ASSEMBLER 8086 8088

di Pierluigi Panunzi

Le direttive dell'Assembler (terza parte)

In questa puntata proseguiamo ancora lo studio delle direttive dell'Assembler 8086/8088, delle quali ci sono rimaste da analizzare le ultime due: le direttive «GROUP» e «ASSUME».

La seconda, come vedremo in particolare nella prossima puntata, è di fondamentale importanza in quanto serve a comunicare all'assemblatore stesso delle informazioni sulle quali poi verranno generate in una maniera oppure in un'altra praticamente tutte le istruzioni del nostro programma in Assembler: la mancanza di tale direttiva, come è facile constatare in banali esempi, comporta un numero notevole di errori dal momento che, come vedremo, l'Assembler non è in grado di decidere da solo sul come tradurre certe istruzioni.

La direttiva «GROUP»

Si tratta di una direttiva utilizzata principalmente in quei casi in cui o il nostro programma, oppure i dati su cui il programma lavora, oppure entrambi gli insiemi, non sono formati da un unico segmento (di codice e di dati), ma viceversa sono costituiti da un certo numero di segmenti che soltanto nella loro globalità costituiscono rispettivamente il codice da eseguire e l'insieme dei dati su cui opera il programma stesso.

È questo il caso ad esempio di un unico programma scritto però da più persone, le quali ognuna ha definito i propri segmenti di codice e di dati, senza magari aver idea di quanto avesse fatto un altro programmatore: dovendo al termine far quadrare i conti, dovremo far sì che tutti i segmenti di codice vengano raggruppati in un unico segmento logico (all'interno del quale rimane ben netta la distinzione tra un segmento e l'altro), così come dovremo agire nei riguardi dei segmenti dati.

Oltre dunque alla riunione di più segmenti in un unico segmento logico, la direttiva in esame consente di ottenere come «side-effects» (gli «effetti collaterali») anche delle migliorie nel programma visto nella sua globalità.

Il fatto di avere un certo numero di segmenti di dati comporta innanzitutto a livello di assemblaggio notevoli problemi: basta ricordare che istante per istante il Date Segment (DS) «punta», fa riferimento, ad uno solo dei segmenti di dati e perciò consente all'assemblatore di raggiungere solo i dati e le variabili contenuti in quel segmento.

Vediamo subito nel listato 1 un esempio di quanto appena detto con un piccolo programma di prova, assemblato con il «MASM.EXE» su di un PC compatibile.

In questo esempio vediamo infatti che l'aver definito come segmento dati corrente il segmento «DATA 1» comporta poi che le variabili GAMMA e DELTA non possono essere raggiunte a meno di non cambiar volta per volta il valore del DS, sia a livello assemblaggio sia poi a livello microprocessore, cosa che non deporrà certo a favore del programmatore...

Non dimentichiamoci mai, infatti che, se non si sta più che attenti, in alcuni casi quanto si dice all'Assembler tramite le direttive sarà poi interpretato in maniera completamente differente dal microprocessore: prova ne è l'esempio riportato nella scorsa puntata, oggetto di un mini-quiz. Viceversa con la direttiva «GROUP» si comunica all'Assembler quali segmenti fanno parte di un unico segmento logico, un «gruppo» per l'appunto.

Nell'esempio del listato 2 vediamo come l'assemblatore non segnali più l'errore, in quanto ora al DS si associa non più un unico segmento, ma un insieme di segmenti, all'interno del quale può raggiungere tutte le variabili.

Analizziamo per un istante il listing ottenuto assemblando il programma corretto: innanzitutto notiamo la creazione del gruppo chiamato «DATA», al quale poi l'Assembler farà riferimento con la particolare notazione «---- R».

I trattini, lo ricordiamo, rappresentano il fatto che l'Assembler, a questo livello, non sa effettivamente a quale indirizzo verranno allocati i segmenti formanti il gruppo «DATA», mentre la «R», riportata poi di seguito in corrispondenza delle variabili appartenenti a «DATA», ricorda al programmatore che l'offset indicato nell'istruzione è «rilocabile» e come tale verrà quasi sicuramente modificato a seconda dell'effettiva allocazione in memoria nei vari segmenti dati.

Per inciso ricordiamo ancora che i trattini simboleggiano non già un offset (come nel caso delle variabili) ma bensì una base di un segmento: mentre perciò per un segmento o gruppo che sia l'assemblatore non ha idea di dove verrà poi allocato in memoria il segmento, viceversa per le variabili

```
                                NAME PROVA
0000                    DATA1   SEGMENT

0000  ??                ALFA    DB ?
0001  ??                BETA    DB ?

0002                    DATA1   ENDS

0000                    DATA2   SEGMENT

0000  ??                GAMMA   DB ?
0001  ??                DELTA   DB ?

0002                    DATA2   ENDS

0000                    CODE    SEGMENT

                                ASSUME CS:CODE,DS:DATA1
0000  B8  ---- R                MOV AX,DATA1
0003  8E D8                    MOV DS,AX
                        ;
0005  A0 0000 R                 MOV AL,ALFA
                        ;
0008  A0 0001 R                 MOV AL,BETA
                        ;
0008  A0 0000 R                 MOV AL,GAMMA
E r r o r   ---     68:Can't reach with segment reg
                        ;
000E  A0 0001 R                 MOV AL,DELTA
E r r o r   ---     68:Can't reach with segment reg
                        ;
0011                    CODE    ENDS
```

Listato 1

```
                                NAME PROVA
0000                    DATA1   SEGMENT

0000  ??                ALFA    DB ?
0001  ??                BETA    DB ?

0002                    DATA1   ENDS

0000                    DATA2   SEGMENT

0000  ??                GAMMA   DB ?
0001  ??                DELTA   DB ?

0002                    DATA2   ENDS

                        DATA    GROUP DATA1,DATA2

0000                    CODE    SEGMENT

                                ASSUME CS:CODE,DS:DATA
0000  B8  ---- R                MOV AX,DATA
0003  8E D8                    MOV DS,AX
                        ;
0005  A0 0000 R                 MOV AL,ALFA
                        ;
0008  A0 0001 R                 MOV AL,BETA
                        ;
0009  A0 0000 R                 MOV AL,GAMMA
                        ;
000E  A0 0001 R                 MOV AL,DELTA
                        ;
0011                    CODE    ENDS

                                END
```

Listato 2

```
                                NAME PROVA
0000                    DATA    SEGMENT

0000  ??                ALFA    DB ?
0001  ??                BETA    DB ?

0002                    DATA    ENDS

0000                    CODE1   SEGMENT

                                ASSUME CS:CODE1,DS:DATA
0000  B8  ---- R                MOV AX,DATA
0003  8E D8                    MOV DS,AX
                        ;
0005  EB 01 90                  JMP LABEL1
                        ;
0008  EA 0000  ---- R   LABEL1: JMP FAR PTR LABEL2
                        ;
000D                    CODE1   ENDS

0000                    CODE2   SEGMENT

                                ASSUME CS:CODE2,DS:DATA

0000  90                LABEL2: NOP
                        ;
0001  A0 0001 R                 MOV AL,BETA
                        ;
0004                    CODE2   ENDS

                                END
```

Listato 3

```
                                NAME PROVA
0000                    DATA    SEGMENT

0000  ??                ALFA    DB ?
0001  ??                BETA    DB ?

0002                    DATA    ENDS

                        CODE    GROUP CODE1,CODE2

0000                    CODE1   SEGMENT

                                ASSUME CS:CODE,DS:DATA
0000  B8  ---- R                MOV AX,DATA
0003  8E D8                    MOV DS,AX
                        ;
0005  EB 01 90                  JMP LABEL1
                        ;
0008  EA 0000  ---- R   LABEL1: JMP FAR PTR LABEL2
                        ;
000D                    CODE1   ENDS

0000                    CODE2   SEGMENT

                                ASSUME DS:DATA

0000  90                LABEL2: NOP
                        ;
0001  A0 0001 R                 MOV AL,BETA
                        ;
0004                    CODE2   ENDS

                                END
```

Listato 4

```
                        NAME PROVA
0000                    DATA    SEGMENT
0000  ??                ALFA    DB ?
0001  ??                BETA    DB ?
0002                    DATA    ENDS
                        CODE    GROUP CODE1,CODE2
0000                    CODE1   SEGMENT
                                ASSUME CS:CODE,DS:DATA
0000  B8  ---- R                MOV AX,DATA
0003  8E D8                    MOV DS,AX
                        ;
0005  EB 01 90                  JMP LABEL1
                        ;
0008  E9 0000 R         LABEL1: JMP LABEL2
                        ;
0008                    CODE1   ENDS
0000                    CODE2   SEGMENT
                                ASSUME DS:DATA
0000  90                LABEL2: NOP
                        ;
0001  A0 0001 R                 MOV AL,BETA
                        ;
0004                    CODE2   ENDS
                                END
```

Listato 5

```
                        NAME PROVA
0000                    DATA    SEGMENT
0000  ??                ALFA    DB ?
0001  ??                BETA    DB ?
0002                    DATA    ENDS
                        CODE    GROUP CODE1,CODE2
0000                    CODE1   SEGMENT
                                ASSUME CS:CODE,DS:DATA
0000  B8  ---- R                MOV AX,DATA
0003  8E D8                    MOV DS,AX
                        ;
0005  EB 01 90                  JMP LABEL1
                        ;
0008  E9 0005 R         LABEL1: JMP LABEL2
                        ;
000B  8A 1E 0001 R              MOV BL,BETA
000F  2A 0E 0000 R              SUB CL,ALFA
                        ;
0013                    CODE1   ENDS
0000                    CODE2   SEGMENT
                                ASSUME DS:DATA
0000  90                        NOP
0001  8B C3                     MOV AX,BX
0003  2B CE                     SUB CX,SI
                        ;
0005  90                LABEL2: NOP
                        ;
0006  A0 0001 R                 MOV AL,BETA
                        ;
0009                    CODE2   ENDS
                                END
```

Listato 6

l'Assembler fornisce appunto l'offset della singola variabile all'interno del proprio segmento di appartenenza, cosa che si può facilmente vedere nella parte iniziale del programma, laddove vengono definiti «DATA1» e «DATA2» e le variabili.

Come abbiamo già detto altre volte, infatti, il programma «finale» e cioè il codice oggetto che verrà poi eseguito dal microprocessore conterrà, in corrispondenza alle variabili ed ai segmenti, degli indirizzi effettivi strettamente legati all'allocazione del programma in memoria.

Altra possibilità che ci consente la direttiva «GROUP» è quella di indicare nella lista dei segmenti appartenenti al gruppo stesso, non solo i nomi effettivi dei segmenti, ma anche un comando del tipo:

SEG nome-della-variabile

per mezzo del quale inseriamo nel gruppo il segmento (del quale non conosciamo il nome) al quale fa parte la variabile indicata: ciò e molto ultile oltreché indispensabile nel caso di variabili di tipo «external» e cioè non effettivamente definite nel programma in esame, ma viceversa definite in un loro segmento, ma in un altro modulo.

Nel caso in cui la direttiva «GROUP» è applicata ad un insieme di segmenti di codice, oltre alla solita unificazione logica dei segmenti stessi, si ottiene anche un codice in generale più compatto, soprattutto per quello che riguarda i salti e le chiamate a subroutine.

Infatti nel caso generale in cui dall'interno di un certo segmento di codice si salti ad un'altra parte di programma appartenente ad un altro segmento, sappiamo già che il codice generato terrà conto del cambiamento di segmento, inserendo l'indirizzo completo dell'etichetta da raggiungere, sotto forma di segment+offset: questo fatto comporta l'utilizzazione di 2 word per quello che viene chiamato in gergo un «salto lungo», contrapposto a quello «corto» (all'interno dello stesso segmento) o a quello «ultra-corto» (sempre all'interno dello stesso segmento, ma con spostamenti in più o in meno al massimo di 128 byte), nei quali casi viene utilizzata rispettivamente una word e un byte.

Vediamo subito nel listato 3 un esempio di programma in cui il codice è spezzato in due segmenti, senza usare la direttiva «GROUP».

In questo caso appunto siamo stati costretti ad usare l'istruzione «JMP FAR PTR» per poter saltare all'etichetta «LABEL2» del secondo segmento, ottenendo perciò un'esplosione in codice formata da un byte (l'opcode della «JMP FAR PTR»), seguito da una word (l'offset di «LABEL2») e da un'altra word (il segmento di «LABEL2», che è stato posto al solito pari a «----» in quanto incognito).

Sfruttando invece la direttiva «GROUP», senza però cambiare ancora l'istruzione «JMP FAR PTR LABEL2», si ottiene il listato 4 in cui praticamente non è cambiato nulla, se non lo snellimento della direttiva ASSUME, che ora fa riferimento al gruppo «CODE».

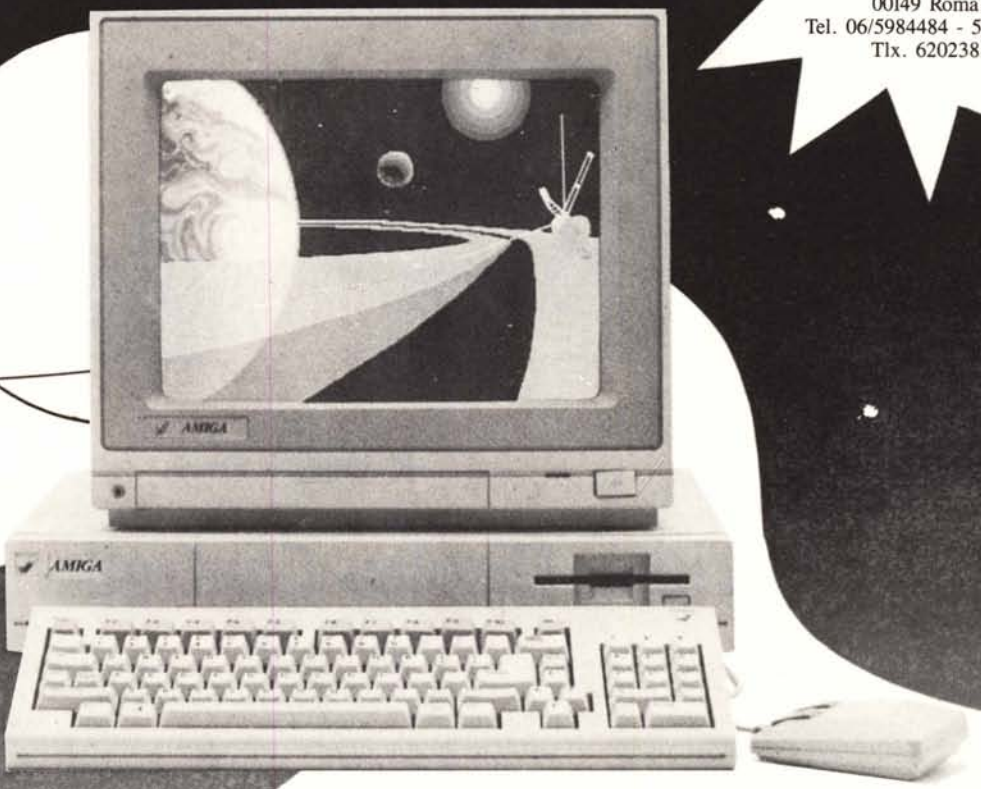
Invece utilizzando l'istruzione «JMP» (di tipo «near»), otteniamo il listato 5 in cui l'istruzione «JMP» contiene soltanto lo «spostamento» in termini di byte a partire dall'istruzione chiamante. In realtà l'esempio riportato può trarre in inganno: il valore «0000 R» posto in corrispondenza dell'istruzione in esame, non significa che l'etichetta in esame si trova a «0» byte di distanza dall'istruzione stessa, ma viceversa rappresenta l'offset di «LABEL2»: in particolare basta aggiungere qualche istruzione per alterare la distanza tra la «JMP» e l'etichetta ed altre istruzioni per variare l'offset dell'etichetta stessa (notare che sta in un altro segmento...) per veder variare il valore «0000» (listato 6).

In entrambi i casi comunque quella che comanda è quella lettera «R» che ancora una volta indica la rilocabilità del programma ed il fatto che poi in realtà al posto del valore «0000» o «0005» troveremo l'effettivo spostamento ottenuto in base a dove e come verrà caricato il programma in memoria.

MC

PROGRAMMA IL FUTURO

DISCOM s.r.l.
Via Padre G.A. Filippini, 119
00149 Roma
Tel. 06/5984484 - 5984857
Tlx. 620238



AMIGA

PUNTI VENDITA SPECIALIZZATI

CARET s.r.l. - Roma - Tel. 06/3285349
CHOPIN INFORMATICA - Roma - Tel. 06/5916462
COMPUSHOP - Roma - Tel. 06/857124
COMPUTEL - Roma - Tel. O6/5816673
COMPUTER LINE - Roma - Tel. 06/384907
COMPUTER MARKET - Roma - Tel. 06/7945493
DATARING - Nettuno - Tel. 06/9806758
DATA SERVICE ITALIA - Roma - Tel. 06/5916438
DATA SOFT - Latina - Tel. 0773/486110
DELTA BIT - Albano - Tel. 06/9304664
ELETTRONICA 2003 - Roma - Tel. 06/5110366
EUROFOTO- Terracina - Tel. 0773/727175
FIORINI 82 - Roma - Tel. 06/4270445
FOTOFLASH - Roma - Tel. 06/779046
IACOPONI - Roma - Tel. 06/867706
METRO IMPORT - Roma - Tel. 06/3607600
MRS - Frascati - Tel. 06/9426684
PIX COMPUTER SERVICES - Roma - Tel. 06/8384184
COMPUTER CENTER - Roma - Tel. 06/7591544

ULTIMISSIME HARDWARE

Hard disk 10-20-40 Mbyte esterni completi di controller, cavi ed interfaccia SCSI
Tape Backup Streaming capacità 20 Mbyte

Sidecar-Espansione hardware per rendere Amiga MS-DOS compatibile 100%. CPU 8088, Drive $5^1/_4$, 3 slots fullsizeliberi, 256KPCRAM espandibile 640 K, CLOCK 4.77 MHZ

Espansioni di memoria esterne da 512 K - 2 Mbyte
Digitalizzatore video Digiview
Modem 75-300-1200 half duplex completo di cavi

PER ULTERIORI INFORMAZIONI:
DISCOM- Via Padre G.A. Filippini 119 - 00149 Roma

NOME .....
COGNOME .....
INDIRIZZO .....
CITTÀ .....