ASSEMBLER 8086 8088

di Pierluigi Panunzi

Il set di istruzioni ISTRUZIONI DI CONTROLLO

prima parte

A partire da questa puntata affronteremo l'analisi dettagliata delle istruzioni che consentono di effettuare un completo controllo sulla programmazione ed in particolare sul flusso di istruzioni: parleremo infatti delle istruzioni di salto, condizionato e non, delle chiamate a subroutine e successivi ritorni al programma chiamante, e delle istruzioni di controllo dei loop.
Per i nostri lettori non si tratta certo di novità in quanto da un lato, nei vari esempi riportati nel corso delle varie puntate, abbiamo più volte incontrato tali tipi di istruzioni, senza dubbio indispensabili in un qualsiasi programma che si rispetti e di una certa complessità; d'altro canto poi tali istruzioni in generale ricalcano analoghe istruzioni presenti in un qualsiasi microprocessore e che in un certo senso ne rappresentano il «corredo».
Nel caso dell'Assembler 8086/88 si ritroveranno ancora una volta argomentazioni relative alla particolare gestione della memoria detta «a segmenti» ed alla dualità «segment-offset». ■

Le istruzioni di controllo - JMP

Come è universalmente noto, anche da chi è alle prime armi nel campo della programmazione, un'istruzione di salto (la «JMP» per l'appunto, che sta per «JuMP») consente di alterare la naturale caratteristica di sequenzialità di esecuzione di un qualunque programma, forzando l'esecuzione di istruzioni poste in un qualsiasi punto della memoria, invece di quelle immediatamente successive: ciò potrà avvenire in maniera «diretta» oppure «indiretta».

Analizziamo dapprima il caso di salti «diretti».

In questo caso il programmatore indicherà il punto a cui saltare mediante un'etichetta e così il programma salterà «direttamente» alla nuova locazione.

L'assemblatore tradurrà opportunamente il «nome» dell'etichetta a cui saltare in modo tale che poi il microprocessore, all'atto dell'esecuzione stessa del programma, salti effettivamente all'istruzione desiderata e cioè all'indirizzo voluto.

Il meccanismo di «traduzione» etichetta-indirizzo fisico, come dicevamo, è strettamente legato alla gestione segmentata della memoria e dipende in maniera particolare dal fatto che l'etichetta di arrivo può o meno appartenere al segmento «attuale», quello in cui il microprocessore ha eseguito le ultime istruzioni: a seconda infatti che l'etichetta si trovi nello stesso segmento oppure in un altro si avrà una differente codifica prima e una differente esecuzione poi.

Supponiamo dunque innanzitutto che l'etichetta si trovi all'interno dello stesso segmento dell'istruzione «JMP» considerata: si avrà ad esempio una situazione simile

```
CODE    SEGMENT
        ...
        ...
ALFA:   MOV AL,33H
        ...
        ...
        JMP ALFA
        ...
        ...
CODE    ENDS
```

nella quale l'etichetta si trova «prima» dell'istruzione di salto, oppure una situazione del tipo

```
CODE    SEGMENT
        ...
        ...
        JMP BETA
        ...
        ...
BETA:   CMP BX,CX
        ...
        ...
CODE    ENDS
```

in cui il «target» (letteralmente il «bersaglio», come si dice in gergo) si trova

«dopo» l'istruzione di salto.

Abbiamo suddiviso il caso di salto all'interno dello stesso segmento (detto «intra-segment jump») nei due sotto-casi visti in quanto è differente il comportamento dell'assemblatore nei due frangenti.

Sottolineiamo a questo punto che in questi casi l'assemblatore si comporta nel modo «più risparmiatore» possibile, come già abbiamo visto in altri casi, nel senso che laddove può risparmiare un byte lo fa senza tanti scrupoli.

Analizziamo dunque il primo sotto-caso, nel quale l'etichetta viene definita «prima» delle sue utilizzazioni come «target»: in questo caso l'assemblatore, mano a mano che analizza il nostro listato sorgente, ad un certo punto incontrerà l'etichetta «ALFA» ed allora, non conoscendola, la memorizzerà all'interno di una sua apposita tabella interna di riferimento, insieme all'offset che ha tale etichetta, offset che nel frattempo, istruzione dopo istruzione, viene correttamente aggiornato.

Nel caso in cui a questo punto già trovasse proprio quell'etichetta all'interno della sua tabella, il che significa che già c'era stato un punto in cui l'etichetta in questione era stata usata per definire una locazione di memoria, allora segnalerebbe il suo disappunto con un messaggio di errore che suona come «etichetta duplicata».

Fatto dunque ciò, l'assemblatore continuerà la sua analisi del sorgente, aggiornando istruzione dopo istruzione l'offset corrente: allorché troverà l'istruzione di salto «JMP ALFA», cercherà all'interno della sua tabellina l'offset corrispondente ed effettuerà un calcolo semplicissimo, la differenza tra l'offset dell'etichetta e quello attuale.

Se tale differenza (e qui si innesca il meccanismo «economico»), espressa in complemento a 2, può essere rappresentata da un solo byte allora così la esprime, mentre in caso contrario la esprime su due byte: in particolare se la differenza tra gli offset, che viene correntemente indicata con il termine «displacement», è minore di 127 (se positiva) o di −128 (se negativa) allora verrà rappresentata con un byte, che infatti come noto può rappresentare in notazione «complemento a 2» un numero compreso tra −128 e +127.

Si parla in tali casi di «direct short jump» o «salto corto diretto».

In tutti gli altri casi, e cioè per valori di displacement compresi tra −32768 e +32767, allora la codifica avverrà su due byte: i lettori preoccupati se in tale modo da un'istruzione di salto si può raggiungere «qualsiasi» locazione all'interno del segmento, troveranno risposta nel fatto oramai più che ripetuto che un segmento di per sé è ampio proprio 65536 byte (64 kbyte) e dunque ogni locazione è raggiungibile, con un displacement positivo o negativo.

Ovviamente nei due casi in cui il displacement è espresso con un solo byte oppure con due, l'opcode relativo sarà completamente differente (ci mancherebbe...): nel caso di «short jump» l'opcode vale «EBH», mentre nell'altro caso vale «E9H».

Vediamo ora cosa succede nel caso in cui l'assemblatore incontra un'istruzione di salto, la cui etichetta ancora non conosca: in tal caso, lungi dal segnalare tale fatto con un errore, il nostro «economo» si comporterà viceversa in maniera contraria alla sua natura, preparando la strada al caso peggiore e cioè presupponendo che l'etichetta posta nell'istruzione abbia un displacement maggiore di 127 o −128 byte, lasciando perciò lo spazio a due byte, il cui contenuto verrà poi effettivamente calcolato successivamente all'atto del «secondo passo».

In tal modo potranno succedere due fatti legati all'effettivo displacement calcolato: se tale valore potrà essere espresso con due byte allora non succederà nulla di particolare, mentre nel caso in cui dovrà essere espresso da un byte soltanto, allora il secondo byte inutilizzato verrà riempito con la codifica di un'istruzione «NOP» e cioè «90H».

Ancora una volta il programmatore può andare incontro all'assemblatore fornendogli l'indicazione se l'etichetta che dovrà ancora individuare si trova o no a meno di 127 byte dall'istruzione di jump.

In particolare se si presuppone che la locazione sarà «vicina», allora invece di «JMP BETA» si potrà scrivere

```
JMP SHORT BETA
```

al che il nostro solerte assemblatore riserverà per il displacement solamente un byte (visto che così gli abbiamo detto noi), ed in caso che tale scelta risulti poi errata allora il «nostro» ci comunicherà che l'etichetta non può essere raggiunta (s'intenda «con un direct-short-jump»).

Ciò può essere utile in quei casi in cui dobbiamo mantenere il nostro programma all'interno di una zona ben precisa e perciò dobbiamo contare i byte utilizzati e risparmiarli di conseguenza.

Altro tipo di salto diretto è quello che viene denominato «inter-segment direct jump», in gergo «salto lungo o lunghissimo» che si ha quando l'etichetta a cui si salta e l'istruzione di salto si trovano in due segmenti differenti, ad esempio in un programma del tipo

```
CODE1   SEGMENT
        ...
        ...
        JMP FAR PTR LONTANA
        ...
        ...
CODE1   ENDS
CODE2   SEGMENT
        ...
        ...
LONTANA LABEL FAR
        ...
        ...
CODE2   ENDS
```

Vediamo innanzitutto che in questo caso il fatto che l'istruzione di JMP si trovi «fisicamente» prima della definizione dell'etichetta è un fatto puramente accessorio, in quanto basterebbe definire dapprima il segmento CODE2 e poi CODE1 per aversi la definizione dell'etichetta prima dell'istruzione di salto che la interessa: questo per dire che nel caso di salto inter-segment poco importa l'ordine di «apparizione» della definizione o dell'«utilizzazione» di una certa etichetta.

In particolare l'assemblatore in ogni caso deve trovarsi a che fare con un'etichetta di tipo «FAR» e cioè appartenente ad un differente segmento, (contrapposta alle etichette «NEAR» appartenenti allo stesso segmento) e perciò allorché ne incontrerà la definizione, non la memorizzerà più nella tabella precedentemente vista, ma in un'altra tabella appunto riservata alle etichette di tipo «FAR» delle quali oltre al nome memorizzerà l'offset ed il segment completo.

Analogamente incontrando un'istruzione di JMP relativa ad un'etichetta «FAR» subito saprà di dover lasciare in ogni caso quattro byte, due per l'offset e due per il segment: se già aveva incontrato l'etichetta, allora tali 4 byte saranno subito riempiti, mentre in caso contrario si aspetterà il «secondo passo».

In questo caso si vede dunque che l'assemblatore non si comporta in maniera differente a seconda se si incontra prima l'etichetta o la JMP, ed in ogni caso tradurrà la «JMP» stessa con il codice «EAH».

Vediamo ora dal punto operativo cosa farà il microprocessore quando incontra sul suo cammino un'istruzione di «JMP» (un codice operativo «E9H», «EBH» o «EAH» rispettivamente seguiti da 1, 2 o 4 byte): nel primo caso («intra-segment direct short jump») sommerà al valore attuale dell'IP («Istruction Pointer») il valore espresso dal byte di displacement, estendendolo però a 16 bit.

Nel secondo caso (si chiama semplicemente «intra-segment direct jump») all'offset attuale sommerà il valore a 16 bit espresso dal displacement, sen-

za ulteriori problemi e, come nel caso precedente, salterà all'istruzione avente tale offset.

Nel terzo caso («inter-segment direct jump») invece il microprocessore assumerà i primi due byte come valore nuovo per l'IP ed i successivi 2 byte come nuovo CS («Code Segment»), senza effettuare cioè calcoli di sorta, ma semplicemente impostando nuovi valori ai due registri IP e CS: a questo punto il salto all'etichetta «FAR» desiderata è puramente automatico...

Fin qui abbiamo parlato di salti «diretti», quelli cioè in cui siamo noi a porre direttamente nell'istruzione l'etichetta a cui vogliamo che il microprocessore salti.

I salti indiretti

In questo caso l'«indirettezza» consiste nel fatto che l'etichetta a cui saltare non è esplicitamente indicata nell'istruzione, ma bensì è raggiungibile indirettamente attraverso il contenuto di una locazione di memoria o di un registro.

Si parlerà, nel caso di salti indiretti, solamente di «intra-segment indirect jump» o di «inter-segment indirect jump» a seconda che l'etichetta da raggiungere rispettivamente appartenga o meno allo stesso segmento di cui fa parte l'istruzione di salto: in questo caso mancano i «short jump» ed ora vedremo subito il perché.

Iniziamo dalle istruzioni di salto indiretto all'interno di un segmento: in tal caso non si avrà più a che fare con un displacement (che potrebbe stare in uno o due byte), ma bensì si ha a che fare con valori assoluti dell'offset dell'etichetta a cui si deve saltare.

Dal momento che dunque abbiamo a che fare con un offset «vero» (e perciò non da calcolarsi), nasce spontanea la possibilità di salvarlo all'interno di un registro o di una cella di memoria di tipo «WORD» per poi saltare indirettamente all'etichetta.

L'utilizzazione migliore di un'istruzione di salto indiretto è la traduzione a bassissimo livello di un'istruzione «CASE» del Pascal o di una «ON × GOTO...» del Basic e cioè tutte quelle occasioni in cui, in base al valore «n» di una certa variabile, si debba saltare all'«n-esima» etichetta di una certa tabella di etichette.

Supponiamo per esempio di scrivere una routine di gestione di un'unità a dischi, per la quale, in base al valore contenuto nell'accumulatore, si possa effettuare una differente operazione, ad esempio una lettura da disco, una scrittura verso il disco o il reset dell'unità a dischi.

Un esempio di frammento di programma che gestisce il tutto è il seguente

```
CODE      SEGMENT
          ...
          ...
TABELLA   DW READ
          DW WRITE
          DW RESET
;
ROUTINE:  MOV BL,AL
          MOV BH,0
          SHL BX,1
          JMP TABELLA[BX]
          ...
          ...
;
READ:     ...
          ...
          ...
;
WRITE:    ...
          ...
          ...
;
RESET:    ...
          ...
          ...
CODE      ENDS
```

In questo esempio vediamo che il valore contenuto in AL viene posto in BL, dopodiché viene costruito un valore a 16 bit in BX, che viene poi moltiplicato per due per fornire un indice all'interno della tabella di indirizzi chiamata appunto «TABELLA».

In questo caso l'assemblatore, all'atto della codifica dell'istruzione di salto, troverà nel campo riservato all'etichetta il nome di una variabile, con l'aggiunta del registro BX come indice: a questo punto saprà trattarsi di un salto indiretto e codificherà in un'altra maniera ancora l'istruzione.

Nel caso in cui si tratti di una locazione di memoria, potremo dunque avere tutti i casi possibili di indicizzazione attraverso registri indice e registri base, con o senza offset aggiuntivi, ben noti in quanto tipici dell'Assembler 8086/88: alcuni esempi sono

```
JMP ETICHETTA ;attenzione ETICHETTA
    e' una cella!
JMP TABLE[BX]
JMP ALFA[BP][SI+5]
```

come pure

```
JMP [BX]
```

In tutti questi casi l'offset a cui si salterà sarà rispettivamente contenuto nella locazione chiamata subdolamente «ETICHETTA», in una locazione all'interno della tabella «TABLE» (a seconda del valore di BX), in una locazione posta all'interno della tabella «ALFA» a seconda del valore congiunto del contenuto del registro BP (il che obbliga la tabella ad essere residente nello Stack Segment, ricordate?) e del contenuto di SI aumentato di 5 ed infine nella locazione il cui offset è posto in BX.

In particolare quest'ultima istruzione («JMP [BX]») si legge: «salta all'istruzione il cui offset è contenuto nella cella di memoria il cui offset è posto nel registro BX». Lasciamo al lettore il compito di «leggere» la penultima istruzione...

Nel caso invece in cui come operando dell'istruzione di JMP compaia un registro, allora il suo contenuto sarà l'offset dell'etichetta a cui saltare, come visto nel caso precedente: si potranno avere casi del tipo

```
JMP CX
JMP AX
JMP BP
JMP SP
```

come pure

```
JMP BX
```

mentre viceversa non sono possibili salti relativi a registri di segmento: in particolare l'ultimo esempio, a differenza dell'ultimo del lotto precedente, si «legge»: «salta alla locazione di memoria il cui offset è contenuto nel registro BX», mentre nel caso precedente BX forniva l'indirizzo della cella di memoria all'interno del quale trovare l'offset a cui saltare.

Abbiamo detto che sono possibili anche gli «inter-segment indirect jump»: questi si ottengono soltanto attraverso locazioni di memoria e non sono possibili attraverso registri (che dovrebbero in questo caso essere presi in coppia).

In questo caso dunque si avrà a che fare con locazioni di memoria che per loro natura saranno delle «double-word» o più comodamente con coppie di «word» dove comunque i primi due byte devono contenere l'offset della locazione ed i successivi due byte il segmento di appartenenza.

Volendo estendere l'esempio precedente a routine appartenenti a segmenti completamente differenti, otteniamo un programma che è modificato così:

```
CODE      SEGMENT
          ...
          ...
TABELLA   DW READ
          DW SEG READ
          DW WRITE
          DW SEG WRITE
          DW RESET
          DW SEG RESET
;
ROUTINE:  MOV BL,AL
          MOV BH,0
          SHL BX,1
          SHL BX,1
          JMP DWORD PTR TABELLA[BX]
          ...
          ...
CODE      ENDS

READ_CODE SEGMENT
READ:     ...
          ...
          ...
READ_CODE ENDS

WRITE_CODE SEGMENT
WRITE:    ...
          ...
          ...
WRITE_CODE ENDS

RESET_CODE SEGMENT
RESET:    ...
          ...
          ...
RESET_CODE ENDS
```

In questo esempio la tabella conterrà sia gli offset che i segmenti delle

routine, per un totale ogni volta di 4 byte e perciò BX deve essere moltiplicato per 4; inoltre nell'istruzione di salto bisognerà indicare che la locazione da cui prelevare l'indirizzo completo è una «double-word» (DWORD PTR) in quanto un'istruzione «JMP TABELLA[BX]» è riferita ad un salto intra-segment dal momento che «TABELLA» è definita di tipo «WORD» e non «DWORD».

Dunque come regola l'assemblatore genererà, in caso di indirizzamento indiretto, un salto «intra-segment» se la variabile è di tipo «WORD» oppure un salto «inter-segment», se di tipo «DWORD»: in quest'ultimo caso, la direttiva «DWORD PTR» serve ad aggiustare le cose se la variabile è di tipo «WORD».

Dal momento che con la direttiva «LABEL» possiamo definire ad esempio «DWORD» ciò che è «WORD», ecco che il programma in esame si potrebbe scrivere pure in questo modo

```
CODE        SEGMENT
            ...
            ...
TAB_DD      LABEL DWORD
TABELLA     DW READ
            DW SEG READ
            DW WRITE
            DW SEG WRITE
            DW RESET
            DW SEG RESET
```

```
ROUTINE:    MOV BL,AL
            MOV BH,0
            SHL BX,1
            SHL BX,1
            JMP TAB_DD[BX]
            ...
            ...
CODE        ENDS
```

dove appunto TAB_DD è proprio di tipo «DWORD».

Analogamente al caso dell'«intra-segment indirect jump», però a patto che la locazione a cui si fa riferimento sia di tipo (esplicitamente o implicitamente) «DWORD», sono consentite istruzioni del tipo

```
JMP DWORDTAB[BX][SI+1]
JMP DWORD PTR ALFA[BP-4]
```

come pure

```
JMP DWORD PTR [BX]
```

dove in quest'ultimo caso si salterà all'istruzione il cui indirizzo completo è posto nella double-word (all'interno del Data Segment, non dimentichiamocelo...) il cui offset è posto all'interno del registro BX.

Terminiamo l'analisi dell'istruzione JMP con la considerazione che in tutti i casi in cui si ha un salto indiretto attraverso una locazione di memoria esplicita o implicita, valgono le considerazioni relative al cosiddetto «Segment Override», secondo il quale (lo ricordiamo) un riferimento ad una locazione appartenente ad un certo segmento per default, può essere «scavalcata» dall'indicazione di un altro segmento.

Infatti, se supponiamo che la nostra tabella di salti (indiretti o diretti che siano) sia posta all'interno del Code Segment (è il caso di programmi che gireranno su EPROM, ad esempio), allora per indirizzare correttamente tale tabella, bisogna effettuare l'«override» indicando esplicitamente il registro CS.

Riprendendo l'ultimo esempio, se la tabella TAB_DD fosse posta nel Code Segment, allora bisogna scrivere

```
JMP CS:TAB_DD[BX]
```

dal quale ancora una volta si vede quali e quante possibilità ci offre questo assemblatore.

Nella prossima puntata continueremo il discorso sulle istruzioni di salto ed in generale sulle istruzioni di controllo, forzatamente interrotto a questo punto per mancanza di spazio.

MC