ASSEMBLER 8086 8088

di Pierluigi Panunzi

La gestione della memoria

Nelle scorse puntate di questa rubrica, abbiamo parlato più volte del fatto che i due microprocessori in esame sono in grado di gestire quantità a 16 bit sia provenienti dalla memoria, sia tramite una serie di registri interni anch'essi a 16 bit. Abbiamo inoltre detto che per facilitare il compito del programmatore in alcune particolari applicazioni, tanto la memoria quanto i registri interni (AX, BX, CX, DX) possono essere visti sotto forma di byte e perciò come quantità ad 8 bit (i registri summenzionati diventano AL, AH, BL, BH, CL, CH, DL e DH).

Ulteriore possibilità per il programmatore è di vedere la memoria anche come dati a 32 bit («double-word»), fatto che risulterà molto utile nel seguito.

Altra caratteristica dei due microprocessori di cui abbiamo ampiamente parlato è il fatto che possono indirizzare fino ad 1 Megabyte di memoria, grazie ai 20 bit di indirizzamento a disposizione.

Il fatto dunque di avere un generico indirizzo a 20 bit e non a 16, come siamo abituati nei microprocessori ad 8 bit, comporta, come è facile vedere, che un indirizzo di una locazione di memoria non può essere completamente espresso dal contenuto di un registro: mancherebbero infatti i quattro bit più significativi ed anche se si riuscisse nell'intento, si ridurrebbe drasticamente l'area di memoria indirizzabile da 1 Megabyte ai soliti 64K byte. Ecco che perciò i progettisti dell'Intel hanno ideato un particolare metodo di «concatenazione» di registri per ottenere ogni volta un indirizzo completo a 20 bit.

Ma torniamo un istante al caso dei microprocessori ad 8 bit; ad esempio nell'8080 e nello Z80 è possibile saltare alla locazione di memoria il cui indirizzo è posto nella coppia di registri HL, effettuando perciò un cosiddetto «salto indiretto»: ciò si ottiene rispettivamente con le ben note istruzioni «PCHL» e «JP (HL)».

Nel caso dell'86/88 invece, come detto, con un singolo registro a 16 bit si potrebbero indirizzare solo 64K byte di memoria: si ha perciò la necessità di introdurre un altro registro che fornisca i quattro bit mancanti.

La riflessione che in generale in un programma si possono distinguere tre zone, una riservata alle istruzioni del programma, una riservata alle variabili ed in genere alle locazioni usate dal programma per la memorizzazione di dati ed una riservata esclusivamente allo stack, ha fatto propendere i progettisti dell'Intel all'introduzione dei ben noti quattro Segmenti i cui primi tre sono appunto riservati al codice di programma (Code Segment), all'insieme dei dati su cui il programma opera (Data Segment), allo Stack (Stack Segment), mentre il quarto è stato riservato per particolari applicazioni in cui si ha necessità di un'ulteriore zona dati (Extra Segment).

Il fatto essenziale in tutto questo ragionamento è che ognuno di questi quattro segmenti è ampio 64K byte e perciò ogni singolo byte al suo interno può essere indirizzato con una semplice word (parola a 16 bit) contenuta in un registro interno oppure in una coppia di locazioni della memoria: in particolare, e ciò lo ritroveremo sempre nel seguito, l'indirizzo che ha una certa locazione di memoria nell'ambito del suo segmento, e perciò il valore memorizzabile in un registro interno, prende il nome di «offset».

Ripetiamo che è importantissimo avere ben chiaro in mente il concetto appena espresso di «offset», in quanto entra in gioco in ogni istruzione, anche quando uno meno se lo aspetta e perciò non solo quando il programmatore è forzato ad indicarlo specificatamente.

Per chiarire perciò il concetto di offset facciamo un esempio. Supponiamo di considerare un segmento di dati (Data Segment), formato (ormai dovrebbe essere chiaro) da al massimo 64K byte oppure da 32K word oppure ancora da 16K double-word, a seconda delle scelte del programmatore: ovviamente i tre tipi di dati possono essere ben mischiati tra loro, come dire che un Data Segment può essere formato da byte, word o double-word mischiati fra loro, ma sempre in numero massimo pari a 64K byte.

Supponiamo dunque per semplicità di considerare solamente byte e di fissare l'attenzione su uno qualsiasi dei 64K byte, ad esempio il ventesimo: ben consci del fatto che nel mondo dei computer si comincia sempre a contare da 0 e non da 1, allora si dirà che l'offset di quel certo byte è pari a 19, 13H se espresso in esadecimale. Il byte successivo avrà un offset pari a 14H, come è ovvio, e così via fino all'ultimo, che avrà un offset pari a FFFFH. Però noi in genere identifichiamo una certa cella, nell'ambito di una zona dati, con un'etichetta (label), che la identificherà in tutto e per tutto nel corso del programma.

Infatti noi sappiamo bene che si può parlare della cella di memoria «ALFA», in quanto ciò è più mnemonico che non un numero rappresentante il suo indirizzo, ed inoltre ben conosciamo la differenza che passa tra «ALFA» (inteso come etichetta) ed «il contenuto di ALFA» (che può essere un byte o una word a seconda della definizione di ALFA).

Nel caso dunque di microprocessori ad 8 bit, in genere parlando di «ALFA», nel senso di etichetta, indichia-

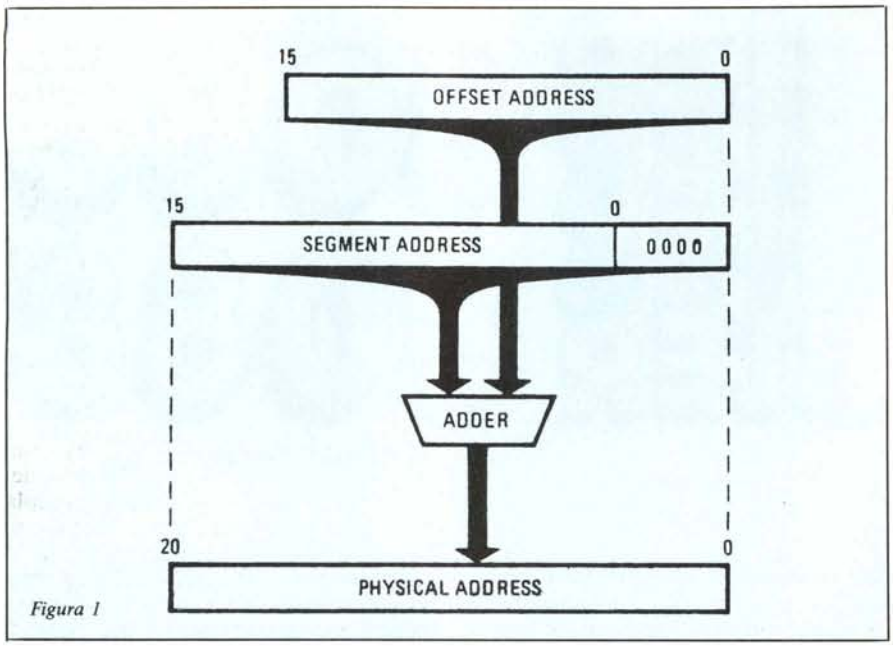

Figura 1

mo l'indirizzo fisico al quale è posta la cella di memoria chiamata «ALFA» e questo fatto lo ritroviamo ad esempio nella «symbol table» del nostro programma una volta assemblato, dove troveremo a fianco ad ALFA il suo indirizzo fisico e non già il suo contenuto... che ovviamente non è noto.

Supponendo poi che ALFA identifichi una coppia di byte e perciò una word, automaticamente dicendo «il contenuto di ALFA» intenderemo non certo il suo indirizzo, ma ciò che è posto all'interno di ALFA.

Tornando al caso dei nostri microprocessori 86/88, il tutto rimane ovviamente invariato, a parte il fatto che parlando di «ALFA» si intenderà il suo offset e cioè la sua posizione relativa all'inizio del Data Segment; ci teniamo che ciò risulti ben chiaro: «ALFA» non sarà più un indirizzo fisico, ma un indirizzo relativo, logico, che si chiama «offset».

Ecco che assemblando un programma in Assembler 86/88, alla fine, nella «symbol table» o «cross reference» che dir si voglia, troveremo, accanto ai nomi delle etichette, il loro offset.

Analogo discorso vale se consideriamo un'etichetta di un programma, ad esempio l'etichetta che identifica una certa subroutine, come pure l'etichetta a cui si salta se si è verificata una certa condizione: anche in questo caso l'etichetta identificherà l'offset di una certa istruzione, nell'ambito del Code Segment e non più un indirizzo fisico.

Ciò che abbiamo detto per il Code Segment e per il Data Segment è valido anche per i due Segment restanti: comunque un'etichetta identifica l'offset di una locazione di memoria (un dato o un'istruzione) nell'ambito del Segment in cui tale locazione si trova.

I registri di segmento

Stabilito dunque cosa si intende per «offset», introdurremmo ora il concetto di «base» (è inglese!), strettamente legato al concetto che già conosciamo di «Segment Register».

Sappiamo già dalle scorse puntate che ad ognuno dei quattro segmenti corrispondono altrettanti registri detti appunto «Segment Register», chiamati CS, DS, ES e SS rispettivamente per il Code Segment, per il Data Segment, per l'Extra Segment e per lo Stack Segment.

La parola «corrispondono» usata poc'anzi sottintende il fatto che il singolo Segment Register stabilisce in un certo qual modo l'indirizzo fisico iniziale del relativo segment e cioè del primo byte costituente il Segment in questione.

In particolare, per comprendere in quale modo indichi l'indirizzo iniziale del suo Segment, ritorniamo un attimo indietro.

Il fatto che i bit di indirizzo siano 20, comporta che gli indirizzi fisici raggiungibili dal microprocessore sono compresi tra i valori (esadecimali) 00000H ed FFFFFH, rappresentati perciò con cinque cifre esadecimali: per confronto un processore ad 8 bit con 16 bit di indirizzamento, potrà puntare locazioni di memoria i cui indirizzi variano tra 0000H ed FFFFH (e cioè formati da 4 cifre esadecimali).

Inoltre, a livello di terminologia, sappiamo che una «pagina» è un insieme di 256 byte: sappiamo ad esempio che la «pagina 0» è quella che va dall'indirizzo 0000H all'indirizzo 00FFH e viceversa dato l'indirizzo 1234H, sappiamo che tale cella si trova nella «pagina 12H», in cui il valore 12H si ottiene eliminando le due cifre meno significative dell'indirizzo.

Nel nostro caso dobbiamo introdurre un altro concetto facile facile, probabilmente già esistente in altri microprocessori: quello del «paragrafo».

Con tale termine si intende un gruppo di 16 byte ed analogamente al caso della «pagina», si ottiene dall'indirizzo di una certa locazione eliminando stavolta una sola cifra meno significativa: ad esempio la cella di cui sopra (1234H) appartiene al paragrafo 123H, così come tutte le celle poste tra gli indirizzi 1230H ed 123FH. Semplice, no?!

Tornando dunque al nostro microprocessore, è molto comodo pensare il Megabyte suddiviso in paragrafi (sono 65536, per l'esattezza) ed ecco che tra gli indirizzi fisici 00000H ed FFFFFH esisteranno 65536 paragrafi, dal «paragrafo 0000H» al «paragrafo FFFFH».

I lettori più arguti avranno già capito che ora il numero di un paragrafo è formato da 16 bit e come tale può essere benissimo depositato in un registro interno, guarda caso proprio un segment register...

Ora siamo dunque pronti: un segment register (ad esempio DS, ma lo stesso vale anche per gli altri 3) deve sempre contenere il numero del paragrafo a partire dal quale si estende il Segment (il Data Segment nel nostro esempio).

Ad esempio, se diciamo che il Code Segment è posto a partire dal «paragrafo 5DAH» allora tale valore verrà posto nel registro CS: facendo rapidissimi calcoli, si vede che l'indirizzo iniziale fisico del Code Segment sarà dato da 5DA0H, ottenuto moltiplicando per 16 il numero del paragrafo, o meglio shiftando di quattro bit a sinistra il numero del paragrafo o ancora meglio aggiungendo a destra del paragrafo uno 0!

Siamo oramai espertissimi nel dire che il paragrafo «3458H» inizia all'indirizzo 34580H, mentre viceversa l'indirizzo EEEEEH è posto nel paragrafo «EEEEH».

Acquisita quindi dimestichezza con questo termine nuovo, ritorniamo all'offset con un quesito, che andremo subito a risolvere.

Supponiamo di sapere che il registro DS contiene il valore 100H e che la variabile ALFA è posta ad un offset di 1111H: a quale indirizzo fisico è dunque posta la cella ALFA?

Il conto si esegue facilmente: il paragrafo del DS è 100H il che vuol dire che l'indirizzo fisico della sua prima cella sarà dato da 1000H. Ora la cella il cui offset è 1111H si troverà ad un indirizzo fisico pari alla somma di 1000H e di 1111H e cioè l'indirizzo desiderato di ALFA varrà 2111H.

Tutto questo l'abbiamo detto «per dovere di cronaca» in quanto in generale non ci interesserà conoscere l'indirizzo fisico di una certa locazione di memoria, se non in casi particolarissimi: quello che invece si dovrà sempre conoscere è il suo indirizzo logico, quasi virtuale, dato dall'insieme delle informazioni «offset» e «base», che, quasi ci dimenticavamo di dire, non è altro che il numero del paragrafo, contenuto nel registro di Segment.

Tornando perciò alla nostra cella ALFA noi diremo che il suo offeset è pari a 1111H e la sua «base» è 100H: se proprio qualcuno ci chiedesse a quale indirizzo fisico si trova, allora dovremmo fare quelle operazioni semplicissime di cui sopra ed in questo caso varrà 2111H. Attenzione che la «base» è quella citata e non il paragrafo a cui appartiene la cella, che in questo caso sarebbe 211H...

Questo lo diciamo in quanto a pensarci bene, in un segmento (di 64K byte) esistono ben 4096 paragrafi, ma l'unico che ci serve è il primo e cioè quello a partire dal quale inizia il segmento stesso.

Ora che siamo edotti su questo argomento possiamo ben comprendere il perché, non appena definiamo un certo segmento di codice con la direttiva «SEGMENT» dobbiamo per prima cosa inizializzare i quattro registri di segmento: solo in questo modo il microprocessore saprà il numero di paragrafo dal quale inizieranno i quattro segmenti. Senza tale inizializzazione, bene che vada i quattro segment register conterranno valori casuali per cui si può già intravedere cosa potrà succedere... Ed abbiamo detto «bene che vada»...

Come si comporta il microprocessore

Dopo tanta teoria andiamo a vedere come si comporta il microprocessore quando deve fare riferimento alla memoria.

Innanzitutto cominciamo col dire che all'interno del micro, nella parte identificata come BIU (Bus Interface Unit) è presente un sommatore a 20 bit (vedasi la figura 1) che consente di effettuare istante per istante la somma tra un offset e la relativa base, quest'ultima moltiplicata per 16.

Questo sommatore viene attivato praticamente ad ogni istruzione e cioè tutte le volte in cui si deve far riferimento ad una locazione di memoria: il microprocessore saprà istante per istante quali sono gli operandi da porre come due addendi del sommatore:

— nel caso di un indirizzo di una locazione di memoria contenente un'istruzione e perciò posto all'interno del Code Segment, allora come «base» prenderà il contenuto del CS e come offset il valore attuale del registro IP (Instruction Pointer): effettuata la somma, in tempi brevissimi dato che il sommatore è di tipo «hardware», porrà il valore ottenuto (a 20 bit) nell' Address Bus, indirizzando così la cella desiderata.

— nel caso in cui la locazione di memoria contenga un dato e perciò si trova all'interno del Data Segment, la «base» sarà il contenuto del DS, mentre l'offset sarà proprio quello fornito dall'istruzione che fa riferimento alla cella di memoria desiderata.

— nel caso in cui si effettui un'operazione coinvolgente lo stack, allora la «base» sarà il registro SS, mentre l'offset sarà fornito dal contenuto del registro SP (lo Stack Pointer).

— infine nel caso di una locazione posta all'interno dell'Extra Segment, allora l'indirizzo verrà generato ponendo nel sommatore come «base» il registro ES e come offset il valore fornito dall'istruzione che fa riferimento alla cella di memoria desiderata.

Ricapitolando possiamo sintetizzare il tutto nel seguente schema:

locazione	base	offset
con istruzione	CS	IP
contenente dati	DS	da programma
operazioni su stack	SS	SP
contenente dati extra	ES	da programma

Come già detto, questo meccanismo viene ripetuto anche parechie volte nel corso dell'esecuzione di una singola istruzione, ma di certo non se ne risente a livello tempi di esecuzione, in quanto già per ogni singola istruzione è noto il numero di volte in cui verrà effettuata una somma a 20 bit e quindi già se ne tiene conto nel computo globale della durata dell'istruzione.

Conclusioni

Con questo abbiamo terminato la presente puntata riguardante l'Assembler dell'8086 e dell'8088: nella prossima approfondiremo l'argomento studiando vari casi che si possono creare a seconda della scelta che si compie sul valore da assegnare ai registri di segmento.

Viceversa vedremo, in base al caso contingente e perciò in base alla quantità di memoria a disposizione, quali sono le migliori strategie per la gestione della memoria ed a quali inconvenienti si può andare incontro. MC