ASSEMBLER 8086 8088

di Pierluigi Panunzi

Costanti e variabili

In questa puntata inizieremo ad affrontare l'argomento dell'«Assembler» non più dal punto di vista astratto dato dalle sue caratteristiche legate ai microprocessori 8086 o 8088, ma cominceremo a parlarne come di uno specifico programma (chiamato per la precisione ASM86) che ha il compito di leggere un certo file (sorgente), fare le sue debite elucubrazioni, e generare in uscita un file (oggetto) che utilizzeremo in seguito come sorgente di un altro programa, il «Linker».

In particolare, dal momento che programmi assemblatori per 86/88 fanno tutti riferimento, seppur con piccole modifiche, al capostipite creato dall'Intel, ecco che parlando di «Linguaggio Assemblativo per 86/88» si intende implicitamente riferirsi all'ASM86 «confondendo» quelle che sono le regole sintattiche-grammaticali dell'uno con quelle dell'altro.

Diciamo questo, anche se in prima analisi potrà sembrare ovvio, in quanto abbiamo a che fare con un linguaggio strettamente legato ad un certo microprocessore e non ad esempio con un linguaggio ad alto livello quale il Pascal o (peggio) il Basic, dei quali esistono implementazioni più o meno differenti a seconda dell'«ambiente» nel quale essi operano: i lettori ben sanno che esistono alcune istruzioni che hanno un significato per un certo Basic ed uno completamente differente in un altro dialetto.

Ecco che perciò siamo sicuri che un'istruzione definita dall'Intel secondo certe regole rimarrà tale qualsiasi sia l'«ambiente» (in particolare il Sistema Operativo) in cui coesiste: le eventuali differenze potranno arrivare da nuove direttive aggiunte per particolari scopi, ma la sostanza rimarrà la stessa.

Tutto questo è stato scritto per la buona pace dei neofiti e cioè di chi si avvicina per la prima volta a questo strano mondo del linguaggio macchina: non potevamo certo dire subito che con la direttiva CODEMACRO uno si può creare un set di istruzioni a proprio uso e consumo...

OOPS, l'abbiamo detto!...

Gli elementi costitutivi di un programma Assembler 86/88.

Con il termine «elementi costitutivi» di un programma intendiamo in particolare gli elementi «sintattici» che ci consentono, secondo certe regole grammaticali molto semplici, di scrivere un programma in Assembler, sotto forma di uno o più moduli logici, costituiti da un insieme di direttive e di istruzioni, a loro volta formate da identificatori ed altri elementi sintattici.

In effetti non c'è (e non ci vuole essere) nulla di nuovo rispetto ad un qualsiasi altro assemblatore: è chiaro però che le differenze si faranno ben notare laddove risulterà differente la «filosofia progettuale» di un certo microprocessore nei confronti di un altro.

Tanto per cominciare, parleremo in questa puntata di quelli che genericamente abbiamo indicato come «elementi sintattici» ed «identificatori», ovvero gli ingredienti per mezzo dei quali si può scrivere una qualsiasi istruzione in Assembler.

Le costanti numeriche e le stringhe

Con il termine generico di «costante» si indica, come è ben noto, una quantità il cui valore non viene alterato dal programma in cui essa risiede, ma bensì è ben noto a qualsiasi livello ed in qualunque istante, dall'«assembly time» al «run time», cioè dall'istante in cui viene definita, all'istante in cui viene utilizzata dal programma che gira.

In particolare le costanti dell'Assembler 86/88 possono essere di due (soli) tipi e cioè di tipo intero e di tipo stringa di caratteri: le costanti intere possono essere espresse in binario, in ottale, in esadecimale nonché in decimale, per la gioia di qualsiasi programmatore, che potrà così esprimere le costanti secondo la base che più gli aggrada.

Le costanti binarie sono ovviamente costituite da una sequenza di 0 ed 1, però seguite dalla lettera «B» (oppure «b» indifferentemente): ad esempio abbiamo

```
0B
1B
100000001B
-101B
```

Le costanti ottali sono costituite da una sequenza di cifre comprese tra 0 e 7, seguite dalla lettera «O» o dalla lettera «Q» (oppure dalle rispettive minuscole): ad esempio abbiamo

```
1Q
11223344Q
-77Q
0O
-1O
```

Le costanti esadecimali sono invece formate da sequenze di cifre comprese tra 0 e 9 nonché le lettere da «A» ad «F» (maiuscole o minuscole indifferentemente, anche mischiate), seguite dalla lettera «H» (oppure «h») ed inoltre se iniziano per una delle lettere tra «A» ed «F» allora devono essere precedute dallo «0», in modo da iniziare una costante esadecimale sempre con una cifra: ad esempio si ha

```
0H
-1H
0FFFH
0BEACH
```

In particolare le ultime due costanti esadecimali rispondono alla regola che richiede come primo carattere una cifra, in quanto «FFFH» e «BEACH» (!) vengono viceversa interpretate come identificatori, come vedremo nel seguito.

Infine le costanti decimali sono le ben note sequenze di cifre comprese tra 0 e 9, seguite o meno dalla lettera «D» a seconda dei gusti: ad esempio abbiamo

```
0
100D
12345
```

In tutti i casi e cioè qualsiasi sia la base che abbiamo scelto, il valore espresso dalla costante deve essere rappresentabile con 17 bit e cioè con i 16 bit di una word più un bit aggiuntivo, indicante il segno positivo o negativo, altrimenti l'Assembler segnalerà errore per impossibilità di rappresentare correttamente una costante. Ciò comporta che i valori rappresentabili con una costante in una certa base possono essere compresi tra i limiti riportati nella tabella sottostante.

−1111111111111111B <=	costante binaria	<=1111111111111111B
−177777Q <=	costante ottale	<=177777Q
−0FFFFH <=	cost. esadecimale	<=0FFFFH
−65535 <=	costante decimale	<=65535

Vedremo poi in seguito che il fatto di poter rappresentare anche quantità negative ridurrà di un bit il massimo consentito.

Le stringhe di caratteri

Con tale termine si intendono le sequenze di caratteri ASCII e perciò stampabili, racchiuse da apostrofi (').

Oltre ai caratteri di cui sopra, che potevano essere lettere, numeri o simboli vari, si possono usare i blank, i TAB, mentre non si possono usare i carriage-return o i linefeed, che devono eventualmente essere impostati a parte.

L'Assembler rappresenterà la stringa di caratteri come una sequenza di byte ognuno contenente la codifica ASCII di ogni carattere, codifica che come noto avviene con 7 bit: ad esempio si ha

```
'MC MICROCOMPUTER'
'apostrofo = '''
'!@#$%^&*()_+-/*",.;«»|'
```

In particolare abbiamo nel secondo esempio l'inserimento di un apostrofo all'interno di una stringa, ottenuto semplicemente mettendone due di seguito, oltre ovviamente a quello iniziale e a quello finale: ecco che infatti volendo impostare la stringa «perché è così», dobbiamo scrivere:

```
'perché' è' così''
```

Viceversa le doppie virgolette ("), che non servono per racchiudere una stringa, possono essere poste all'interno di una stringa senza problemi: un esempio è l'ultimo dei tre visti precedentemente, mentre un altro, relativo ad un'altra stringa (... e disse "ciao"...), è il seguente: '...e disse: "ciao"...'

Un caso particolare di stringhe di caratteri è quello in cui la lunghezza della stringa è pari ad uno o a due, nei quali casi la stringa può direttamente identificarsi con un byte o una word corrispondente alla codifica ASCII del o dei caratteri: ad esempio si ha che le stringhe 'P' e 'MC' sono rispettivamente equivalenti ai valori esadecimali 50H e 4D43H e come tali vengono tradotte dall'assemblatore.

L'identità tra le stringhe di uno o due caratteri con i valori espressi in byte ed in word si ripercuote in particolare sul fatto che una stringa formata da un solo carattere può sostituire un valore espresso come byte dovunque può essere posto un valore immediato, esprimibile con un byte.

Ecco che un'istruzione che fa il confronto tra il contenuto del registro AL ed il valore 41H, codifica ASCII della lettera «A», può essere scritta sia

```
CMP AL41H che CMP AL,'A'
```

Analogo discorso, ma con delle piccole precauzioni, vale nel caso di costanti immediate a 16 bit e stringhe di due caratteri: anche in questo caso la sostituzione può essere effettuata senza problemi, se non quello di porre attenzione al fatto che una stringa viene posta in memoria un byte dopo l'altro, mentre un numero espresso in esadecimale con due byte viene allocato in memoria con il byte meno significativo (LSB) prima e l'MSB poi.

Ad esempio l'istruzione

```
MOV AX,4142H
```

può essere sostituita dall'istruzione

```
MOV AX,'AB'
```

in quanto solo in questo modo verrà posto in AL il valore 42H corrispondente alla lettera «B»; ciò avviene invece a differenza del caso di definizione di stringhe in memoria, nel qual caso, scrivendo ad esempio

```
ALFA DW 'AB'
```

otterremmo il valore 41H («A») nell'LSB e 42H («B») nell'MSB.

Gli identificatori

Con tale termine si intendono delle sequenze di caratteri aventi un significato simbolico per l'Assembler in quanto rappresentano appunto i simboli sui quali lavora l'Assembler.

Genericamente si può dire che gli identificatori sono le parole del linguaggio interpretato dall'Assembler ed in particolare sono l'insieme di tutte le parole-chiave (codici mnemonici, registri, parole riservate, ecc.), i simboli (le variabili e le etichette del programma), i numeri (le costanti nelle varie basi) ed altri elementi particolari del linguaggio (le direttive dell'assemblatore).

Il primo tipo di identificatori è dunque quello formato dalle parole-chiave (keyword) dell'Assembler, intendendo con tale termine sia i codici mnemonici delle varie istruzioni (ad esempio MOV, XLAT, STOSB, ecc.), sia i nomi ben noti dei registri su cui operano le singole istruzioni (ad esempio AX, BP, SI, SS, ecc.): in questo caso si tratta di simboli che il programmatore non può riassegnare, altrimenti l'Assembler, non sapendo più che pesci prendere, segnalerà errore.

Ad esempio non si può definire come nome di una variabile una parola-chiave tipo «AX» con un'istruzione del tipo

```
AX DW?
```

in quanto poi l'Assembler non saprebbe come interpretare un'istruzione del tipo

```
MOV AX,5
```

Passando dunque al secondo tipo di identificatori, quelli rappresentanti le variabili e le etichette di programma, diciamo che le sequenze di caratteri che li compongono devono sottostare a tre regolette, simili a quelle a cui siamo abituati con altri linguaggi, sia ad alto che a basso livello, nell'ambito della definizione di variabili ed etichette.

In particolare le variabili e le etichette possono avere una lunghezza qualsiasi, ma solo i primi 31 caratteri vengono considerati dall'Assembler; inoltre il primo carattere deve essere o alfabetico (da «A» a «Z», maiuscolo o minuscolo indifferentemente, anche mischiato) oppure uno dei caratteri speciali «@», «-» e «?», dove però il «?» non può esistere da solo come identificatore; l'ultima regola è che gli eventuali caratteri successivi possono essere sia i precedenti che le cifre da «0» a «9».

Ecco che perciò le seguenti sequenze di caratteri sono dei validi identificatori di variabili e/o etichette

```
PIPPO
BEACH
FFFFH
Variabile-numero-uno
Variabile-numero-settecentoventotto
```

```
aaaaAAAA
-1234567890123456789012345678901AAAAA
-1234567890123456789012345678901BBBBB
```

dove gli ultimi due identificatori sono considerati come lo stesso identificatore da parte dell'Assembler in quanto identici fino al trentunesimo carattere: è ovvio che in questo modo i nomi delle variabili possono diventare mnemonici al massimo.

Viceversa non sono validi identificatori di variabili e/o etichette le seguenti sequenze di caratteri

```
Valore-in-$
1234567890-AAAA
'variabile'
```

in quanto rispettivamente contengono un carattere «$» non ammesso per gli identificatori, iniziano con una cifra, che come sappiamo può essere posta solo come secondo carattere ed infine perché i caratteri racchiusi tra apostrofi sappiamo rappresentare una stringa alfanumerica.

Il terzo tipo di identificatore è rappresentato da quelle sequenze di caratteri che non si riferiscono ad una locazione di memoria (come le variabili o le etichette), ma vengono associate a delle quantità numeriche grazie alla direttiva EQU, che si trova praticamente in tutti gli assemblatori.

Ecco che ad esempio impostando la seguente istruzione

```
INDICE-MASSIMO EQU 100
```

associamo all'identificatore «INDICE-MASSIMO» il valore 100 e con tale valore verrà sostituito dall'Assembler ogni volta che viene incontrato: ad esempio l'istruzione

```
MOV BX, INDICE-MASSIMO
```

caricherà nel registro BX il valore decimale 100.

L'ultimo tipo di identificatori ammessi dall'Assembler sono le cosiddette direttive, le altre parole chiave «di attributo», nonché gli operatori.

Su questi particolarissimi identificatori ritorneremo in dettaglio nel corso delle varie puntate ed in particolare per molti di essi si dovrà parlare a lungo: alcuni di questi già li conosciamo: abbiamo già più volte parlato di SEGMENT, di EQU di WORD e BYTE nonché di PROC.

Comunque anche su questi torneremo presto a riparlare per definirne in dettaglio le caratteristiche e soprattutto confrontando quanto appena esposto con degli esempi pratici. La prossima puntata parleremo in dettaglio delle variabili e su come possono essere inizializzate ed inoltre vedremo quali sono le regolette, molto semplici, con le quali si scrivono le singole istruzioni di un nostro programma in Assembler.

MC

Commodore sono più grandi in tutto.

PIÙ GRANDI NELLA PROFESSIONALITÀ.

GRUPPO ETHOS

I grandi Commodore sono nati per affrontare agevolmente i compiti più gravosi.
Perfettamente compatibili lavorano efficacemente con tutti i programmi della vastissima biblioteca MS-DOS: dai wp ai data base, dai fogli elettronici agli integrati.
Ma nel campo dei personal, Commodore è impegnata a fondo, perciò ha realizzato il nuovo Commodore AT, attualmente il più potente tra i personal compatibili.
Qualsiasi necessità di elaborazione abbiate, la qualità dei personal Commodore è alla vostra portata.
A tutti i livelli.

Commodore