ASSEMBLER 8086 8088

di Pierluigi Panunzi

Il set di istruzioni
Istruzioni di controllo

seconda parte

Proseguiamo in questa puntata l'analisi dettagliata delle istruzioni di controllo del flusso di un programma, che sono le istruzioni di salto, condizionato e non, le chiamate a subroutine e successivo ritorno al programma chiamante, e le istruzioni di controllo dei loop.
La scorsa puntata è stata dedicata interamente all'istruzione JMP, se non altro per introdurre dei concetti che ci ritorneranno utili nell'analisi delle successive istruzioni di salto, in particolare l'istruzione CALL.

Le istruzioni di controllo - CALL
Le «direct CALL»

Eccoci dunque all'istruzione che consente di effettuare la chiamata ad una subroutine, con la certezza (grazie alla presenza di un'istruzione di «RET», ritorno) che al termine dell'elaborazione della subroutine stessa si ritornerà proprio all'istruzione successiva alla chiamata della subroutine: come si vede non c'è niente di nuovo rispetto a quelli che sono i dettami del funzionamento di un sottoprogramma.

Quello che invece è semmai differente è la modalità di indicazione al microprocessore della locazione a cui si deve saltare per proseguire nell'elaborazione: praticamente, a parte alcune differenze che non mancheremo di far notare, si hanno tutte le possibilità offerte dall'istruzione, JMP, analizzata la scorsa puntata.

Così come per la JMP, per la CALL sono possibili due tipi di salti: diretti ed indiretti. Tra l'altro parleremo di «salti» e non di «chiamate» perché in realtà con una CALL si effettua un «salto», preceduto, come è facile intuire, dal salvataggio nello stack del cosiddetto «indirizzo di ritorno»: in questo caso, dato che per l'8086/88 un indirizzo è espresso come coppia «offset-segment», allora a seconda dei casi verrà salvato nello stack solo l'offset oppure entrambe le quantità, come vedremo caso per caso.

Iniziamo dunque dalle CALL «dirette», che ci consentono di saltare ad una routine indicando nell'istruzione l'etichetta della routine stessa.

A seconda poi che l'etichetta e perciò la routine completa sia posta all'interno dello stesso segmento dell'istruzione CALL oppure sia posto in un altro segmento, si parlerà rispettivamente di «direct intra-segment CALL» e di «direct inter-segment CALL»: iniziamo dal primo sottocaso.

Le «direct intra-segment CALL» sono dunque «salti» (chiamate...) a routine all'interno dello stesso segmento dell'istruzione chiamante e sono del tipo

```
CALL ETICHETTA
```

dove «ETICHETTA» può essere posta in un qualsiasi punto all'interno del segmento.

A differenza del caso dell'istruzione JMP, l'istruzione CALL prevede nella codifica sempre due byte indicanti ancora una volta di quanto bisogna spostarsi in avanti o indietro (rispetto all'offset attuale) per raggiungere l'etichetta desiderata: in tal modo comunque ci si può muovere in avanti fino ad un massimo di 32767 locazioni ed

all'indietro fino ad un massimo di 32768 byte.

Non esiste in questo caso la «CALL corta», in cui il salto avverrebbe ad un'etichetta posta tra 127 byte in avanti e 128 byte all'indietro: una chiamata ad un'etichetta posta a 10 byte di distanza in avanti viene comunque codificata con tre byte (uno di codice operativo, pari a E8H, e due per il cosiddetto «displacement»).

Va da sé che se l'etichetta da raggiungere (il «target») è posta in avanti rispetto alla CALL, allora il displacement sarà positivo e il valore espresso in esadecimale sarà compreso tra 0000H, e 7FFFH, mentre se il target è posto all'indietro, allora il displacement posto nei due byte potrà assumere un qualsiasi valore tra 8000H e FFFFH.

Dal punto di vista dell'assemblatore, che legge il nostro file contenente le istruzioni del programma, non ci sono più i problemi visti la scorsa puntata legati al fatto che l'etichetta, all'atto della chiamata, sia già «conosciuta» oppure no, dal momento che in ogni caso il displacement è, come detto, posto in due byte: tutto è molto tranquillo. Succede però un fatto un po' strano, legato al fatto che per l'8086/88 ha fondamentale importanza il concetto di «segmento», inteso come insieme di locazioni di memoria. Supponiamo dunque di avere un programma del tipo mostrato in figura A, in cui la chiamata ad una subroutine, la quale è posta alle prime locazioni di un segmento, è viceversa posta «in fondo» al segmento: in questo caso la chiamata «CALL ALFA» è posta all'offset 0FFF0H, mentre «ALFA» è posta ad offset 0000H.

Siamo dunque nel caso di «target» posto prima della chiamata stessa, come dire che la chiamata avviene ad una subroutine posta «indietro» rispetto alla chiamata stessa. Giusto, no?! No! Qui si innesca un meccanismo a prima vista strano...

Se facciamo un po' di conti vediamo che il displacement da porre all'interno della CALL, per poter raggiungere l'etichetta ALFA, non può essere negativo in quanto maggiore di −32768 (ad occhio e croce stiamo sui 65500 e più byte all'indietro, senza voler fare i conti rigorosi...).

A pensarci bene, invece, sappiamo che un «Segmento» è lungo esattamente 65536 byte, considerati non come un vettore dotato di inizio (all'offset 0000H) e di fine (all'offset 0FFFFH), ma, come abbiamo detto più volte, come un elemento chiuso su se stesso, senza inizio né fine ciò che in termine tecnico si chiama una «coda circolare».

In questo tipo di struttura il byte successivo a quello avente offset 0FFFFH è sempre e solamente quello posto all'offset 0000H, mentre non è assolutamente vero che il segmento «finisca» all'offset «tutte-F» (come farebbe il povero microprocessore arrivato a tale offset? di certo non si ferma!!!).

Ecco che dunque, malgrado le apparenze, l'etichetta «ALFA» è posta **dopo** la chiamata alla stessa e perciò il displacement sarà **positivo** e pari, se non andiamo errati, a 13 (000DH).

Ecco che tutte le volte che il displacement supera i valori massimi consentiti (tanto positivo che negativo), la situazione viene ribaltata in modo da ottenere un displacement lecito: al solito noi programmatori non dobbiamo assolutamente preoccuparcene, dal momento che il tutto funziona perfettamente.

Dunque a prescindere da dove è posta l'etichetta a cui si vuole saltare, nel caso delle CALL «dirette» e «intrasegment» abbiamo visto che ha significato solo l'offset dell'etichetta, per mezzo del quale viene calcolato il displacement, ecco che nello stack verrà salvato solamente l'offset dell'istruzione successiva alla CALL stessa: al ritorno dalla subroutine (come vedremo analizzando le istruzioni RET), il valore estratto dallo stack sarà proprio l'offset dell'istruzione a cui si deve tornare.

Passando ora alle «direct inter-segment CALL», diciamo che si hanno quando l'istruzione di chiamata e l'etichetta di arrivo sono poste in due segmenti differenti: in particolare del «target» sarà significativo l'indirizzo completo dato dalla coppia offset-segment e come tale verrà riportata all'interno della codifica.

A

```
                NAME PROVA

CODE            SEGMENT
                ASSUME CS:COI

ALFA:           MOV AX,1234H
                RET
SOPRA:          NOP
                NOP

                ORG OFFFOH
START:          CALL ALFA
                JMP SOPRA
CODE            ENDS
                END START
```

Infatti subito dopo il byte che contraddistingue l'operazione in esame (il byte vale 9AH) si hanno altri 4 byte, due per l'offset ed altri due per il segment.

In questo caso non ha più senso sapere se l'istruzione a cui si salta è posta prima o dopo la CALL, in quanto si ha un salto fisico ad un indirizzo completo.

A differenza dunque dell'altro tipo di CALL dirette, in questo caso deve essere salvato nello stack anche il segmento da cui si è partiti, in quanto altrimenti il successivo ritorno avverrebbe in un posto totalmente sbagliato: ecco che infatti nello stack viene salvato dapprima il CS (Code Segment register) e subito dopo l'IP (Instruction Pointer).

Vedremo parlando delle RET che queste due informazioni verranno successivamente estratte per effettuare un ritorno corretto.

Le «indirect CALL»

Passiamo ora alle «indirect CALL» e cioè alle chiamate a subroutine delle quali non si specifica un'etichetta (magari perché non si è in grado), ma si sa che l'indirizzo a cui si dovrà saltare è contenuto in una variabile in memoria oppure in un registro della CPU.

Anche in questo caso possono essere di tipo «intra-segment» oppure «inter-segment» e cioè rispettivamente all'interno dello stesso segmento oppure verso un altro segmento completamente differente.

Analizziamo dapprima le «intrasegment».

In questo caso, già sappiamo, come succedeva per l'istruzione JMP, che nell'istruzione non comparirà più l'etichetta della routine che si vuole attivare, ma si troverà o una locazione di memoria oppure un registro, il cui contenuto è proprio il valore dell'offset dell'etichetta stessa.

Avremo perciò istruzioni del tipo

```
CALL BX
CALL DI
```

per quanto riguarda i registri, mentre nel caso delle locazioni di memoria sono lecite tutte le possibili combinazioni di indirizzamento: semplice, «basa-

```
B

     CALL ALFA      ;ALFA e' una variabile di tipo word
     CALL ALFA[BX]
     CALL ALFA[BX][DI+5]
     CALL WORD PTR BETA[SI] ;BETA e' definito come byte
     CALL WORD PTR [BX]
     CALL WORD PTR [SI]
     CALL WORD PTR CS:[BP][SI+24]
```

```
C

     CALL DOUBLE             ;DOUBLE e' una double-word
     CALL DOUBLE[SI-4]
     CALL DWORD PTR ALFA     ;ALFA e' un byte o una word
     CALL DWORD PTR BETA[BX][SI] ;idem per BETA
     CALL DWORD PTR [BX]
     CALL DWORD PTR ES:[BP-2][SI+55]
```

to», «indicizzato» o «indicizzato-basato» che già conosciamo molto bene.

L'unica accortezza che bisogna avere è far sì che la locazione contenente l'offset dell'etichetta a cui si salterà sia di tipo «WORD», come pure, nel caso di indirizzamento indiretto indicizzato senza specificazione di una cella di memoria, la locazione puntata deve essere una word.

Come esempio pratico vediamo alcune chiamate riportate in figura B.

La prima è semplice ed è il caso in cui ALFA (variabile di tipo word) contiene l'offset del target.

La seconda è sintomo dell'esistenza di una tabella di indirizzi di subroutine e BX permetterà di saltare alla BX-esima, secondo un meccanismo che ricorda l'istruzione «ON X GOSUB n,m,...» del Basic.

La terza è ancora riferita ad una tabella a cui si accede in base al valore di BX e di DI sommando ancora 5 (chissà a chi mai serviranno mostruosità del genere!).

La quarta è una chiamata alla subroutine il cui offset è posto in memoria all'indirizzo del byte BETA a cui deve essere sommato il contenuto di SI: siccome BETA era stato generato come byte (magari per motivi particolari) allora per effettuare una chiamata corretta bisogna farne precedere il nome dalle due parolette «WORD PTR» per mezzo delle quali l'istruzione si legge «effettua la CALL al contenuto della WORD puntata (PoinTeR) da... ecc.».

La quinta e la sesta sono due casi di indirizzamento indiretto basato (la quinta) ed indicizzato (la sesta) in cui l'offset dell'istruzione a cui si salta è contenuto nella cella di memoria il cui indirizzo è posto rispettivamente in BX ed in SI.

La settima invece dice che si deve saltare all'istruzione il cui offset è contenuto nella cella di memoria appartenente al «Code Segment» (c'è infatti il prefisso di override «CS:») il cui offset è a sua volta dato dalla somma del contenuto del registro BP e del registro SI incrementato di 24.

Con quest'ultimo esempio, vediamo che in genere, come vuole la regola, un riferimento ad una locazione di memoria prescrive che essa sia contenuta nel «Data Segment» e se non è così allora bisogna far precedere il nome della variabile dall'opportuno prefisso di «override».

In tutti questi casi di «indirect intrasegment CALL», prima di effettuare il salto all'etichetta il cui offset viene ottenuto per via indiretta, viene salvato nello stack l'offset della successiva istruzione, così come avveniva per le CALL dirette, in quanto si è sicuri che non si cambierà di segmento.

Per quanto riguarda le «indirect inter-segment CALL» va detto subito che il «target» in questo caso deve essere individuato con un indirizzo completo, formato dalla coppia offset-segment, che perciò deve essere contenuto in due word successive di memoria, ma non in registri, che in questo caso dovrebbero essere due.

Ecco che dunque in questo tipo di CALL potranno comparire: nomi di locazioni di memoria di tipo «doubleword» (definite come DD) più o meno indicizzate e basate, nomi di locazioni di memoria di tipo byte o word stavolta precedute dalle parolette «DWORD PTR» ed anche queste indicizzate e basate a seconda delle necessità, oppure infine locazioni di memoria (ancora intese come doubleword) il cui indirizzo è contenuto all'interno dei registri base e indice.

In tutti i casi possibili che vedremo, comunque, nella word meno significativa ci sarà l'offset, mentre in quella più significativa ci sarà il segment, entrambi ovviamente della locazione a cui si salterà.

Avremo perciò le possibilità riportate in figura C.

La prima si riferisce ad una locazione che è stata definita come doubleword e che già contiene l'offset ed il segment del target.

La seconda dice che l'offset ed il segment della cella a cui saltare sono contenuti nella doubleword posta in memoria 4 byte prima della locazione la cui label è DOUBLE a cui si somma il contenuto del registro SI.

La terza e la quarta rappresentano casi di locazioni già definite come BYTE o WORD e perciò devono essere precedute da «DWORD PTR»: nella quarta in particolare si ha ancora a che fare con un indirizzamento basato ed indicizzato.

La quinta rappresenta invece il caso in cui l'indirizzo della coppia di word, contenente l'offset ed il segment della locazione «target», è a sua volta contenuto nel registro BX.

La sesta infine, come al solito la più complicata, ci dice che l'indirizzo della locazione a cui salteremo è contenuto in una doubleword non già all'interno del «Data Segment», ma bensì all'interno dell'«Extra Segment» (vedasi l'override «ES:») il cui indirizzo è ottenuto sommando i contenuti del registro BP (diminuito di 2) e del registro SI (stavolta aumentato di 55).

Terminiamo dunque l'analisi delle CALL dicendo che in quest'ultimo caso di «inter-segment indirect CALL», prima dell'effettivo salto alla locazione desiderata, verrà effettuato il salvataggio nello stack sia dell'offset che del segment della locazione di memoria successiva alla CALL stessa, secondo il meccanismo già visto per le «direct inter-segment CALL».

Inutile dire che in nessuno dei casi visti di chiamata a subroutine per mezzo di un'istruzione CALL risulterà alterato alcun flag, come era ovviamente lecito aspettarsi. Inoltre c'è da dire che, a differenza di quanto si poteva trovare ad esempio nello Z80, non esistono nel caso dell'8086/88 le chiamate a subroutine condizionate allo stato dei flag (di solito il Carry e lo Zero): si possono però agevolmente simulare con un salto condizionato e successiva chiamata a subroutine, oppure si possono creare delle apposite «codemacro», secondo un meccanismo alquanto complesso che tralasciamo.

Le istruzioni di controllo - RET

Ecco l'istruzione che deve sempre andare in coppia con quella preceden-

temente analizzata, se non vogliamo che ci siano malfunzionamenti, oppure a meno che non siamo talmente tanto bravi ed esperti da poterla utilizzare anche non accompagnata dalla «fida» CALL.

In perfetta sincronia con i tipi di CALL, per quello che concerne l'appartenenza o no allo stesso segmento dell'istruzione RET e dell'istruzione a cui si tornerà, si hanno due tipi di RET, gli «intra-segment RET» e gli «inter-segment RET», non essendoci ovviamente la distinzione tra «diretto» ed «indiretto», che in questo caso non hanno alcun senso.

L'«intra-segment RET» è una delle poche istruzioni «tranquille», nel senso che, dal basso del suo semplice codice operativo pari a C3H, consente il ritorno da una chiamata «intra-segment» (sia diretta che indiretta) dal momento che estrae dallo stack un valore che provvederà subito a porre nell'Instruction Pointer (IP).

Ma si sa che nel mondo dell'informatica bisogna prevedere parecchie situazioni e cercare di orientarsi sempre più verso applicazioni più spinte: in un mondo (quello informatico) in cui si parla sempre di «task» e di programmazione in «multi-tasking», un'istruzione di RET così semplice non poteva in previsione essere sufficiente. In un ambiente multi-tasking, infatti ogni task si porta appresso delle informazioni (contenuti dei registri principalmente) che devono essere salvate da qualche parte (proprio nello stack!) all'atto dell'attivazione e della disattivazione: soprattutto il passaggio da un task all'altro comporta in genere un sovrautilizzo dello stack stesso, nel quale rimarrebbero delle locazioni inutilizzabili da altri processi.

Senza scendere in ulteriori dettagli, è molto comodo avere una istruzione di RETurn, che oltre a ripristinare l'indirizzo di ritorno nell'IP, in un certo senso scarti altre word poste nello stack, che altrimenti rimarrebbero lì inutilizzabili, ma soprattutto porterebbero via spazio nello stack stesso.

Ecco che dunque esiste l'istruzione

RET n

dove «n» NON è un numero di linea (come il Basic potrebbe farci trarre in inganno), ma è il numero di word che bisogna scartare dallo stack subito dopo aver ripristinato l'Instruction Pointer.

Questo scarto di «n» word avviene nel modo più semplice e naturale possibile, incrementando cioè di «n» il contenuto dell Stack Pointer (SP): in tal modo una successiva istruzione che ha a che fare con lo stack (una CALL, una PUSH, una POP o una RET o tante altre) farà sì che vengano utilizzate le word che una RET normale non avrebbe «liberato».

L'istruzione di RET «immediato» consta di tre byte, uno per il codice operativo (che vale C2H) e due per il valore immediato, che può perciò assumere valori anche molto grandi, però di pochissima utilità pratica se non in programmi che però diventerebbero molto critici da usare e da gestire.

Per quanto riguarda gli «inter-segment RET», abbiamo anche in questo caso l'istruzione «tranquilla», accanto a quella «orientata al futuro».

In particolare la prima è di nuovo una «RET» (indistinguibile da quella «intra-segment» almeno a livello mnemonico), che viceversa è codificata con il byte CBH e che effettua il ripristino dapprima dell'Instruction Pointer (IP) e poi del Code Segment register (CS), estraendo dunque dallo stack una coppia di word.

Torniamo però un istante al fatto della indistinguibilità delle due RET, «intra» e «inter»: in particolare sarà l'assemblatore a decidere di quale tipo si tratti a seconda che la procedura che con essa si chiude sia di tipo rispettivamente NEAR oppure FAR. Già ne abbiamo parlato nel numero 57 di MCmicrocomputer, allorché avevamo proposto una specie di mini-quiz.

Per quanto riguarda la «inter-segment RET» «immediata» anche in questo caso si ha un'istruzione del tipo

RET n

dove ancora una volta «n» può avere un qualsiasi valore rappresentabile con due byte e dove ovviamente il codice operativo è differente, dato dal valore CAH.

Concludendo, dunque, quando incontrerà quest'ultima istruzione, il microprocessore effettuerà nell'ordine il ripristino dell'Instruction Pointer (IP), del Code Segment (CS) ed in più incrementerà il contenuto dello Stack Pointer (SP) del valore immediato posto nell'istruzione.

Con questo abbiamo terminato l'analisi della chiamata di subroutine e del ritorno da esse, sia che siano poste nello stesso segmento del programma chiamante, sia che siano poste in un segmento completamente differente.

Nella prossima puntata proseguiremo nell'analisi delle istruzioni di controllo del flusso di elaborazione, parlando delle istruzioni di salto condizionato e, spazio permettendo, delle istruzioni di controllo dei cicli.

MC