa all’indirizzo `DPTR`.

```
puts: CLR A
        MOVC A, @A+DPTR
        JNZ more
        RET
more: CALL putc
        INC DPTR
        JMP puts
```

10.2.3 Input con echo di numeri, caratteri, stringhe da `SBUF`.

Richiesta di un numero in `A`.

```
        JNB RI, $
        MOV A, SBUF
        CLR RI
        CALL putc
        SUBB A, #'0'
```

Richiesta di un numero al più due cifre. Il risultato è in `R1`.

```
        JNB RI, $
        MOV A, SBUF
        CLR RI
        CALL putc
        SUBB A, #'0'
        MOV R1, A
        JNB RI, $
        MOV A, SBUF
        CLR RI
```

```
      CALL putc
      CJNE A, #0DH, NoCr
      SJMP fine
NoCr: SUBB A, #'0'
      MOV R2, A
      MOV A, R1
      MOV B, #10
      MUL AB
      ADD A, R2
      MOV R1, A
fine:
```

Richiesta di un carattere in `A`.

```
      JNB RI, $
      MOV A, SBUF
      CLR RI
      CALL putc
```

Richiesta di una stringa in memoria esterna (`@DPTR`) lunga massimo `R1`.

```
gets: JNB ri, $
      MOV A, SBUF
      CLR RI
      CALL putc
      CJNE a, #0DH, NoCr
      SJMP fine
NoCr: MOVX @DPTR, A
      INC DPTR
      DJNZ R1, gets
fine: MOV A, #0
      MOVX @DPTR, A
```

10.2.4 Implementazione *bubble sort*

Massimo 99 elementi, in `A` c'è il numero di elementi.

```
      MOV R5, A
      MOV R6, A
reinizio: MOV B, R5
      MOV R7, B
      MOV DPTR, #indirizzo_inizio_vettore_da_ordinare
ciclo: MOVX A, @DPTR
      MOV R2, A
      INC DPTR
      MOVX A, @DPTR
```

```
      MOV R3, A
      SUBB A, R2
      JBC PSW.7, eminore
      MOV A, R3
      MOV B, R2
      SJMP emaggiore
eminore: MOV A, R2
      MOV B, R3
emaggiore: MOVX @DPTR, A
      DEC DPL
      MOV A, B
      MOVX @DPTR, A
      INC DPTR
      DEC R7
      CJNE R7, #1, ciclo
      DJNZ R6, reinizio
```

10.2.5 Implementazione *counting sort*

Massimo 99 elementi non maggiori di 99. In `A` c'è il numero di elementi.

```
      MOV R6, A
      MOV A, #0
      MOV DPTR, #indirizzo_inizio_vettore_presenze
cleanp: MOVX @DPTR, A
      INC DPTR
      MOV R0, DPL
      CJNE R0, #indirizzo_fine_vettore_presenze, cleanp
      MOV DPTR, #indirizzo_inizio_vettore_da_ordinare
costruisco: MOVX A, @DPTR
      PUSH DPL
      PUSH DPH
      MOV DPTR, #indirizzo_inizio_vettore_presenze
      ADD A, #indirizzo_inizio_vettore_presenze
      MOV DPL, A
      MOVX A, @DPTR
      INC A
      MOVX @DPTR, A
      POP DPH
      POP DPL
      INC DPTR
      DJNZ R6, costruisco
      MOV R0, #0
      MOV DPTR, #indirizzo_inizio_vettore_presenze
      MOV R2, #indirizzo_inizio_vettore_da_ordinare
aggiorno: MOVX A, @DPTR
      CJNE A, #0, conto
      SJMP nonconto
conto: MOV R1, A
```

```
      PUSH DPL
      PUSH DPH
sottociclo: MOV DPTR, #indirizzo_inizio_vettore_da_ordinare
      MOV DPL, R2
      MOV A, R0
      MOVX @DPTR, A
      INC R2
      DJNZ R1, sottociclo
      POP DPH
      POP DPL
nonconto: INC DPTR
      INC R0
      CJNE R0, #numero_elementi_possibili, aggiorno
```

10.3 Note sul software utilizzato

I programmi sono stati compilati ed è stato effettuato il debug sotto *Windows XP* con il *Keil µVision2 IDE* versione 2.38a: esso è una comoda interfaccia al compilatore *C51*, all'assemblatore *A51* ed al linker *BL51* della *Keil*.
Tale suite è disponibile in versione "evalutation", con il limite di 2 kB nella dimensione dei programmi: il sito di riferimento è www.keil.com.
Inoltre per la scrittura sulla memoria dell'8051 dei binari, si è utilizzato l'utility *Flashtools98*, fornito dalla ditta assemblatrice della basetta, www.phytec.de.

10.3.1 Utilizzo di Keil µVision2

Per scrivere un programma 8051 con il "*Keil µVision2*", è necessario creare un progetto e definire quale microcontrollore intendiamo utilizzare. Una volta aperto il "*Keil µVision2*", selezionare Project, quindi New Project... e dare un nome al progetto che vogliamo creare; quindi ci viene presentato un elenco di microcontrollori, tra cui scegliere il nostro. Nonostante la marca non sia OKI, il chip più simile è l'INFINEON C501. Lo selezioniamo ed alla domanda se vogliamo copiare il "Standard 8051 Startup Code" ed aggiungerlo rispondiamo di no.
Bisogna assicurarsi che su View, Project Window sia selezionato; ora per aggiungere un file al progetto appena creato, con il tasto destro sul Source Group 1, selezionare Add Files.

Per compilare i sorgenti, su Project, Rebuild all target files farà il necessario; se vogliamo compilare anche i file oggetti `.hex` (binari da scaricare nella memoria del microcontrollore), con tasto destro su Target 1, selezionare Options for Target 'Target 1', e nella scheda Output, selezionare Create HEX File.

10.3.2 Il debug

La *tool-chain* del *Keil µVision2* conprende un debugger, che serve anche come simulatore di base, chiamato "*Simulator*". Per accedervi sul menù Debug selezionare Start/Stop Debug session.
In questo modo si può eseguire il test e debug delle proprie applicazioni senza provarle fisicamente sulla basetta dell'8051. Il debugger permette la simulazione di diverse periferiche (oltre che naturalmente tutte le istruzioni del microcontrollore), quali le porte seriali e i timer. Esso è facilmente utilizzabile e permette l'inserimento di *breakpoint*, e l'esecuzione passo passo del programma, implementando anche le funzionalità step-over, step-into e step-out per continuare o meno questa modalità di esecuzione (linea per linea) anche nelle sottofunzioni chiamate.
È inoltre possibile richiamare la Watch window che automaticamente visualizza il contenuto delle variabili selezionate durante il debug. Tutte le opzioni citate sono facilmente azionabili cliccando sulle relative icone della toolbar di debug.

Tuttavia esiste anche un'altra modalità di debug, utilizzando l'interfaccia "Keil Monitor 51".
Questa modalità permette di eseguire il debug direttamente sulla basetta, permettendo anche il test delle periferiche. È necessario caricare sulla basetta, oltre al programma che intendiamo debuggare, anche l'interfaccia (detta *monitor*) sul banco di memoria di codice del microcontrollore.
Questo programma permette di effettuare il download del programma utente non nella memoria di codice (la FLASH) ma nella RAM. Infatti cambia l'impostazione di default dell'8051, che prevede un'architettura tipo Harvard con accesso alla parte di codice e di dati attraverso due diverse strutture fisiche di memoria (memoria non volatile la prima, quale una FLASH e una RAM la seconda). Infatti il monitor inizializza un modello di architettura della memoria tipo von Neumann, memorizzando il codice di programma in RAM nel range di indirizzi 0000h-7FFFh. Solo così si è possibile effettuare la modifica del codice a runtime (ad esempio inserendo breakpoint).

Utilizzando la solita interfaccia di debug del Keil µVision2 è possibile interfacciarsi con il monitor della basetta. Tuttavia l'esecuzione di quest'ultimo comporta che si riservino alcune risorse del microcontrollore, quali l'interfaccia seriale (e il relativo interrupt) e il timer 1. Il programma utente non potrà utilizzare queste risorse in quanto riservate per la comunicazione tra il monitor e Keil µVision2.

10.3.3 Utilizzo di Phytec Flashtools98

Dopo aver connesso con il cavo seriale la basetta dl microcontrollore (ovviamente alimentata), si deve premere su entrambi gli interruttori e rilasciare prima il boot e quindi il reset. Apriamo il *Flashtools98* e lo connettiamo con un baud rate qualunque. Quindi sulla scheda Utility, cancellare il banco 1 (il secondo dei due): attenzione, bisogna selezionare solo il Bank 01, il primo (lo 00) contiene il programma di boot e dati di sistema che non dovranno mai essere cancellati (a meno di non sapere quello che si sta facendo), se lo si cancella non sarà più possibile far partire il microcontrollore. Quindi bisogna scaricare sulla memoria il file assemblato `.hex` con Download. A questo punto basta disconnettere il *Flashtools98* e premere il tasto di boot sulla basetta, e se tutto è andato bene il programma è partito.

11. BIBLIOGRAFIA

11.1 L'hardware 8051

- **PHYTEC microMODUL-8051 Hardware-Manual** (Edition April 1999), disponibile nei CD del pacchetto "Spectrum CD" a corredo con la basetta "Phytec u-modul 8051".

- **OKI MSM80C154S/83C164S CMOS 8-bit Microcontroller datasheet**, © *OKI Semiconductors*, 1996.

- **The Final Word on 8051**, *Matthew Chapman*, 1994 (reperibile su Internet in formato pdf).

- **Corso 8051,** *Salvatore Pagano*, disponibile all'indirizzo *http://www.salvatorepagano.brianzaest.it\corsi\corso8051\corso.htm*

11.2 Programmazione Assembler 8051

- **PHYTEC microMODUL-8051 QuickStart Instructions** (Edition december 2002), disponibile nei CD del pacchetto "Spectrum CD" a corredo con la basetta "Phytec u-modul 8051".

- **A51 Assembler A251 Assembler** (Macro Assemblers for the 8051 and MCS® 251 Microcontrollers) User's Guide 04.95, *Vari autori*, © Copyright 1988-1995 Keil Elektronik GmbH., and Keil Software, Inc.

- **8051 tutorial**, Craig Steiner, © Copyright 1997 – 2004.

- **Macro Assembler and Utilities** (Macro Assembler, Linker/Locator, Library Manager, and Object-HEX Converter for 8051, Extended 8051, and 251 Microcontrollers), User's Guide 02.2001, *Vari autori*, © Copyright 1988-1995 Keil Elektronik GmbH., and Keil Software, Inc.

- **A51 8051 Cross-Assembler**, *William C. Colley*, Copyright © 1990-96 Systronix, Inc.

- Le guide presenti su *http://www.8052.com*.

12. INDICE DELLE FIGURE E TABELLE

12.1 Indice delle figure

Fig. 1.1 – Architettura interna 8051: schema a blocchi ... 7
Fig. 1.2 – microMODUL-8051: vista anteriore del modulo ... 8
Fig. 1.3 – microMODUL-8051: vista posteriore del modulo ... 8
Fig. 1.4 – Basetta Phytec ... 9
Fig. 1.5 – Disposizione PIN ... 9
Fig. 2.1 – Tipi di memoria dell'8051 ... 12
Fig. 2.2 – Organizzazione della memoria On-Chip ... 13
Fig. 2.3 – Metodi di funzionamento memoria di programma ... 17
Fig. 4.1 – Tipi di registri SFR ... 26
Fig. 4.2 – La PSW ... 26
Fig. 5.1 – Funzionamento seriale: modo 0 ... 32
Fig. 5.2 – Funzionamento seriale: modo 1 ... 33
Fig. 5.3 – Funzionamento seriale: modo 2 ... 33
Fig. 5.4 – Funzionamento seriale: modo 3 ... 33
Fig. 8.1 – Il processo di scrittura di un programma ... 50

12.2 Indice delle tabelle

Tab. 1.1 – Descrizione piedinatura 8051 ... 10
Tab. 1.2 – Funzioni dei PIN a seconda dell'impostazione del modo di funzionamento del microcontrollore ... 11
Tab. 3.1 – Sintesi dei modi d'indirizzamento ... 22
Tab. 4.1 – Register bank selector RS1 e RS2 ... 27
Tab. 5.1 – Registro SCON: funzione dei singoli bit ... 31
Tab. 5.2 – Registro SCON: funzione dei bit SM0/SM1 ... 31
Tab. 5.3 – Generazione dei principali baud rate ... 35
Tab. 6.1 – I registri SFR per l'uso dei timer ... 38
Tab. 6.2 – I timer: registro TMOD ... 39
Tab. 6.3 – I timer: registro TMOD, bit TxMx ... 40
Tab. 6.4 – I timer: esempio di conteggio ... 40
Tab. 6.5 – I timer: registro TCON ... 41
Tab. 7.1 – Interrupt: indirizzi e polling ... 45
Tab. 7.2 – Gli interrupt: registro IE ... 46
Tab. 7.3 – Gli interrupt: registro IP ... 47
Tab. 9.1 – Instruction set: istruzioni aritmetiche ... 55
Tab. 9.2 – Instruction set: istruzioni logiche ... 56
Tab. 9.3 – Instruction set: istruzioni trasferimento dati RAM interna ... 56
Tab. 9.4 – Instruction set: istruzioni trasferimento dati RAM esterna ... 57
Tab. 9.5 – Instruction set: istruzioni gestione tabelle dati ... 58
Tab. 9.6 – Instruction set: istruzioni booleane ... 59
Tab. 9.7 – Instruction set: istruzioni salto incondizionato ... 59
Tab. 9.9 – Instruction set: istruzioni salto condizionato ... 61
Tab. 9.10 – Instruction set completo ... 62