ASSEMBLER 8086 8088

di Pierluigi Panunzi

La struttura interna dei due microprocessori dal punto di vista software

Siamo oramai abbastanza avanti nella conoscenza di questi due microprocessori dal punto di vista «fisico», e cioè della loro struttura interna, rappresentata da due parti fondamentali: è meglio, prima di andare avanti, fare una specie di riassunto.

La prima parte dunque è detta «Execution Unit» (EU) ed è principalmente preposta all'esecuzione di calcoli logico-aritmetici in generale utilizzando un certo numero di registri dati, quattro per l'esattezza, due registri «indice» nonché altrettanti registri «puntatori».

La seconda parte invece rappresenta tutta la circuiteria e la logica di interfacciamento con l'esterno, sfruttando due di tre sotto-blocchi rispettivamente per l'interfaccia verso il bus (BIU, Bus Interface Unit) e per la generazione e la gestione dei segnali di controllo sia in input che in output.

Il terzo sotto-blocco è quello che ci interesserà di più in quanto è strettamente legato alla programmazione: sappiamo già che è formato dai 4 registri di «segmento» e dall'«Instruction Pointer» (IP, il ben noto program counter degli altri microprocessori).

Come già ripetuto più volte, i registri di quest'ultimo sotto-blocco sono fondamentali ed utilizzati praticamente nel corso di ogni istruzione Assembler: la loro importanza comporta che non si può iniziare a programmare in Assembler se non si sono ben comprese le caratteristiche e le funzioni di tali registri.

Intendiamoci: non è certo difficile programmare con l'8086, ma si deve sempre tenere in mente la struttura logica di un programma Assembler, struttura che ora andremo ad analizzare.

La struttura logica di un programma Assembler 8086/8088

Per affrontare il problema della struttura logica di un programma in Assembler bisogna aver felicemente superato il primo ostacolo consistente nella definizione dettagliata di quali sono le funzioni che il programma deve svolgere, quali dati deve ricevere in input e quali valori deve fornire in output.

In pratica daremo per scontata questa fase, che tuttavia non è per nulla da sottovalutare: definiti cioè gli obiettivi che si prefigge il programma, cominceremo ad isolare le varie parti di esso costruendo così un insieme di «moduli» o «procedure», relativi ad una certa funzione, ad esempio un modulo per l'inizializzazione, uno per l'input dei dati, uno per l'elaborazione vera e propria, ecc. In questo modo ci risulterà più agevole il compito di debuggare il programma, isolando di volta in volta gli errori dei singoli moduli.

Il fatto poi di suddividere il programma in vari moduli non ne riduce certo le potenzialità, anzi, basta cambiare un certo modulo per far compiere al programma altri tipi di operazioni.

Viceversa, nonostante il frazionamento, il programma rimane di certo un'unica entità: ciò si riscontra soprattutto a livello Assembler, per il quale il nostro programma sarà, sì, suddiviso in un certo numero di parti, ma anche inglobato in un unico segmento, il Code Segment.

In dipendenza dal fatto di aver suddiviso il programma in vari moduli, dovremo in generale gestire altrettanti insiemi di dati sui quali operano i moduli, presi singolarmente: tutti questi nuovi moduli di dati andranno a formare quello che abbiamo già definito come Data Segment.

Infine, dato che ogni modulo avrà le sue chiamate a subroutine (CALL) nonché salvataggi nello stack (PUSH), con i relativi ripristini (POP), ecco che avremo in generale un certo numero di livelli di stack per ogni modulo, da riunire in un unico stack, in un insieme detto Stack Segment. Del quarto tipo di segmento, l'Extra Segment, non parleremo ora, ma solo in seguito, quando se ne presenterà la necessità.

Analizziamo il Code Segment

Visto dunque in generale quali sono gli elementi che compongono quegli insiemi chiamati Segmenti (di dati, di codice e di stack), analizziamo ora in dettaglio il primo tipo di segmento.

A completamento di quanto detto finora, all'interno del Code Segment possono trovare posto una o più «entità» completamente a sé stanti che prendono il nome (e qui può nascere una certa confusione...) di «SEGMENT»: con tale nome si intende un modulo logico, una parte di programma o al limite un programma vero e proprio, racchiuso, a livello di «testo da assemblare», dalle direttive per l'Assembler

<nomemodulo> SEGMENT

e

<nomemodulo> ENDS

una sorta di «begin-end» del Pascal.

Di moduli logici, «segment», strutturati in questo modo:

```
nome        SEGMENT
            ...
            istruzioni
            ...
nome        ENDS
```

abbiamo detto che ne possono comparire più di uno, a discrezione del programmatore ed a seconda del grado di difficoltà raggiungibile dal programma.

Non a caso parlando di «segment» gli abbiamo associato l'attributo di «entità a sé stante»: infatti un qualunque «segment» può contenere singolarmente istruzioni di programma fino ad un'occupazione totale di memoria di 64 kbyte!

Seguendo il ragionamento dello scorso paragrafo, dal momento che si tratta di entità logiche separate, è lecito aspettarsi che ad ogni «segment» di programma corrisponda un altro «segment», stavolta relativo ai dati, alle costanti ed alle variabili usate dal «segment» di programma in esame.

A complicare ulteriormente le idee, anche i «segment» contenenti i dati sono a tutti gli effetti delle entità ben separate le une dalle altre, tant'è che analogamente al caso del codice, il «segment» di dati è indicato nel modo seguente:

```
nome        SEGMENT
            ...
            dichiarazioni e definizioni
            di variabili
            ...
nome        ENDS
```

Inoltre c'è da dire che l'entità logica identificata come «segment» di codice può a sua volta contenere al suo interno delle sotto-entità, cioè dei sotto-moduli, cioè dei sotto-programmi e perciò subroutine le quali, per la gioia di chi legge il programma, possono essere indicate come «PROCedure», con tanto di direttive di apertura e di chiusura: una subroutine richiamata più volte guadagnerà in leggibilità se verrà scritta nel modo seguente:

```
nome        PROC
            ...
            istruzioni della subroutine
            ...
nome        ENDP
```

I programmatori che conoscono il Pascal ben sanno quanto giovamento porta l'essere in un certo senso costretti ad «inquadrare» il proprio programma secondo una ben precisa serie di regole: ma mentre ciò era finora un pregio del Pascal e perciò di un linguaggio ad alto livello, ecco che ora si ritrova anche a livello più basso. Ai lettori che hanno trovato complicati i ragionamenti precedenti consigliamo di farsi coraggio e procedere nella lettura, perché ora viene qualcosa, se vogliamo, ancora di più delicato...

I registri di segmento

Abbiamo dunque più volte parlato di quelle entità chiamate «segment»: fissiamo l'attenzione su uno di tali «segment» e per la precisione uno di quelli contenenti linee di programma.

Ecco che ora dobbiamo richiamare in causa un oggetto che nel frattempo pareva quasi scomparso dalla circolazione (il nostro microprocessore!), per vedere come si comporta nel gestire tutte queste entità: il tutto avviene grazie ai quattro «segment register», che ricordiamo essere

CS relativo al Code Segment
DS relativo al Data Segment
SS relativo allo Stack Segment
ES relativo all'Extra Segment

In particolare il registro CS serve a definire quale particolare «segment» di codice stiamo trattando il DS indica quale «segment» di dati contiene le variabili usate dal programma, lo SS indicherà una zona in cui risiederà lo stack, mentre per l'ES valgono infine le stesse considerazioni del DS, ma relative ad altri dati che eventualmente necessitano.

Estendendo dunque il ragionamento non già ad un solo «segment» di codice, ma a più «segment», si può vedere che ad ognuno di essi corrisponde così un certo *valore* del CS ed analogamente ad ogni altro segment di dati corrisponderanno altrettanti *valori* del registro DS; perciò sia nel caso di un unico segment di codice che, a maggior ragione, nel caso di svariati segment, è fatto obbligo al programmatore di inizializzare correttamente e prima di ogni altra cosa i registri di segmento. Vediamo ora come si fa con un esempio pratico:

```
PROGRAMMA   SEGMENT
            ASSUME CS:PROGRAMMA,DS:DATI,
                   SS:STACK,ES:NOTHING
            MOV AX,DATI
            MOV DS,AX
            MOV AX,STACK
            MOV SS,AX
            ...
            istruzioni
            ...
PROGRAMMA   ENDS
DATI        SEGMENT
            ...
            definizioni varie
            ...
DATI        ENDS
STACK       SEGMENT
            ...
            definizione dello stack
            ...
STACK       ENDS
```

Iniziamo da «PROGRAMMA»: come abbiamo appena detto bisogna, ogni volta che si definisce un segment con l'omonima direttiva, assegnare un valore ai quattro registri.

In particolare questa assegnazione va innanzitutto comunicata all'assemblatore tramite la direttiva *«ASSUME»* seguita dal nome del segment register e dal segmento a cui si deve riferire.

Vediamo perciò nell'esempio che CS ora dovrà fare riferimento al segment chiamato «PROGRAMMA», DS invece dovrà fare riferimento a «DATI» e così SS a «STACK». Dato che in questo caso supponiamo di non usare l'Extra Segment, ecco che la direttiva ASSUME assegnerà ad ES un valore pari a «NOTHING».

Effettuata perciò questa assegnazione «soft», *bisogna* ora inizializzare in modo «hard» i registri di segmento in modo che l'8086/8088 possa usarli.

Per far ciò usiamo delle semplici istruzioni di «move» (MOV), grazie alle quali si assegnerà, passandolo attraverso il registro AX, il «valore» del segment DATI al DS ed il «valore» del segment STACK ad SS.

I lettori più attenti, oltre all'ovvia «non assegnazione» del registro ES in quanto non utilizzato (ricordate il «NOTHING»?), avranno senz'altro notato che inoltre *non* si è effettuata l'inizializzazione del CS a livello «hard»: questa è una regola generale in quanto non esiste un'istruzione di caricamento diretto di tale registro, anche perché sarebbe del tutto inutile.

A tempo debito vedremo come ciò avviene viceversa automaticamente o con un'istruzione implicita: per i più impazienti diciamo che ciò si ottiene ad esempio o con una «long jump», con un particolare salto indiretto o con una routine di interrupt o al reset...

Tornando ora al caso di più segment, il programmatore deve perciò porre, come prime istruzioni di ogni segment di codice, l'inizializzazione corretta e completa dei segment register: lo ripetiamo ancora una volta in quanto altrimenti il programma non potrà mai funzionare; viceversa, dimenticando qualcosa si otterranno già a livello Assembler delle segnalazioni di errore.

Bisogna infatti tenere sempre bene in mente il fatto che, come sono noti «a noi» in base ai nostri ragionamenti, i segment register devono essere noti sia all'assemblatore che al microprocessore: per avvalorare questo ragionamento mostriamo subito un esempio in cui si dimentica qualcosa e perciò accade qualcosa di insolito.

Supponiamo di avere la seguente situazione:

```
PROGR1      SEGMENT
        ASSUME CS:PROGR1,DS:DATI1,
               SS:STACK,ES:NOTHING
            ...
            MOV AX,PIPPO
            ...
PROGR1      ENDS
PROGR2      SEGMENT
        ASSUME CS:PROGR2,DS:DATI2,
               SS:STACK,ES:NOTHING
            ...
PROGR2      ENDS
DATI1       SEGMENT
            ...
DATI1       ENDS
DATI2       SEGMENT
            ...
PIPPO       DW ?
            ...
DATI2       ENDS
STACK       SEGMENT
            ...
STACK       ENDS
```

in cui vi sono due segment di codice ed i rispettivi segment di dati.

Osserviamo la direttiva ASSUME all'interno di «PROG1»: in essa abbiamo associato a DS il segment di dati «DATI».

Ciò significa che sia l'assemblatore che il microprocessore (tramite le successive MOV che abbiamo tralasciato) andranno a cercare le variabili nel segmento indicato all'inizio nell'ASSUME (per l'assemblatore) e contenuto nel DS (per il microprocessore): nel caso della variabile PIPPO (definita come «word» tramite la direttiva DW ed alla quale non assegniamo alcun particolare valore, grazie al «?») si ha che essa appartiene ad un segmento diverso da quello contenuto in DS e come tale risulterà «irraggiungibile» dall'assemblatore.

Questo fatto verrà segnalato dall'assemblatore stesso con un apposito messaggio d'errore in cui viene evidenziato l'aspetto di «non raggiungibilità» di una certa variabile: a nulla vale l'obiezione «ma io la variabile PIPPO l'ho definita!» e non è però nemmeno questo il momento di vedere la soluzione di questo problema alquanto delicato, soluzione che non è per nulla intuitiva.

Una pausa di riflessione

Come è lontano il mondo degli 8 bit e viceversa come è irto di insidie quello dei 16 bit...

Abituati come siamo a definire variabili qua e là nei nostri programmi Assembler, non ci sfiora nemmeno lontanamente l'idea che l'assemblatore si rifiuti (è proprio il caso di dirlo!) di tradurre un'istruzione in cui appare una variabile che noi abbiamo definito, seppure in un altro segmento...

Abbiamo detto dei nomi delle variabili: la stessa cosa succederà, se non stiamo più che attenti, anche nel caso delle «label», all'interno di istruzioni di salto o di chiamate a subroutine, a meno che non prendiamo le debite precauzioni, definendo opportunamente le label stesse, come vedremo meglio in dettaglio nel seguito.

Proseguendo perciò nelle nostre riflessioni, si sta sempre più affinando nelle nostre idee il concetto di «segment», inteso, lo ripetiamo ancora una volta, come entità a sè stante e perciò completamente distinta dagli altri segment: è come se parlassimo di un programma scritto ad esempio in Assembler Z80 e che fa riferimento alle variabili di un altro programma...

Naturalmente il paragone va preso con le dovute cautele: in realtà i programmi posti in segmenti diversi sono sì da considerare separati, ma ovviamente devono poter comunicare, altrimenti tanto varrebbe non scriverli per niente insieme.

Ricordiamoci poi, e questo si comincia già ad intravvedere da questa puntata, che con l'86/88 si possono creare sistemi multi-utente e/o multitasking, nel qual caso rispettivamente un utente od un task non è tenuto, non deve, non può (a meno di controindicazioni) sapere ciò che un altro utente o task sta facendo o ha fatto.

Conclusioni

Al termine di questa ponderosa puntata conviene esaminare i concetti sin qui esposti con gli occhi di chi già conosce l'Assembler dell'86/88: gli «addetti ai lavori» probabilmente troveranno nuova e strana l'impostazione puramente «software» dell'argomento «segment».

Basta infatti sfogliare un qualsiasi manuale dell'Intel, riguardante appunto questi processori, per trovarsi sommersi di motivazioni «hardware» nella spiegazione del concetto di «segment», il tutto condito con somme di contenuti di registri per ottenere il sospirato indirizzo di una certa variabile o etichetta che sia. Riteniamo che già così come è stato esposto l'argomento necessiti delle debite pause di riflessione e di «digestione»: la prossima puntata introdurremo un altro concetto importantissimo (l'«offset»), che ci servirà a chiarire alcuni punti rimasti oscuri. MC

DA OGGI È MUSICA PER TUTTI CON

SOUND BUGGY

nuovidea

Musica dal calcolatore, musica vera, musica tua!

Con l'unità periferica SOUND BUGGY, e la tastierina musicale SIEL da sovrapporre a quella alfanumerica, il tuo Commodore 64 si trasforma in un'autentica band.

Se sei già esperto di musica SOUND BUGGY ti porterà alla perfezione. Se sei un principiante diventerai, in pochi giorni, concertista e arrangiatore, comporrai musica tua e potrai ascoltarla in una perfetta registrazione elettronica, collegandoti a ogni impianto stereo, videotelevisivo, monitor C 64.

UN ECCEZIONALE PACCHETTO DI PROGRAMMI

Grazie allo straordinario software di SOUND BUGGY potrai eseguire o comporre su 24 ritmi (12 preregistrati), disporrai di ben 28 timbri strumentali (14 preregistrati), correggere, migliorare, registrare.

In più, tramite interfaccia MIDI, SOUND BUGGY comunica anche con expander, sintetizzatori, sequencer ecc.

Insomma, SOUND BUGGY è un vero prodigio dell'elettronica al servizio della tua creatività musicale.

Questa Cedola rappresenta l'unico modo di ordinare SOUND BUGGY, e riceverlo completo di tastierina e di programmi su disco e cassetta. ***Ritagliala e spediscila subito.***

CEDOLA PRIVILEGIATA DI ACQUISTO SOUND BUGGY

Da inviare in busta chiusa a: **"Filodiretto SIEL"**
SIEL Società Industrie Elettroniche s.p.a.
CASELLA POSTALE 199 - 63039 S. Benedetto del Tronto

☐ Sì, desidero acquistare SOUND BUGGY, la vostra unità periferica per C 64. Speditemela contrassegno completa di **minitastiera, pacchetto software** sia su disco che su cassetta, **libretto istruzioni** al prezzo speciale di L. 185.000 (incluse L. 27.650 IVA e L. 3.750 di spese postali). È inteso che il mio SOUND BUGGY sarà protetto da **Garanzia per 1 anno.**

SIEL®

Nome _____ Cognome _____

Indirizzo _____

CAP _____ Località _____

Data _____ Firma _____
(per i minori occorre quella del genitore)

MAXIPRESTAZIONE IN MINISPAZIO
Il tuo laboratorio musicale, completo e perfetto, è tutto qui: il Commodore 64, il SOUND BUGGY, la minitastiera SIEL, il software.