ASSEMBLER 8086/8088

di Pierluigi Panunzi

Le direttive dell'assembler

(seconda parte)

In questa puntata ci occuperemo in dettaglio delle direttive
LABEL
PROC ... ENDP
e cioé proseguiremo il discorso iniziato la scorsa puntata: iniziamo dunque dalla prima.

La direttiva LABEL: le etichette

In generale sappiamo che l'«etichetta» («label») è un nome associato ad una particolare locazione di memoria, che, a seconda delle scelte del programmatore, può contenere o un'istruzione oppure un dato: nel primo caso si parla di «etichetta» vera e propria di una certa istruzione che verrà utilizzata in istruzioni di salto o di chiamata a sobroutine (le procedure di cui parleremo in seguito), mentre nel secondo caso si preferisce comunemente parlare di «variabile» e come tale verrà usata in istruzioni coinvolgenti dati e celle di memoria.

In tutti e due i casi dunque la sintassi della direttiva in esame è la seguente:

```
name LABEL type
```

dove «name» è il nome prescelto per l'etichetta o la variabile, mentre il «type» può essere uno dei seguenti:

— «NEAR» o «FAR» per le etichette in senso stretto

— «BYTE», «WORD» o «DWORD» per le variabili.

Analizziamo dunque i tipi «NEAR» e «FAR»: in generale possiamo dire che se un'etichetta posta all'interno di un certo segmento di codice possiede l'attributo «NEAR», allora può essere «raggiunta» (con salti, ad esempio), solo dall'interno del segmento stesso, mentre è del tutto irraggiungibile a partire da un altro segmento.

Viceversa un'etichetta avente l'attributo «FAR» può essere raggiunta tanto dal segmento di appartenenza quanto da un qualunque altro segmento.

Per vedere più da vicino la differenza tra i due tipi, basti sapere (in quanto torneremo in dettaglio nelle prossime puntate) che per saltare ad un'etichetta di tipo «NEAR» l'assembler considera solo lo scostamento in termini di numero di byte tra l'etichetta stessa e l'istruzione che la richiama: è questo un meccanismo adottato dall'assembler 8086/88 in tutte le istruzioni in salto condizionato e non, nonché nelle chiamate a procedure e prende il nome di «indirizzamento relativo», in quanto in questo caso non interessa conoscere l'esatta posizione dell'etichetta, ma solo quanto è distante dal punto in cui ci troviamo.

Questo fatto sarà molto comodo in quanto, come vedremo, permetterà di generare un codice di programma «indipendente dalla posizione» di memoria in cui è allocato: è spianata la strada alla cosiddetta «rilocabilità» di un programma.

Per quanto riguarda invece il tipo «FAR», ecco che invece in questo caso si conosce in qualunque momento l'esatta posizione dell'etichetta, in termini di «base» e «offset» e di ciò terrà conto l'assembler quando andrà a codificare un'istruzione di salto coinvolgente appunto un'etichetta «FAR»: in questo caso si potrà ancora parlare di rilocabilità ma in più otterremo come detto che l'etichetta in questione potrà essere raggiunta da qualsiasi altro segmento.

Ancora, nel primo caso il codice generato sarà più corto che non nel secondo caso, in quanto in quest'ultimo bisogna «portarsi dietro» l'informazione completa dell'indirizzo dell'etichetta.

Per quanto riguarda i tipi «BYTE», «WORD» e «DWORD», si applicano, come detto a locazioni di memoria contenenti dei dati: già sappiamo che con le direttive «DB», «DW» e «DD» si possono definire delle variabili formate rispettivamente da un byte, da una word (2 byte) e da una double—word (4 byte) e dunque di tipo «BYTE», «WORD» e «DWORD».

Ora, il tipo designato ed abbinato ad una certa variabile può essere cambiato grazie alla direttiva «label», che consente così di fare riferimento a stesse locazioni in memoria, considerandole variabili di tipo differente.

Ad esempio supponiamo di aver definito la seguente variabile:

```
ALFA DW ?
```

che perciò è di tipo «WORD» e come tale potrà essere inserita solo in istruzioni coinvolgenti dati a 2 byte: se invece volessimo fare riferimento al primo byte della variabile ALFA e magari chiamarlo BETA per comodità (ad esempio perché ad esso faremo riferimento più volte nel corso del programma), allora potremo usare la direttiva «label» e si avrà:

```
BETA LABEL BYTE
ALFA DW?
```

In questo caso perciò potremo ad esempio caricare nel registro AX (a 16 bit) la variabile ALFA con l'istruzione

```
MOV AX,ALFA
```

mentre potremo caricare in DL (ad 8 bit) la parte bassa di ALFA con l'istruzione

```
MOV DL, BETA
```

ed anche la parte alta di ALFA nel registro CH (ancora ad 8 bit) con l'istruzione

```
MOV CH,BETA+1
```

Come vedremo nel seguito ci sarà un metodo alternativo per ottenere la stessa funzionalità metodo che non ci obbligherà a creare una nuova variabile.

La direttiva PROC: le procedure

Con il termine di procedura, per chi non lo conoscesse, si intende una parte di un certo programma scritto in assembler 8086/88, che può essere attivato, «chiamato», da un qualsiasi altro punto del programma, tramite la ben nota istruzione «CALL»: per effetto di questa istruzione il programma interrompe la propria sequenza lineare di istruzioni per saltare alla procedura in questione, terminata la quale si ha il ritorno alla sequenza lineare proprio nell'istruzione successiva all'istruzione di chiamata alla procedura stessa.

Questo ritorno avviene grazie ad un'istruzione di «return» («RET»), che consente così di abbandonare la procedura: per quanto riguarda l'attivazione di una certa procedura aggiungiamo che si può effettuare sia a partire dall'etichetta iniziale della procedura stessa, sia a partire da un qualsiasi altro punto «interno» alla procedura, definito con una «label».

Per quanto riguarda l'uscita dalla procedura, abbiamo già detto: il punto o i punti in cui ciò avvenga sono a discrezione del programmatore, grazie all'uso di una o più istruzioni «RET».

La sintassi per la dichiarazione di una procedura è la seguente:

```
name      PROC      [NEAR o FAR]
          ...
        procedura
          ...
name      ENDP
```

dove «name» è, oltre che il nome effettivo della procedura stessa, anche il suo «entry point» principale: «name» è a tutti gli effetti un'etichetta.

Sul significato della clausola «NEAR» o «FAR» valgono esattamente le stesse considerazioni fatte per le «label»: nel caso manchi l'indicazione («NEAR» o «FAR»), allora la procedura intera sarà considerata «NEAR» e come tale potrà essere attivata solo all'interno del segmento di appartenenza, mentre nel caso venga specificato l'attributo «FAR» allora può essere chiamata anche da altri segmenti.

Apparentemente dunque esiste una identità di funzionamento tra il meccanismo che sta alla base delle «label» e delle «procedure»: quello che invece è completamente differente è il comportamento del microprocessore nel caso di salto ad un'etichetta o di chiamata ad una procedura.

È ben noto infatti che nel caso di «salto ad un'etichetta», il processore non ha necessità di «ricordarsi» alcun indirizzo di ritorno in quanto era precisa scelta del programmatore far sì che il controllo passasse ad una precisa parte del programma anziché proseguire per istruzioni successive.

Nel caso di «chiamata a sobroutine» (che qui si chiamano, come detto, «procedure», da leggersi indifferentemente sia in italiano che in inglese) invece il processore deve memorizzare, come è prassi corrente, l'indirizzo di ritorno al quale appunto rientrare al termine della procedura.

In questo caso si innesca per l'8086/88 uno di due meccanismi a seconda che la procedura sia «NEAR» o «FAR»: nel caso di procedura «NEAR», allora l'istruzione «CALL» sarà una «CALL relativa», così come accadeva per le «JMP», nel senso che l'assembler tradurrà il nome della procedura come «scostamento» dell'Instruction Pointer (IP) rispetto al valore attuale.

Ricordiamo dunque che questa «CALL» è limitata al proprio segmento di appartenenza e dunque il microprocessore, quando la incontrerà, salverà nello stack l'offset dell'indirizzo successivo alla «CALL», al quale appunto ritornare: basta che salvi solo l'offset di quell'indirizzo in quanto per ipotesi il segmento è lo stesso.

Viceversa nel caso di procedura di tipo «FAR» allora la faccenda si potrebbe complicare un tantino: innanzitutto ci si potrebbe domandare come fa l'assemblatore a capire se la procedura chiamata è di tipo «NEAR» o «FAR» dato che la «CALL» è la stessa, o forse bisogna specificarlo?

Innanzitutto il programmatore sa già se la procedura è «NEAR» o «FAR» ed in quest'ultimo caso deve aver già posto tale attributo nella definizione della procedura stessa.

L'assembler, davanti ad un'istruzione «CALL» si comporta in questa maniera: se l'etichetta era già definita precedentemente (sia essa «NEAR» che «FAR»), allora si comporterà senza tante complicazioni creando nell'uno o nell'altro caso due tipi differenti di codice, uno contenente uno «scostamento», l'altro contenente l'offset ed il segment, come già sappiamo.

Invece se la procedura si trova «dopo» (si ha in questo caso una «forward reference») allora a seconda del tipo della procedura si avrà una differente interpretazione: se in particolare la procedura risulterà «NEAR» allora andrà tutto bene in quanto l'assemblatore prevede in questo caso che la procedura sia «NEAR», mentre se la procedura risulterà poi di tipo «FAR», allora l'assemblatore segnalerà che una «forward reference» richiede la presenza di un'indicazione aggiuntiva da parte del programmatore.

In questo caso infatti si deve aiutare l'assembler comunicandogli la «lontananza» della procedura tramite una particolare convenzione sulla quale ritorneremo più in dettaglio: tanto per dare un assaggio, invece di scrivere semplicemente

```
CALL procname
```

dove «procname» è appunto il nome della procedura, si dovrà impostare l'istruzione:

```
CALL FAR PTR procname
```

dove troviamo le due parolette «FAR» e «PTR».

La prima è di facile interpretazione, mentre sulla seconda appunto ritorneremo.

Visto dunque come si fa una chiamata ad una procedura, vediamo ora il meccanismo del ritorno.

Ancora una volta, come è ovvio, il tutto dipende dal fatto che la procedura sia «NEAR» o «FAR», dal momento che il microprocessore dovrà riprendersi dallo stack rispettivamente solo l'offset oppure sia l'offset che il segment dell'indirizzo di ritorno.

Ecco che dunque sorgono nuove regole dettate appunto dal meccanismo di chiamata—ritorno: in particolare se una procedura è definita «NEAR» allora tutti i suoi eventuali entry—point devono essere delle label «NEAR», ottenibili sia con la direttiva «LABEL» vista precedentemente, sia con l'etichetta inserita nell'istruzione, ad esempio rispettivamente:

```
PROCEDURA PROC       NEAR
          ...
ETICH     LABEL      NEAR
          MOV  AX,ALFA
          ...
          RET
          ...
PROCEDURA ENDP
```

oppure più semplicemente

```
PROCEDURA PROC       NEAR
          ...
ETICH:    MOV  AX,ALFA
          ...
          RET
          ...
PROCEDURA ENDP
```

In entrambi i casi le istruzioni «RET» saranno di tipo «NEAR» (sì, anche loro!) ed il micro—processore riprenderà dallo stack solo l'offset dell'indirizzo di ritorno.

Stesso discorso vale se la procedura è definita come «FAR», nel qual caso i vari entry—point devono essere tutti di tipo «FAR» e perciò definiti con la direttiva «LABEL FAR»: entry point non definiti in tal modo provocherebb-

bero subito una segnalazione d'errore.

Ovviamente all'interno di una procedura di tipo «FAR» possono esistere etichette di tipo «NEAR», ma altrettanto ovviamente saranno punti a cui si salterà a partire dalla procedura stessa oppure da...

Ebbene sì, ci stiamo addentrando in un classico terreno minato, ma visto che ci siamo è il caso di studiare un piccolo esempio.

Supponiamo di avere un programma principale, posto in un certo segmento, il quale (malandrino!) chiami in momenti successivi due procedure «FAR», dunque poste in un segmento differente: fin qui nulla di male in quanto già sappiamo che le chiamate saranno del tipo «CALL» se le procedure sono definite già in precedenza oppure «CALL FAR PTR» nel caso di «forward reference», mentre le «RET» saranno di tipo «FAR», anche se in questo caso l'istruzione rimane semplicemente «RET» senza altri attributi.

Supponiamo ora, appunto per complicare le cose, che la prima procedura chiami a sua volta la seconda procedura, il che non sarebbe nemmeno tanto strano: Come farà il povero programmatore e cioé noi?

Vediamo perciò un piccolo esempio (A), che altro non è che il file PROVA.LST prodotto dall'assemblatore MASM a partire dal file sorgente PROVA.ASM: il tutto su di un IBM compatibile.

Volutamente abbiamo lasciato l'errore sulla chiamata alla procedura PROC3: appunto come dicevamo prima l'assemblatore si aspetta per default una procedura «NEAR», ma trovandola invece «FAR» segnala l'errore.

Ponendo dunque l'istruzione «CALL FAR PTR PROC3» al posto della precedente, allora tutto va bene, come si vede nel frammento (B) del nuovo file PROVA.LST ottenuto riassemblando il file PROVA.ASM.

Torniamo dunque al nostro problema iniziale: come ci si deve comportare nelle chiamate tra procedure «FAR» poste però nello stesso segmento?

Come si vede dal listing completo, il tutto è «trasparente» (e perciò invisibile) per il programmatore, al quale basta effettuare la chiamata con

A

```
                                NAME PROVA

0000                    SEG1    SEGMENT

0000                    PROC2   PROC FAR
0000  8B C8                     MOV CX,AX
0002  81 C1 0100                ADD CX,100H
0006  CB                        RET
0007                    PROC2   ENDP

0007                    PROC1   PROC FAR
0007  B8 0005                   MOV AX,5
000A  9A 0000 ---- R            CALL PROC2
000F  8B C1                     MOV AX,CX
0011  CB                        RET
0012                    PROC1   ENDP

0012                    SEG1    ENDS

0000                    SEG2    SEGMENT
                                ASSUME CS:SEG2
0000  B8 0004                   MOV BX,4
0003  9A 0007 ---- R            CALL PROC1
0008  03 D8                     ADD BX,AX
000A  9A 0000 ---- R            CALL PROC2
000F  E8 0017 R                 CALL PROC3
E r r o r  ---          71:Forward needs override or far
0012  2D 07FF                   SUB AX,7FFH
0015  EB FE             LOOP:   JMP LOOP

0017                    PROC3   PROC FAR
0017  03 C0                     ADD AX,AX
0019  CB                        RET
001A                    PROC3   ENDP

001A                    SEG2    ENDS

                                END
```

B

```
0000                    SEG2    SEGMENT
                                ASSUME CS:SEG2
0000  B8 0004                   MOV BX,4
0003  9A 0007 ---- R            CALL PROC1
0008  03 D8                     ADD BX,AX
000A  9A 0000 ---- R            CALL PROC2
000F  9A 0019 ---- R            CALL FAR PTR PROC3
0014  2D 07FF                   SUB AX,7FFH
0017  EB FE             LOOP:   JMP LOOP

0019                    PROC3   PROC FAR
0019  03 C0                     ADD AX,AX
001B  CB                        RET
001C                    PROC3   ENDP

001C                    SEG2    ENDS
```

C

```
                                NAME PROVA

0000                    SEG1    SEGMENT

0000                    PROC2   PROC FAR
0000  8B C8             PROC2N: MOV CX,AX
E r r o r  ---          62:No or unreachable CS
0002  81 C1 0100                ADD CX,100H
0006  CB                        RET
0007                    PROC2   ENDP

0007                    PROC1   PROC FAR
0007  B8 0005                   MOV AX,5
000A  E8 0000 R                 CALL PROC2N
000D  8B C1                     MOV AX,CX
000F  CB                        RET
0010                    PROC1   ENDP
```

D

```
                                NAME PROVA

0000                    SEG1    SEGMENT

                                ASSUME CS:SEG1
0000                    PROC2   PROC FAR
0000  8B C8             PROC2N: MOV CX,AX
0002  81 C1 0100                ADD CX,100H
0006  CB                        RET
0007                    PROC2   ENDP

0007                    PROC1   PROC FAR
0007  B8 0005                   MOV AX,5
000A  E8 0000 R                 CALL PROC2N
000D  8B C1                     MOV AX,CX
000F  CB                        RET
0010                    PROC1   ENDP

0010                    SEG1    ENDS

0000                    SEG2    SEGMENT
                                ASSUME CS:SEG2
0000  B8 0004                   MOV BX,4
0003  9A 0007 ---- R            CALL PROC1
0008  03 D8                     ADD BX,AX
000A  9A 0000 ---- R            CALL PROC2
000F  E8 0017 R                 CALL PROC3
0012  2D 07FF                   SUB AX,7FFH
0015  EB FE             LOOP:   JMP LOOP

0017                    PROC3   PROC
0017  03 C0                     ADD AX,AX
0019  C3                        RET
001A                    PROC3   ENDP

001A                    SEG2    ENDS

                                END
```

«CALL» ed il ritorno con «RET».

Al limite se la procedura chiamata non è ancora stata definita allora la chiamata avverrà con l'ormai ben nota «CALL FAR PTR».

Ma noi non siamo soddisfatti, vogliamo forzare la mano... Non è possibile, ci domandiamo subdolamente, far sì che la chiamata di PROC2 da parte di PROC1 sia forzata al tipo «NEAR» dal momento che entrambe le procedure si trovano nello stesso segmento? Come dire: dato che il segmento è lo stesso non ci va bene che la chiamata sia di tipo «FAR». Per fare ciò sappiamo che dobbiamo creare un secondo entry—point alla PROC2, entry—point stavolta di tipo «NEAR»: stiamo volutamente trasgredendo la regola che vuole tutti gli entry point dello stesso tipo, semplicemente per vedere fin dove è possibile arrivare.

Dunque creiamo una label automaticamente «NEAR» all'interno di PROC2 (grazie ai «:» che seguono la label «PROC2N») e sostituiamo la chiamata in PROC1 con una «CALL PROC2N»: assemblando il tutto otteniamo un listato del quale mostriamo solo una parte (C).

L'errore che appare dovrebbe essere chiaro: per effettuare un salto all'interno di uno stesso segmento dobbiamo aggiungere la direttiva «ASSUME» (sulla quale ritorneremo la prossima puntata), cosa che facciamo subito per ottenere stavolta un listato (D), miracolosamente, privo di errori!

Siamo riusciti ad ingannare abilmente l'assemblatore, il quale da questo punto di vista non poteva farci assolutamente nulla: abbiamo forse trovato un baco di tale potente programma, dato che ci aspetteremmo almeno un «warning» fosse uscito?! Forse sì, ma siamo ben lontani dalla programmazione «normale»...

Il fatto è che solo apparentemente abbiamo ingannato il malcapitato MASM ed infatti la sua rivincita se la prende quando andremo a "linkare" e poi ad eseguire il programmino: il nostro tentativo sarà infruttuoso ed inspiegabilmente il programma non girerà assolutamente, mandando in crisi il nostro povero PC...

Riusciranno i nostri lettori a trovare la soluzione di questo mistero? Quelli più smaliziati, che già conoscono bene l'assembler dell'8086/88 troveranno quasi subito dov'è il «busillis» e cioé in quale punto accade qualcosa di strano: invitiamo perció i lettori a rispondere a questo mini—quiz «banale».

In palio non c'è assolutamente nulla, se non la citazione nel corso della rubrica: la risposta verrà comunque data in una delle prossime puntate.

MC

MODULUS. L'AMICO DELL'HOMO SAPIENS.

Tu che sei un homo sapiens lo sai, i computer, possono fare le cose più incredibili. Però di solito se ne stanno belli tranquilli al loro posto, senza spostarsi di un millimetro. Modulus, no. Lui si diverte soltanto se gli fai fare quattro passi. Modulus, infatti, non è solo cervello, ma voce, occhi, braccia, sensori e ruote: tutto quello che occorre ad un personal robot per essere rivoluzionario.

In cosa consiste la sua rivoluzione? Nelle sue prestazioni, innanzitutto, che erano incredibili fino ad oggi per un robot delle sue dimensioni e del suo costo.

E nella sua modularità, che consente l'acquisto successivo di elementi componibili fino a raggiungere la configurazione di un androide. Ognuno di questi step successivi, naturalmente, ha la sua specificità ed una sua ragion d'essere autonoma. Se hai un Home Computer, un po di pratica e molta fantasia, la "Versione Base" fa proprio per te. Ben programmata è capace di muoversi a due velocità diverse rilevando gli urti e fermandosi, tracciare un percorso preordinato con precisione perfetta, disegnare con pennarelli, segnalare il suo funzionamento tramite un display, dialogare con un Home Computer via cavo o etere grazie ad un apparecchio di comunicazione a radiofrequenza. Insomma, per dirla in due parole, la "Versione Base" sarà la tua raffinata periferica semovente.

Se ti fa piacere sapere in anticipo che stasera pioverà, allora "Service & Security Robot" è il tuo compagno ideale. In questa configurazione Modulus dispone oltre che della base, anche di una serie di moduli a spicchio, ciascuno dei quali può contenere componenti hard/software. Questa "torta tecnologica" rende possibile, perciò, una crescita il cui limite evolutivo non è definibile.

Attualmente sono disponibili una stazione meteorologica, capace di prevedere le condizioni atmosferiche (se messa in relazione con un computer); un sistema SONAR per la misura di distanze e l'individuazione di sorgenti di luce, calore, rumore e l'inseguimento di umani; un sistema VOCE per una sintesi vocale di elevata qualità e per il riconoscimento di suoni di comando; un sistema SICUREZZA per la rilevazione di fughe di gas, acqua e fumo.

Inoltre una CPU a 16 bit dotata di 128 Kbyte Ram, 128 Kbyte Rom, 16 Kbyte Ram con alimentazione tampone e cartuccia Rom per i programmi applicativi, rende possibile il funzionamento di tutti i sistemi anche svincolati da qualsiasi Personal Computer.

Ma quello che il "Service & Security Robot" ha di meglio è la possibilità dell'inserzione di un braccio. Questo, oltre ad essere caratterizzato da una ampia possibilità di movimento, una velocità nettamente superiore a quelle fornite dai robot della precedente generazione ed una precisione elevata, dispone di un particolare controllo della forza di presa sulla mano.

Mica male! Oltre a dirti che tempo farà, Modulus ti porge anche l'ombrello! Se poi sei nato sotto una radice quadrata, allora ne farai delle belle con "Moddy". La versione più evoluta di Modulus può fare tutto quello che fanno le precedenti e centomila altre in più, perché ha anche dué braccia una testa e due occhioni molto, molto espressivi.

Anche se non sei uno scienziato folle, però, potrai ricavare grande piacere dalla compagnia di "Moddy", che con la sua voce o con il suo monitor ha mille cose da raccontarti mentre ti dà una mano nelle occupazioni domestiche, nei tuoi hobbies preferiti o nelle tue attività più impegnative.

E adesso, homo sapiens, per saperne di più non ti resta che recarti nei migliori negozi di elettronica ed HiFi; oppure ritaglia il coupon qui sotto e riceverai ampio materiale illustrativo. Intesi?

Desidero ricevere maggiori informazioni su Modulus **MC/11/86**

Nome _____

Via _____

Cap. _____ Città _____

MODULUS

A NEW ERA IN HOME ROBOTICS

MODULUS È PRODOTTO E DISTRIBUITO DA SIRIUS.

MILANO FIORI PALAZZO F2 - 20094 ASSAGO (MI) ITALY - TEL. (02) 8245321 - TELEX 325584

Advema Cooper