

# Informazioni su SDCC per MCS51

Questo testo è un tentativo di condividere alcune informazioni sull'uso di SDCC nella programmazione degli MCS51. SDCC è un compilatore open source per i microcontrollori 8051 (nonchè anche altri).

MCS51 è il termine usato da Intel per Micro Controller System. Il "51" è usato per riferirsi alla famiglia di microcontrollori basata sullo stesso core (8031, 8051, 8751, 8032, 8052, ...). Una preziosa fonte di informazioni su tali componenti sono i manuali utente e i data books della Intel.

(Oggi giorno, esistono poi tutta una serie di "derivati" che vanno dalla semplice copia, quindi semplicemente una terza parte da cui approvvigionare il componente, a oggetti che hanno prestazioni migliorate: più veloci, più periferiche, convertitori a bordo, memoria indirizzabile, numero di piedini e ingombri ridotti, fino ad arrivare a coprocessori o interi sistemi on-a-chip. N.d.T.)

## Organizzazione della Memoria

All'inizio, non è facile capire l'organizzazione della memoria dell'architettura degli MCS51. Inizialmente, la Intel aveva come componenti della famiglia i processori 8031/8051/8751. Lo 8051 ha una ROM a bordo per la memoria programma (code), mentre lo 8031 non ha memoria di programma interna (ROMless), ma accede a componenti esterni (EPROMs) che la contengono. Lo 8071 ha una EPROM interna, quindi presenta una finestra per la sua cancellazione.

Questi elementi originali della famiglia hanno tutti 128 bytes di RAM interna per i dati, chiamata memoria diretta (direct data memory).

Successivamente, venne rilasciata una versione diversa del chip con ulteriori 128 bytes di RAM interna, chiamati 8032/8052/8752. Questa memoria è chiamata memoria indiretta, e gli si accede in maniera diversa dai 128 bytes diretti.

Il processore può inoltre indirizzare fino a 64KB di RAM esterna e fino a 64KB di programma (tra memoria interna ed esterna). Il processore esegue i programmi esclusivamente dalla memoria di programma, sia interna che esterna, ma non può eseguire codice memorizzato nelle aree di memoria RAM, sia interna che esterna (questo non è del tutto vero: con alcuni trucchi hardware, o mediante opportuni componenti esterni quali PAL o PLD, è possibile, per le versioni ROMless, far vedere l'area dati come area codice e viceversa. Ciò si usa per esempio quando si vuole aggiornare da seriale il programma contenuto in una FLASH mediante un boot-loader. N.d.T).

Il set di registri è parte dei 128 byte di memoria dati diretta: sono organizzati in 4 banchi da 8 registri a 8bit ciascuno (banchi da 0 a 3). I registri sono normalmente chiamati da R0 a R7. Questi banchi risiedono nei primi 32 bytes (0x00-0x1F). Questa area può essere usata come memoria dati generica, e se uno o più banchi di registri non sono utilizzati, vi si possono memorizzare le proprie variabili.

Generalmente, il banco 0 è il set di registri di default, mentre gli altri tre sono utilizzati nelle Interrupt Service Routine (routine di gestione degli interrupt).

Nei primi 128 bytes ci sono poi alcuni registri particolari (chiamati SFR, Special Function Register) nell'area compresa tra 0x80 e 0xFF, tra i quali l'accumulatore (ACC o A), il registro B, lo

stack pointer SP, ecc. (E' in tale area che vengono allocati gli SFR per accedere alle varie funzioni delle periferiche a bordo del micro: pertanto il loro numero e memoria occupata varia a seconda del processore usato. Attenzione a non sovrascrivere su SFR utilizzati da tali funzioni N.d.T.). Vedere i datasheet del micro utilizzato per la loro allocazione e dettagli d'uso.

Per confondere ulteriormente la faccenda, subito dopo l'area dei banchi di registri, è presente l'area indirizzabile al bit. Sono i bytes compresi tra 0x20 e 0x2F, cioè 16 bytes, accessibili al singolo bit, per un totale di 128 bits. Tali bits sono indicati come 0x00 fino a 0x7F (quindi, il bit di indirizzo 0x01 è il bit numero 1 del byte allocato in 0x20, e il bit 0x0B è il bit 3 del byte allocato in 0x21: semplice no? N.d.T.). Ci sono delle apposite istruzioni che utilizzano efficientemente tali singoli bit, generalmente usati come flag. Se tali bit non vengono utilizzati, i bytes da loro occupati possono essere utilizzati come semplice memoria diretta.

(Anche alcuni degli SFR sono indirizzabili al bit, nella fattispecie quelli il cui indirizzo termina con 0 o 8 in esadecimale. E' il caso di analizzare bene i datasheet per capire bene quali siano e come poterli utilizzare in maniera più efficiente. Ciò è particolarmente importante nella programmazione in assembler, ma anche in C è importante sapere che una variabile può essere contenuta in un solo bit, anziché un char o un int! N.d.T.)

### **Mapa di Memoria**

Per riassumere, esistono le seguenti aree di memoria:

#### DATA segment (DSEG, data)

Indirizzi 0x00-0x7F; include i banchi di registri, l'area indirizzabile al bit e l'area dello stack. SDCC può essere configurato per far iniziare l'allocazione delle variabili a partire da 0x30, cioè dopo i banchi dei registri e l'area al bit. Nel caso non si usino questi ultimi, è possibile partire anche da 0x20, come da qualsiasi altro indirizzo.

#### XDATA segment (XSEG, xdata)

Questa è l'area di memoria esterna, ampia al massimo 64KB, generalmente posta in memorie RAM fisicamente esterne al micro.

#### CODE segment (CSEG, code)

Memoria di programma, ampia al massimo 64KB, che può essere sia interna che esterna al micro. Esiste un apposito piedino del controllore che serve a stabilire se si deve usare la ROM interna o esterna (è il pin EA, External Access. Anche questo è un trucco per utilizzare la ROM interna come boot-loader e quella esterna per il programma in Flash o RAM tamponata da batteria N.d.T.)

#### STACK segment (SSEG)

Normalmente, tale area viene fatta partire dalla fine della area DATA utilizzata per le variabili. Lo stack cresce verso l'alto, e può anche essere posto nell'area indiretta (se avete un 8052). Attenzione: la mappatura di SDCC è accurata solo quando si utilizza l'opzione *--stack-after-data*. Per default, SDCC pone lo stack dopo l'ultimo banco di registri utilizzato. Controllate il codice assembler per essere sicuri.

### IDATA segment (IDATA, idata)

Questa è l'area di memoria RAM aggiuntiva di 128 bytes presente sugli 8032/8052. Lo stack può essere posizionato in tale area, purchè ovviamente questa sia presente (controllare i datasheet).

### BIT segment (BSEG, bit)

Area indirizzabile al bit, che è allocata da 0x20 a 0x2F in RAM diretta. Può essere efficientemente utilizzata per implementare flag a bit (True/False), oppure tralasciata e utilizzata come memoria normale.

### Register Space

Area dei banchi dei registri, che è allocata da 0x00 a 0x1F in RAM diretta. Il primo banco (quello di default) va da 0x00 a 0x07, gli altri sono utilizzabili come banchi registri per le ISR o come area di memoria normale. L'uso di un banco alternativo per le ISR è consigliabile, anzichè il normale salvataggio (push&pop) dei registri utilizzati, in modo da migliorare il tempo di risposta agli interrupt.

## **Considerazioni**

Come si è visto, è una architettura piuttosto complessa, e rende piuttosto difficoltoso se non impossibile definire delle regole ferree nel suo utilizzo. Per il fatto che i banchi registri opzionali e l'area indirizzabile al bit possono essere ridefiniti come area di memoria normale, ci sono tante combinazioni possibili di configurazione. L'area indiretta presente negli 8032/8052 complica ulteriormente la cosa, dato che è completamente assente negli 8031/8051.

Le dimensioni dell'area programma e della memoria esterna possono variare tantissimo. Un sistema può essere composto da soluzioni single chip con appena qualche centinaio di bytes di programma residente nel micro, fino a sistemi con EPROM esterne da 64KB (p.e. la 27C512) e/o una RAM esterna da 64KB, per applicazioni gravose.

La scelta dell'utilizzo delle varie aree di memoria dipende molto da vari fattori, come le esigenze hardware; inoltre, l'efficienza e la velocità del programma possono essere fortemente influenzate da come si utilizzano le varie aree.

Per ricapitolare ulteriormente, SDCC definisce le aree dell'MCS51 come segue:

- data (DSEG) - memoria diretta (128byte, comprensiva i registri, bit-flag, stack)
- idata (ISEG) - memoria indiretta (128byte, presente solo su 8052/8032)
- bit (BSEG) - indirizzamento al bit (parte di DSEG, posta tra 0x20 e 0x2F)
- xdata (XSEG) - area dati esterna (fino a 64KB)
- SSEG - area dello stack (variabile, limitata a 256bytes, in data e/o idata)
- OSEG - area di overlay, in DSEG, dove SDCC sovrappone le variabili
- code (CSEG) - memoria programma, interna o esterna (fino a 64KB)

## Perchè guardare il file MAP

Il file .map generato da SDCC mostra quale e quanta memoria sarà utilizzata dal programma. Se si ignora dove verranno allocate le variabili, probabilmente rischiamo di sbordare lo stack o di sovrapporre aree di memoria in modo non corretto, con malfunzionamento del programma. E' piuttosto importante definire bene l'utilizzo della memoria da parte del programma, per evitare grattacapi successivi.

SDCC mette a disposizione diverse opzioni per definire la configurazione della memoria, come ad esempio la *--stack-after-data*. Non c'è un metodo semplice per determinare se esiste un conflitto, dal momento che tante sono le possibili permutazioni. Tenete presente anche che SDCC non sa se state usando un 8051 o un 8052, quindi se potete usare o meno l'area IDATA.

## Stack - dove e quanto?

Lo stack è utilizzato per salvare temporaneamente registri e variabili al momento di entrare in una ISR o in una funzione. I valori sono poi ripristinati all'uscita. E' necessario sapere quanta memoria avete a disposizione per lo stack e quanta ne verrà effettivamente utilizzata. Se l'utilizzo eccede la disponibilità, generalmente si hanno bizzarri e imprevedibili comportamenti che si concludono con un crash del programma.

Normalmente lo stack è posto in memoria diretta dopo lo spazio allocato per le variabili definite in tale area. Se utilizzate un 8052/8032, lo stack potrà andare oltre il limite di 0x7F, e si estenderà in area indiretta. Se alcune variabili sono state definite in IDATA, lo stack potrebbe sovrascrivere quelle poste all'inizio di tale area. Quindi, per esempio, se si vogliono porre variabili in area indiretta, è meglio porle in indirizzi alti (vicino al limite 0xFF), dove è improbabile che lo stack arrivi.

Si può avere un certo controllo su quanti byte di stack verranno utilizzati. Le ISR possono essere configurate per utilizzare lo stack o un banco registri alternativo. Si può cercare di non nidificare troppo le funzioni, dal momento che ciascuna chiamata occupa come minimo 2 bytes per l'indirizzo di ritorno (come le call in assembler, del resto), mentre le ISR ne occupano almeno 3 (in più, sono salvati i flag).

Se si dà un'occhiata al codice generato, ci si può fare una certa idea delle dimensioni dello stack. Talvolta si può fare una analisi verificando in un sistema (meglio un simulatore) funzionante e poi analizzando il dump della memoria per vedere bene dove è arrivato lo stack.

## Lo stack nel file MAP

Lo stack comincia all'indirizzo SSEG, che si può analizzare nel file .map generato da SDCC. Se vedete che è troppo vicino al limite della memoria interna, dovrete preoccuparvi per eventuali overflow. Generalmente, i sistemi MCS51 richiedono uno stack di almeno 32 bytes. Se utilizzate un micro 8052, conviene porre lo stack nel segmento idata e cercare di porre tutte le altre variabili nell'area data.

## **Il segmento DATA**

Nel file .map si può fare un controllo anche dell'area DATA utilizzata. E' particolarmente importante se utilizzate un micro 8051, cioè senza area IDATA. Se per esempio il segmento DSEG comincia a 0x40 e avete lo stack a partire da 0x60, avrete solo 32bytes (0x20) disponibili per le variabili in area DATA. Se la richiesta eccede tale limite, ci saranno sicuramente problemi.

## **Cosa fare in mancanza di spazio**

Se non avete più spazio, dovete cercare di riorganizzare un po' l'utilizzo della memoria, giochino non sempre facile. Se non utilizzate molte variabili al bit (BSEG), forse potete riutilizzare un po' di byte di tale area (al massimo 16). Se non utilizzate tutti i tre banchi di registri aggiuntivi, potete recuperare anche questi bytes, fino a un massimo di 24. SDCC utilizza alcune tecniche interne per cercare di ottimizzare l'uso della memoria per voi, comunque, è buona norma capire bene dove e come sono allocate le varie aree.

## **Opinioni e considerazioni su gli MCS51**

La famiglia MCS51 pare essere animata da vita propria, tanto che anche la Intel non la controlla più. Tra l'altro, la Intel uscì con la famiglia MCS96 a 16bit, che doveva rimpiazzare la MCS51. Invece, questa ha continuato ad esistere e anzi a moltiplicarsi grazie a una moltitudine di produttori, che producono compatibili a basso costo o dalle prestazioni migliorate. Grazie a ciò, al fatto che erano già utilizzati in molte applicazioni embedded non troppo complicate, al basso costo e alla possibilità di approvvigionamento da più fonti, è diventata indubbiamente la famiglia di microcontrollori più utilizzata specialmente in ambito industriale.

Questo micro lavora egregiamente per piccoli progetti embedded, mentre mostra i suoi limiti nelle applicazioni con software più grossi. Anche se ha una architettura della memoria a dir poco bizzarra, si può contare su una quantità di software già sviluppato veramente impressionante.

Il punto critico del sistema è la limitatezza dello stack. Praticamente, questo è limitato a un massimo di circa 128 bytes. Per piccoli programmi di poche centinaia di bytes, questo non è un problema. Ma se si hanno programmi più consistenti, quali ad esempio una interfaccia utente, il codice può diventare davvero aggrovigliato a causa della mancanza di uno stack appropriato.

(Va fatto notare che in tali casi, però, generalmente si utilizza una struttura di sistema con memoria esterna, e SDCC con il suo modello Large consente l'allocazione dello stack nella memoria esterna, pertanto esteso oltre i limiti dei 256 bytes interni) (N.d.T.).

Un'altra complicazione è la struttura frammentata della memoria, che costringe a dedicare particolare attenzione alla allocazione della stessa. Il risultato può sembrare un po' caotico. Poi, la memoria esterna è accessibile solo tramite un modo di indirizzamento indiretto, tramite il DPTR. Anche se ormai molti derivati mettono a disposizione più di un datapointer e anche istruzioni per gestire i decrementi (non possibili negli 8051 standard), tale gestione è molto poco efficiente, confrontata con quanto offrono altri processori.

A parte il costo, un'altra ragione che spinge all'utilizzo di tale processore è la sua disponibilità, anche nel tempo. Un 80C31 probabilmente sarà disponibile per un'altra decina d'anni; non credo che ciò si possa dire per molti altri micro, eccetto forse i Motorola 68HCxx che paiono avere un seguito devoto simile agli MCS51. Se guardiamo per esempio agli 8088, pensate che si possano ancora acquistare?

La progettazione di sistemi embedded contiene scelte di autodifesa per applicazioni a piccoli volumi produttivi. Se avete grandi volumi, la disponibilità non è un grandissimo problema, potete anche permettervi di riprogettare; ma se producete 50 o 100 pezzi l'anno, non potete permettervi di riprogettare il tutto ogni pochi anni.

Karl.

Traduzione non troppo letterale a cura di Andrea Contrucci