ASSEMBLER 8086 8088

di Pierluigi Panunzi

Le direttive dell'assembler
(prima parte)

Da questa puntata ci occuperemo in dettaglio delle direttive dell'assembler e cioè di quei comandi che si forniscono all'assembler e che verranno da quest'ultimo eseguiti: sappiamo già per averlo visto in alcuni esempi e per averne già accennato, che le direttive sono una parte integrante e fondamentale di un programma scritto in assembler 8086/8088 e senza alcune di esse non potrebbe essere generata nemmeno una linea di codice in linguaggio macchina.

Iniziamo da quelle direttive relative alla segmentazione ed alle possibilità di indirizzamento e di allocazione dei programmi nella memoria; si tratta delle direttive

SEGMENT ... ENDS
ORG
PROC ... ENDP
GROUP
LABEL
ASSUME

In questa puntata ci occuperemo delle direttive SEGMENT ed ORG: cominciamo dunque dalla prima.

La direttiva «SEGMENT ... ENDS» (l'«Alignment»)

Abbiamo visto proprio nelle primissime puntate che la segmentazione è una caratteristica fondamentale della programmazione in assembler 8086/88: non ritorniamo ovviamente sui concetti già ampiamente espressi, anche se per forza di cose alcuni verranno in parte ripetuti.

Ricordiamo dunque in brevi parole che un programma in assembler 8086/88 è formato dall'insieme di un certo numero di unità logiche chiamate «segment», in generale distinti tra segmenti di codice e segmenti di dati, secondo una regola che è più che altro formale: è frequente infatti vedere segmenti di codice contenenti dei dati e viceversa. La scelta spetta ovviamente al programmatore ed alla situazione contingente, per la quale si deciderà qual è la strategia migliore.

Comunque sappiamo già che un «segment» può raggiungere un'estensione massima di 64kbyte ed inoltre deve avere un certo nome, che ci permetterà di riferirsi ad esso ed agli elementi che lo compongono.

Quello che ora andiamo ad analizzare sono ulteriori informazioni relative al segment, informazioni che appunto devono essere fornite nella direttiva SEGMENT, che ha la sintassi:

```
name SEGMENT [alignment] [combinability] ['class']
```

Mentre per il «name» non c'è nulla da aggiungere a quanto sappiamo, andiamo ad analizzare le tre opzioni, che possono anche non essere presenti e che infatti abbiamo indicate tutte e tre tra «parentesi quadre».

L'opzione «alignment» (letteralmente «allineamento») serve a specificare, laddove sia espressamente necessario, in che modo appunto il segment deve essere allineato in memoria, all'atto della sua allocazione vera e propria: supponiamo, per comprendere meglio di cosa stiamo parlando, di avere due segment qualunque e di lunghezza qualsiasi.

Ora questi segmenti si troveranno in memoria presumibilmente l'uno di seguito all'altro e dato che la lunghezza del primo può essere qualunque, si può fare in modo che il secondo segmento inizi non all'indirizzo successivo libero, ma ad esempio a partire da un certo indirizzo non del tutto specificato, ma sicuramente posto dopo l'ultimo indirizzo occupato dal primo segmento.

Le possibilità in tal senso possono essere di cinque tipi e rispettivamente:

PARA, BYTE, WORD, PAGE ed INPACE

Il primo (PARA) è quello di default e cioè viene assunto automaticamente nel caso che nella direttiva non compaia un esplicito «alignment»; secondo questo tipo di allineamento, il segmento comincerà ad un indirizzo posto all'inizio di un «paragrafo», che già sappiamo essere costituito da 16 byte.

Ecco che perciò un segmento di allineamento «PARA» inizierà ad un indirizzo fisico avente l'ultima cifra (parleremo sempre di valori esadecimali) uguale a 0.

Ad esempio un segmento può iniziare all'indirizzo fisico 34560H, a partire cioè dal «paragrafo 3456H»; tanto per rinfrescarci le idee, la prima locazione del nostro segmento avrà quest'ultimo valore come «segment» e come offset». La decima locazione? Essa avrà un offset di 000AH (appunto pari a 10) ed un segment ancora pari a 3456H: il suo indirizzo fisico, assoluto, se ben ci ricordiamo, è dato dalla somma del «segment» moltiplicato per 16 (o più semplicemente al q...e aggiungiamo uno 0 a destra) e del-

l'«offset», che perciò nel caso nostro vale 3456AH.

E l'ultima locazione del segmento, cioè la 64k-esima, che indirizzo avrà? Presto detto: il suo offset sarà pari ad OFFFFH ed il suo segment sarà sempre 3456H, il che dà un indirizzo fisico di 34560H + 0FFFFH = 4455FH.

Ecco che perciò il nostro segmento si estenderà al massimo tra gli indirizzi 34560H e 4455FH compresi: ovviamente potrà essere anche più piccolo ed anzi in genere sarà così; basta ricordarsi che non potrà mai eccedere i 64k. Per quanto riguarda i tipi di allineamento «byte» e «word», il ragionamento è più semplice: nel caso di «byte» il nostro segmento potrà iniziare a qualsiasi indirizzo fisico della memoria, senza alcuna restrizione, mentre nel caso «word» il segmento inizierà ad una cella di memoria avente indirizzo pari.

Vediamone dunque due esempi, dapprima per il caso BYTE.

Supponiamo che il nostro segment inizi all'indirizzo 12345H (perché così ha voluto il caso...): la prima locazione del segmento, dal momento che questo possiede per nostra scelta un alignment di tipo «BYTE», avrà come «segment» il valore 1234H e come «offset» il valore 0005H.

È questo un caso interessante ed allo stesso tempo anomalo: di solito siamo stati abituati a vedere un segmento che iniziava con un offset nullo, mentre nel caso di allineamento di tipo byte è facile vedere che l'offset iniziale può variare tra 0000H e 000FH, dove quest'ultimo è proprio il caso più sfavorevole, in cui sarebbe stato iniziare al byte successivo, per avere così un segmento allineto al «paragrafo». Comunque non c'è da preoccuparsi, dal momento che sarà solo l'assemblatore e più tardi il «locater» a tenere conto della situazione: come dire che ancora una volta il tutto è trasparente (e perciò invisibile) per il programmatore, che sa soltanto di aver scelto uno dei cinque possibili tipi di allineamento.

Proseguendo oltre, il tipo «word» prevede che l'indirizzo di partenza del segment sia pari: ad esempio un segment può iniziare all'indirizzo 0CCCCCH.

In questo caso la prima locazione del segment avrà un «segment» pari a 0CCCCH ed un «offset» pari a 000CH: anche stavolta cioè il valore iniziale dell'offset può non essere nullo e anzi può valere 0, 2, 4, 6, 8, 0AH, 0CH e 0EH.

Per quanto riguarda l'«alignment» di tipo «PAGE» diciamo che richiede l'inizio di un segmento ad un indirizzo multiplo di 256 (100H), avente perciò due zeri come cifre meno significative: ad esempio un segment inizierà all'indirizzo fisico 03F400H per cui la prima locazione di memoria avrà un valore del «segment» pari a 3F40H ed un «offset» nullo.

Per quanto riguarda l'ultimo tipo di allineamento, «INPAGE», si può dire che si tratta di un caso particolare raramente utilizzato in quanto richiede che il segmento risieda completamente all'interno di una «pagina»: dato che quest'ultima è ampia 256 (100H) byte, ecco che anche il segmento in questione deve essere di lunghezza minore o uguale a 256 byte, il che è effettivamente una limitazione, a meno di non richiederlo appunto specificatamente in quanto la situazione lo richiede.

Un esempio di quest'ultimo tipo può essere una tabella di 50 valori che desideriamo contenere all'interno di una pagina, non importando a questo punto l'indirizzo iniziale: in tal modo ognuno dei 50 byte della tabella avrà un valore fisso del «segment» (per la precisione terminante per «0»), mentre l'«offset» assumerà 50 valori consecutivi.

Ad esempio supponendo che l'indirizzo iniziale del segmento sia 876A4H, ecco che il primo byte della tabela avrà come «segment» il valore 8760H ed un offset di 00A4H: l'ultimo elemento della tabella, di indirizzo fisico 87724H avrà ancora come «segment» il valore 8760H (e non 8772H, né tantomeno 8770H) e come «offset» il valore 0124H.

Anche se quest'ultimo esempio può sembrare alquanto cervellotico, in questo caso siamo sicuri che il valore del «segment» rimane in tutti i casi costante, però con la restrizione che il segment non deve avere più di 256 byte.

La direttiva «SEGMENT ... ENDS» (la «Combinability»)

L'opzione «Combinability» di un segmento rappresenta la «combinabilità» di un certo segmento nei riguardi degli altri segmenti facenti parte del nostro programma e di altri moduli e dunque definisce in quale modo il segmento è connesso con gli altri: la «combinability» può assumere sei differenti tipi

(niente), PUBLIC, COMMON, AT expression, STACK e MEMORY

Il primo è quello di default e si ottiene non indicando alcun tipo di combinabilità: si riferisce ad un segmento a se stante appartenente ad un unico modulo o programma.

Una combinabilità di tipo PUBLIC indica invece che tutti i segmenti aventi tale attributo ed aventi lo stesso nome (ma ovviamente appartenenti a moduli differenti), al momento del «linking», verranno sistemati contiguamente e non cioè separati da altri segmenti.

Il tipo COMMON è invece un poco più complesso e richiede che più segmenti aventi lo stesso nome condividano la medesima zona di memoria a tal punto che una stessa locazione fisica di memoria potrà essere indirizzata tramite etichette o nomi di variabili completamente differenti a seconda del modulo «chiamante»: è questo un caso molto particolare, usato parecchie volte, e che come tale richiede grandissima attenzione.

Un esempio banale di un segmento di appena tre byte può essere il seguente:

```
ESEMPIO SEGMENT COMMON
ALFA DB?
BETA DW?
ESEMPIO ENDS
```

dove appunto il primo byte è individuato dalla variabile ALFA mentre la successiva coppia di byte individua la variabile BETA di tipo «word».

Tutto questo in un certo modulo, ovverossia in un certo programma. Ora supponiamo che in un altro modulo, che poi andrà linkato con il pre-

cedente, compaia un segmento siffatto

```
ESEMPIO SEGMENT COMMON
GAMMA DW?
DELTA DB?
ESEMPIO ENDS
```

Ecco che ora i primi due byte sono individuati da una word chiamata GAMMA, mentre il terzo byte è individuato dalla variabile DELTA.

Succede dunque che i tre byte summenzionati possono essere chiamati in maniera completamente differente nell'ambito dei due moduli, mentre viceversa sono fisicamente gli stessi: nel nostro esempio vediamo che GAMMA ha come byte meno significativo ALFA e come byte più significativo la «parte bassa» di BETA. È facile dedurre anche che DELTA corrisponde in tutto e per tutto con la parte più significativa di BETA, con tutte le conseguenze del caso e trucchetti che se ne possono trarre.

Il quarto tipo di combinabilità (AT <expression>), che già abbiamo incontrato, prescrive che il segmento sia posto ad un indirizzo tale da avere come «segment» il valore dato da «expression» ed «offset» pari a 0.

Ad esempio se scriviamo

```
SEGMENTO SEGMENT AT 3333H
```

ciò significa che noi vogliamo che la sua prima locazione di memoria sia posta all'indirizzo fisico 33330H.

Il quinto tipo di combinability (STACK), richiede che tutti i segmenti di tale tipo di combinabilità ed aventi lo stesso nome siano posti in un unico segmento, in overlay e perciò non in segmenti allocati consecutivamente: come si vede si tratta di una gestione ancora completamente differente perciò ne vediamo subito un esempio.

Supponiamo di avere in un modulo un segmento (che necessariamente coinvolgerà uno stack) definito così:

```
STACK_SEGMENT     SEGMENT STACK
                  DW 100 DUP (?)
STACK_TOP         LABEL WORD
STACK_SEGMENT     ENDS
```

e cioè da cento word non inizializzate (grazie all'uso della DUP con il «?») ed in cui l'elemento «affiorante» dello stack ha come etichetta STACK_TOP.

In un altro modulo invece supponiamo di avere il seguente segmento relativo allo stack:

```
STACK_SEGMENT     SEGMENT STACK
                  DW 30 DUP (?)
TOP_OF_STACK      LABEL WORD
STACK_SEGMENT     ENDS
```

dove ora lo stack è previsto di solo trenta byte e dove in questo caso l'elemento affiorante si chiama TOP_OF_STACK.

Perciò in questo caso si otterrà un unico segmento in overlay «verso indirizzi alti» di lunghezza pari a 130 word: in generale sarà di lunghezza pari proprio alla somma delle singole lunghezze, ma in ogni caso si considererà coincidente l'indirizzo dell'elemento affiorante.

Ecco che perciò sia TOP_OF_STACK che STACK_TOP avranno lo stesso «offset» nell'ambito del segmento ottenuto ed in ogni caso rappresenteranno simbolicamente l'ultima word dello stack segment: non dimentichiamoci infatti che uno stack si sviluppa per indirizzi decrescenti.

L'ultimo tipo di combinabilità di un segmento è, come detto, «MEMORY» e si riferisce ad un insieme di segmenti che condividono la stessa zona di memoria, analogamente al caso del «COMMON», ma in più il segmento ottenuto verrà posto al di sopra di tutti gli altri segmenti generati.

La direttiva «SEGMENT ... ENDS» (la «Class»)

Quest'ultimo attributo di un segmento non è altro che un nome qualsiasi (di lunghezza massima pari a 40 caratteri e posto tra «apici») che possiamo attribuire ad un certo segmento: quello che si ottiene è che segmenti di caratteristiche completamente differenti ed appartenenti per giunta a moduli diversi, ma aventi «class» uguale, verranno posti in memoria insieme, l'uno di seguito all'altro.

È buona norma classificare i segmenti di più moduli da linkare poi insieme, per mezzo di nomi tipo 'CODE' per tutti i segmenti di codice (aventi allineamento e/o combinabilità differenti), 'DATA' per tutti i segmenti di dati e 'STACK' per tutti i segmenti coinvolgenti uno stack: dato che i nomi possono essere qualsiasi, nulla impedisce di usare parole riservate quali appunto CODE, DATA e STACK, come pure usare nomi completamente inventati.

La direttiva ORG

Questa direttiva, che già era presente negli assembler di microprocessori ad 8 bit quali l'8080 e lo Z80, ha il compito di settare un valore diverso da quello di default per l'«offset» all'interno di un certo segmento.

Abbiamo visto nei paragrafi precedenti che l'offset iniziale della prima locazione di memoria appartenente ad un certo segmento dipende fortemente dall'alignment prescelto per quel segmento: ma dato che l'allineamento di default e più usato è quello PARA, allora non è limitativo dire che in generale l'offset iniziale di un segmento è sempre zero.

La direttiva ORG comunque dà la possibilità al programmatore di fissare l'offset delle locazioni a partire da quella successiva alla direttiva e fino ad un'altra eventuale direttiva ORG.

La direttiva in esame ha una sintassi molto semplice:

```
ORG <expression>
```

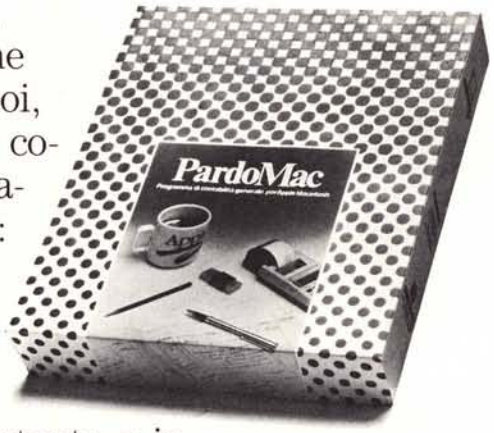
dove in <expression> indichiamo appunto il valore da assegnare all'offset: tale valore può appunto provenire dal calcolo di un'espressione contenente anche simboli, che però devono già essere stati definiti, ma si otterrà comunque un valore modulo 64k e cioè comunque compreso tra 0 e 65535 (tra 0000H e 0FFFFH).

Generalmente nella maggior parte dei programmi la direttiva ORG non è necessaria, ma comparirà solo in particolari applicazioni in cui si deve prestabilire l'indirizzo esatto di una certa locazione: comunque il numero di tali direttive in un programma può essere qualsiasi e non è nemmeno necessario che gli indirizzi forniti con <expression> siano di valore crescente.

Per quanto riguarda quest'ultima affermazione però si può correre il rischio di creare più blocchi di dati posti allo stesso indirizzo ed in tali casi si otterrebbe una non desiderata sovrapposizione, che vedrebbe come «vincitore» l'ultimo dei blocchi incriminati definti con le ORG, come dire che ogni volta l'assembler si fida dell'ultima ORG fornita. MC

Adesso facciamo i conti

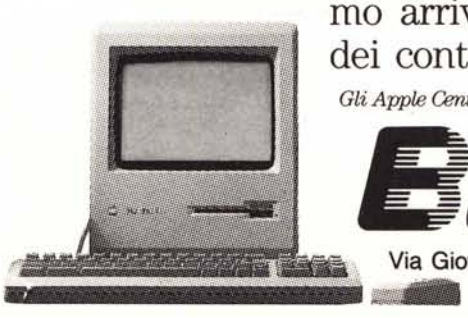
Cara la nostra partita doppia, finalmente hai trovato pane per i tuoi denti: anche per noi, «poveri mortali», che non conosciamo bene l'informatica, c'è una nuova arma: PardoMac.



PardoMac contro la mancata quadratura dei bilanci, un sistema semplice, potente e incredibilmente versatile. Facile da usare anche senza l'aiuto di un «Einstein». Potente e veloce perché sfrutta la tecnologia di Macintosh (es. 2000 nominativi di clienti possono essere ordinati in meno di 4 secondi). Versatile perché permette la gestione contemporanea di più finestre sullo schermo. Finalmente qualcuno ci viene in aiuto.

Easy-byte, con PardoMac ha risolto tutti i problemi fiscali e gestionali della contabilità. Siamo arrivati alla resa dei conti.

Gli Apple Center vi stanno già aspettando.



PardoMac®
è un programma di contabilità generale per Apple Macintosh™
Realizzato da Easy Byte, Sydne Co.

Macintosh è un marchio di McIntosh Laboratory Inc. ed è usato su sua licenza.

easy·byte s.r.l.
Via Giovanni Villani, 24 - 00179 Roma
Tel. 7811519-7887926