ASSEMBLER 8086 8088

di Pierluigi Panunzi

I modi di indirizzamento

Siamo arrivati in questa puntata alla discussione di un argomento molto importante ed al tempo stesso molto interessante, quello dei modi di indirizzamento, che ritroveremo praticamente in ogni istruzione che fa riferimento ad un dato residente nella memoria oppure in uno dei registri.

In generale possiamo dire che l'assembler dell'86/88 ci fornisce parecchie differenti possibilità di indirizzamento di operandi, sia che l'istruzione sia ad un operando solo, sia che ne preveda due, nel quale ultimo caso prendono rispettivamente il nome di «destinazione» e di «sorgente».

Nel caso di istruzioni ad un unico operando non vi sono restrizioni, in generale, sul tipo di indirizzamento, sfruttabile con una certa istruzione, in quanto già sappiamo dalle scorse puntate che la caratteristica peculiare dell'Assembler dell'86/88 è la struttura a «matrice» che consente di usare quasi tutti i tipi di indirizzamento con quasi tutti i possibili registri.

Invece nel caso di istruzioni a due operandi si hanno alcune restrizioni, che, come vedremo, sono in un certo senso ovvie e cioè non «forzate».

Infatti in questo caso la «sorgente» (e cioè l'operando posto sulla destra, dopo la «,») può in generale essere una costante, altrimenti detta «valore immediato», inglobata nell'istruzione stessa: in tal caso la «destinazione» (l'operando posto a sinistra, prima della «,») non può ovviamente essere una costante, ma può essere o una locazione di memoria oppure un registro.

In caso contrario almeno uno dei due operandi deve essere un registro, mentre l'altro può essere o un altro registro oppure una locazione di memoria.

Come si può vedere dunque non è consentita un'operazione che abbia come due operandi locazioni di memoria (ma questo è in generale vietato da tutti i microprocessori), almeno in modo «esplicito»: vedremo infatti che esiste tutta una serie di istruzioni, dette «operazioni su blocchi di memoria», le quali operano in maniera «implicita» su coppie di locazioni di memoria.

In definitiva possiamo schematizzare tutte le possibili combinazioni nella tabellina seguente, in cui abbiamo indicato genericamente gli operandi con i termini «costante», «registro» e «memoria», senza preoccuparci per ora dei modi di indirizzamento.

Operandi per istruzioni a due operandi

Destinazione	Sorgente
registro	costante
memoria	costante
registro	registro
memoria	registro
registro	memoria

Analizziamo la prima possibilità: per quanto riguarda la costante, possiamo dire che si può trattare o di un valore a 16 bit (word) oppure ad 8 bit (byte) a seconda che il registro sia rispettivamente a 16 o ad 8 bit.

Già a questo livello di conoscenza possiamo anticipare il comportamento dell'Assembler nel caso si utilizzino costanti ad 8 bit con registri a 16 bit o viceversa costanti a 16 bit con registri ad 8 bit: nel primo caso tutto va bene in quanto il valore costante è solo «apparentemente» ad 8 bit in quanto può essere immediatamente convertito ad un valore a 16 bit. Nel secondo caso invece l'assemblatore genererà una segnalazione d'errore in quanto non si vuole assumere responsabilità sulla perdita di precisione nel troncare un valore a 16 bit in uno a solo 8.

Tra l'altro questa è la filosofia ispiratrice per l'Assembler in tutte le occasioni analoghe: in particolare l'Assembler dell'86/88 gode della fama di essere «strongly-typed» e cioè tenacemente legato al «tipo» del o degli operandi.

Per quanto riguarda il secondo caso valgono le stesse considerazioni del punto precedente con in più il vantaggio di una maggiore elasticità (però a costo di un leggero appesantimento formale delle istruzioni) nel trasferire quantità ad 8 o a 16 bit in celle di memoria che potranno considerarsi sia ad 8 che a 16 bit, indipendentemente dalla dichiarazione iniziale: come dire che la cella ALFA, definita come una word, potrà essere spezzata in due byte ed indirizzata conseguentemente, senza doversi impelagare in strani

trucchi, «alla Z80», tanto per fare un esempio.

Per il caso di registri come operandi vale l'ovvia regola che entrambi devono essere o ad 8 bit o a 16 bit, come già avevamo detto e come è più che naturale.

Per i due tipi restanti di coppie di operandi valgono considerazioni analoghe a quelle precedentemente riportate: ferma restando l'ovvia necessità di uguaglianza di «tipo» dei due operandi (ad 8 o a 16 bit), ritroveremo ancora la possibilità di gestire come byte una singola metà di una locazione di memoria di tipo word e viceversa come word una coppia di celle consecutive. Comunque nel prosieguo torneremo in dettaglio sul problema.

Vediamo dunque ora più da vicino la questione dei modi di indirizzamento delle locazioni di memoria.

Indirizzamento diretto

Il primo tipo di indirizzamento è quello più naturale e cioè quello diretto, secondo il quale una cella di memoria viene individuata dal suo nome simbolico, con il quale è stato pure definito il tipo della variabile: ad esempio con

```
ALFA    DW  ?
BETA    DB  ?
```

definiamo una variabile di tipo word (ALFA) ed una di tipo byte (BETA).

Possiamo perciò indirizzare l'una o l'altra variabile con il nome posseduto, ad esempio

```
MOV AX,ALFA
MOV BETA,10H
```

pongono rispettivamente in AX il contenuto della cella ALFA ed in BETA il valore immediato 10H.

In entrambi i casi il nome della variabile viene sostituito con l'offset della rispettiva cella, ovviamente riferito al Data Segment corrente, e tradotto con un valore appunto a 16 bit.

Se per caso volessimo puntare la decima cella di memoria a partire dalla cella di indirizzo ALFA, allora l'operando sarà semplicemente ALFA+10: bisogna fare però molta attenzione che il «displacement» (spostamento) che si aggiunge oppure sottrae all'etichetta indicata, è comunque espresso in byte e perciò tornando all'esempio di ALFA e BETA, quest'ultima può essere indirizzata (attenzione!) con

```
ALFA+2
```

in quanto solo sommando 2 all'offset di ALFA si ottiene quello di BETA.

Ovviamente in questo caso nessuno si sognerà di indirizzare BETA come displacement rispetto ad ALFA, in quanto BETA stessa possiede la propria etichetta; ben diverso è invece il discorso relativo ai vettori dei quali solo la prima cella possiede un'etichetta.

Ad esempio supponiamo di voler creare un'area di buffer chiamata BUFFER di 200 word. Otteniamo ciò con la direttiva

```
BUFFER DW 200 DUP(?)
```

Ora, per indirizzare il ventesimo elemento (word) di tale struttura, allora dobbiamo sommare, all'offset di BUFFER, 19 volte la lunghezza di ogni elemento (2 byte) e perciò dobbiamo scrivere ad esempio

```
MOV BUFFER+19*2,CX
```

dove è lecito indicare come displacement un'espressione, contenente perciò anche la moltiplicazione: tale espressione è evidentemente calcolata dall'assemblatore e fornirà un indirizzo «fisso», appunto del ventesimo elemento del BUFFER.

Indirizzamento indiretto

Questo particolare tipo di indirizzamento, che si ritrova in quasi tutti i microprocessori, anche ad 8 bit, assume nell'Assembler dell'86/88 una particolare importanza e distinzione in quanto può coinvolgere uno o addirittura due registri tra quattro possibili (BP,BX,SI,DI), secondo un meccanismo che prevede parecchie combinazioni.

In particolare in tale tipo di indirizzamento l'indirizzo di una certa locazione su cui agire non è esplicitamente posto nell'istruzione stessa sotto forma di offset (come succede invece nel caso dell'indirizzamento diretto), ma viceversa si trova all'interno di un certo registro, specificato nell'istruzione, racchiuso tra parentesi quadre.

A seconda che il registro in questione sia BX e BP da una parte oppure SI e DI dall'altra, si parla rispettivamente

```
*****           l'indirizzo della cella e' uguale a...
[BX]              contenuto del registro base BX
[BP]                  "       "     "     "   BP
[SI]                  "       "     "   indice SI
[DI]                  "       "     "     "    DI
[BX][SI]            cont. di BX + cont. di SI
[BX][DI]              "   "  "  +    "    "  DI
[BP][SI]              "   "  BP +    "    "  SI
[BP][DI]              "   "  "  +    "    "  DI
ALFA[BX]            offset di ALFA + cont. di BX
ALFA[BP]              "   "    "   +    "    "  BP
ALFA[SI]              "   "    "   +    "    "  SI
ALFA[DI]              "   "    "   +    "    "  DI
ALFA[BX][SI]      offset di ALFA + cont. di BX + cont. di SI
ALFA[BX][DI]        "   "    "   +    "    "  BX +    "    "  DI
ALFA[BP][SI]        "   "    "   +    "    "  BP +    "    "  SI
ALFA[BP][DI]        "   "    "   +    "    "  BP +    "    "  DI
 [BX+valore]      cont. di BX + valore (ad 8 o a 16 bit)
   ...            idem per BP, DI e SI

ALFA+valore[BX]  \     ...  + valore (ad 8 o a 16 bit)
ALFA[BX+valore]  /
   ...                idem per BP,SI e DI

ALFA+valore[BX][SI] \
ALFA[BX+valore][SI]  >  ... + valore (ad 8 o a 16 bit)
ALFA[BX][SI+valore] /
   ...                idem per BX+DI, BP+SI e BP+DI
```

Tabella 1

```
0000                    DATA SEGMENT
0000  ????               ALFA DW ?
0002  ????               BETA DW ?
0004                    DATA ENDS

0000                    CODE SEGMENT
                             ASSUME CS:CODE,DS:DATA
                             ASSUME ES:NOTHING,SS:NOTHING
0000  3E: 8B 86 0000 R       MOV AX,DS:ALFA[BP]
0005  3E: 8B 86 0000 R       MOV AX,ALFA[BP]
000A  8B 86 0000 R           MOV AX,SS:ALFA[BP]
000E  3E: 8B 46 00           MOV AX,DS:[BP]
0012  8B 46 00               MOV AX,[BP]
0015  8B 46 00               MOV AX,SS:[BP]
0018                    CODE ENDS

                             END
```

Listato 1

di «indirizzamento indiretto attraverso un registro base» oppure «attraverso un registro indice»: in tal modo abbiamo a disposizione quattro tipi di indirizzamento indiretto.

Non contenti di ciò possiamo ottenere l'indirizzo della locazione desiderata come somma del contenuto di un registro base con il contenuto di un registro indice, il che comporta la possibilità di altre quattro combinazioni.

Infine per ognuna di queste otto combinazioni sin qui ottenute si può aggiungere nell'istruzione stessa un offset, sia sotto forma del nome di una cella di memoria, sia (anche contemporaneamente alla precedente) sotto forma di un valore positivo o negativo ad 8 o a 16 bit: questo fatto comporta da solo una proliferazione di nuove possibilità di indirizzamento indiretto.

Ma dopo tanta teoria andiamo a vedere, sotto forma di tabella, quali sono tutte le possibilità che ci offre l'Assembler dell'86/88, utilissime quando si ha a che fare con tabelle o addirittura matrici.

Supponiamo dunque di voler caricare il registro AX con il contenuto di una certa coppia di locazioni (dal momento che AX come sappiamo è a 16 bit), che raggiungeremo con tutti i possibili tipi di indirizzamento indiretto: la nostra istruzione sarà genericamente una

MOV AX,•••••

dove ovviamente al posto di «*****» potremmo porre quanto indicato in tabella 1; nella seconda colonna della tabella troveremo indicato in che modo viene calcolato l'indirizzo di una certa cella.

In particolare diciamo che gli operandi racchiusi tra « > » producono un effetto perfettamente identico, come dire che il «valore» ad 8 o a 16 bit può essere aggiunto indifferentemente (e perciò a seconda delle circostanze) sia all'etichetta, che all'interno della prima che della seconda parentesi quadra, essendo tra l'altro consentite tutte le possibili combinazioni di valori aggiunti alle tre entità.

In particolare un operando del tipo di

ALFA + 5[BX + 7][DI − 2]

è perfettamente lecito e produce un codice perfettamente identico a quello delle seguenti parti di istruzioni:

ALFA[BX + 10][SI]
ALFA + 100[BX][SI − 90]
ecc.

nelle quali il valore da aggiungere è sempre dato da «offset di ALFA + 10» (comunque lo si scriva!).

Questioni di segmenti...

Ritorniamo un istante sulla questione del segmento associato per default ad un certo registro, questione che avevamo lasciato in sospeso e sulla quale ritorneremo in parecchie altre occasioni.

Sappiamo oramai, per averlo ripetuto parecchie volte, che per quanto riguarda i registri usati per l'indirizzamento indiretto, BX, SI e DI fanno riferimento per default al Data Segment e cioè l'indirizzo di una certa locazione di memoria, calcolato differentemente a seconda del tipo di operandi dell'istruzione, rappresenta l'offset nell'ambito del Data Segment, intendendo con ciò che il segmento di appartenenza della locazione di memoria interessata è contenuto nel registro DS.

Viceversa sappiamo che il registro BP fa riferimento per default allo Stack Segment: questo, nel caso in cui il tipo di indirizzamento preveda semplicemente l'operando «[BP]» senza il nome di una locazione di memoria in quanto in quest'ultimo caso quello che prevale è il Data Segment.

Questo si può vedere nel piccolo programma di prova riportato nel listato 1: in esso abbiamo riportato più esempi di istruzioni coinvolgenti la variabile ALFA ed il registro BP.

In particolare si può notare che la prima, la seconda e la quarta istruzione del mini-programma hanno il primo byte pari a 3E, rappresentato in evidenza dall'assemblatore grazie all'uso dei «:» (si tratta del MASM), ed indicante un «segment override».

In poche parole nelle tre istruzioni indicate, siccome si fa riferimento sia ad ALFA (appartenente al Data Segment), sia a BP (che per default fa riferimento allo Stack Segment), l'assemblatore è costretto a forzare il Data Segment con il prefisso 3E in quanto altrimenti la variabile ALFA non potrebbe essere raggiunta nell'ambito dello Stack Segment: il valore contenuto nel registro BP va dunque a sommarsi all'offset di ALFA per fornire un offset relativo al Data Segment, appunto in virtù del prefisso.

La terza istruzione invece contiene un «segment override» da parte del programmatore, che perciò vuole forzare l'indirizzo ottenuto all'interno dello Stack Segment: in questo caso l'assemblatore non ha bisogno di aggiungere in testa al codice dell'istruzione alcun prefisso di override in quanto in questo caso sarà proprio il default dell'istruzione (Stack Segment) a prevalere.

Nelle ultime due istruzioni non compare alcun «override», a livello di esplosione in Assembler da parte del MASM in quanto in entrambi i casi l'assemblatore prende in considerazione il segmento di default e cioè lo Stack Segment: a tale proposito il prefisso «SS:» posto nell'ultima istruzione è ridondante e perciò inutile, come lo era il prefisso «DS:» nella prima istruzione e l'«SS:» della terza.

Concludiamo questa puntata tornando un istante sulla terza istruzione: in questo caso il programmatore deve essere ben conscio del fatto che il prefisso «SS:» farà indirizzare ad una cella dello Stack Segment quando viceversa vuole usare una cella appartenente al Data Segment.

Con tutta probabilità è proprio quello che voleva il programmatore, in quanto, nel caso in cui ALFA sia un vettore, non si indirizzerebbe assolutamente un suo elemento, ma una certa cella dello Stack.

Evidentemente il tutto deve essere maneggiato con le debite cautele, anche se tutto sommato siamo in presenza di un caso limite.

In tutti i casi comunque un'analisi della codifica in Assembler può fornire più informazioni di quelle indicate nell'istruzione stessa.

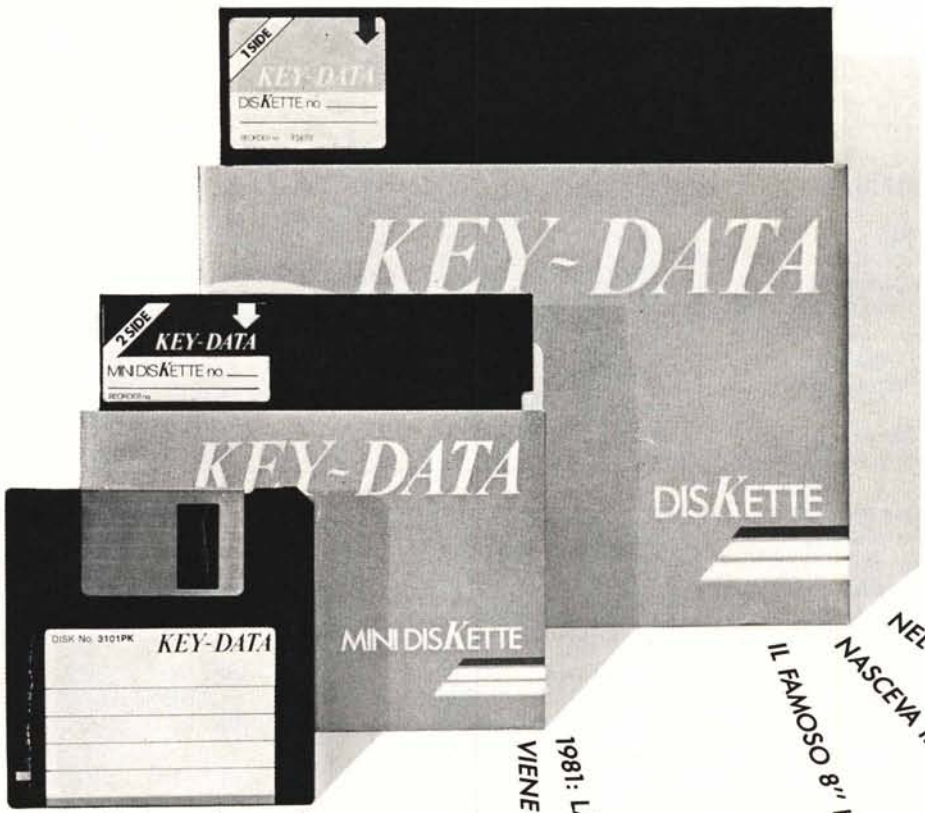
MC

gi erre informatica

presenta

K KEY-DATA

L'EVOLUZIONE CHE GIRA NEL TEMPO



NEL LONTANO 1980 NASCEVA IL PRIMO FLOPPY-DISC

IL FAMOSO 8" KEY-DATA

1981: LA RIVOLUZIONE DEI PERSONAL COMPUTERS VIENE INTRODOTTO SUL MERCATO IL 5"¼ KEY-DATA

1984: CONTINUA L'EVOLUZIONE

NASCE IL PICCOLO MICROFLOPPY

IL 3,5" KEY-DATA

I MINIDISKETTE 5"¼ KEY-DATA SONO DISPONIBILI IN VARIE COLORAZIONI

GI-ERRE INFORMATICA s.r.l.
42100 REGGIO EMILIA VIA UMBRIA 36/A TEL. 0522 38655 • 512345
70125 BARI VIA MONTE S. MICHELE 2/B TEL. 080 415975
95100 CATANIA P.ZZA GALATEA 2 TEL. 095 375222