ASSEMBLER 8086 8088

di Pierluigi Panunzi

Il set di istruzioni

Istruzioni di stringa

prima parte

In questa puntata parleremo di cosa si intende, nell'assembler dell'8086/88, con il termine di stringa ed analizzeremo le istruzioni di gestione delle stringhe. Vedremo che con tale termine non si intendono necessariamente insiemi di caratteri ASCII, così come siamo abituati lavorando con linguaggi ad alto livello.

Le «stringhe»

Tale termine, che come detto può ingenerare confusione se paragoniamo il suo significato con quello assegnatogli da linguaggi ad alto livello (Basic e Pascal in testa), rappresenta innanzitutto un generico blocco di dati, siano essi byte o word, sui quali possiamo effettuare un certo numero di operazioni «primitive», intendendo con quest'ultimo termine il fatto che si tratta di operazioni «basilari», «standard» ed in un certo senso «irrinunciabili» sulle quali eventualmente costruire operazioni più complesse.

Tali operazioni sulle stringhe sono paragonabili alle quattro operazioni sui numeri, le quali sono anch'esse delle operazioni «primitive»: come per calcolare un integrale utilizziamo una opportuna sequenza di operazioni primitive su operandi «numero» così avremo operazioni più complesse su stringhe, ad esempio la sostituzione di tutte le occorrenze di certi dati all'interno di un dato blocco con altri dati di un altro blocco.

Proprio quest'ultimo esempio può mostrare che le «stringhe-ad-alto-livello» (quelle di caratteri ASCII, tanto per intenderci) sono solo un caso particolare del più generico «blocco di dati elementari»: l'esempio infatti può benissimo riferirsi alla sostituzione, all'interno di un testo, di tutte le occorrenze di un certo vocabolo con un altro, operazione che è fondamentale in un qualunque word processor.

Tra parentesi il termine «stringa», di uso ormai corrente, è una brutta italianizzazione del termine inglese, «string»: con tale termine gli anglofoni intendono (oltre ad una ventina di altri significati, come loro solito) sia il termine «stringa» (inteso come laccio da scarpa), sia il termine «fila» (ad esempio di perle in una collana), sia appunto l'atto di infilare le perle per creare una collana. Riteniamo che gli ultimi due significati siano più vicini alla realtà rappresentata da un certo numero di caratteri o dati generici che siano, uno posto di seguito all'altro (quasi in fila indiana!), tanto è vero che ne possiamo eliminare da un certo punto in poi come estrarre uno ad uno: in questo senso non vediamo come un laccio da scarpa (che è il significato che diamo noi alla parola «stringa», anche se francamente non l'usiamo poi spesso) possa avere qualcosa a che fare con un insieme di caratteri o dati.

Comunque il termine è ormai talmente radicato nella terminologia corrente che non ci ricordiamo più del suo significato vero...

Chiusa dunque questa parentesi eti-

mologica, torniamo alle «stringhe-8086»: un singolo byte, tre word consecutive, un intero segmento di 64K byte sono tre esempi di stringhe sulle quali possiamo appunto eseguire certe operazioni.

Non importa il «contenuto» della o delle celle di memoria che costituiscono la stringa, ma ci interessa appunto l'insieme di dati nella sua interezza, insieme sul quale e con il quale possiamo compiere operazioni di spostamento di comparazione, di analisi, di caricamento e di lettura.

Particolarità dell'assembler 8086/88 è quella di avere delle operazioni primitive al massimo, che agiscono sul singolo elemento dell'insieme considerato, ma che possono essere agevolmente estese ad un insieme molto grande di singoli elementi.

Entrando più nei dettagli abbiamo visto già che possiamo lavorare su elementi dati da byte o word, posti in memoria: ora a seconda se l'operazione lo richiede, si avrà in alcuni casi un blocco di dati «sorgente» ed un blocco di dati «destinazione».

Ancor più esattamente potremo avere un byte o una word all'interno del gruppo sorgente ed un byte o una word all'interno del gruppo destinazione: intuitivamente gli spostamenti e le comparazioni avranno un'entità sorgente ed una destinazione, le analisi ed i caricamenti avranno solo entità destinazioni mentre infine le letture avranno solamente le entità sorgenti.

Tutte le volte che parliamo di sorgenti e di destinazioni dobbiamo tener presente che ci riferiamo a locazioni di memoria, che come tali posseggono un indirizzo rappresentato dalla solita coppia «segment:offset»: nel caso di byte o word sorgenti, si «punterà», all'entità per mezzo del registro SI, intendendo per default che il dato si trova nel Data Segment (DS) corrente, mentre viceversa (e qui si scopre una prima applicazione dell'Extra Segment) nel caso di byte o word destinazione si punterà a tale dato per mezzo del registro DI (per quanto riguarda l'offset) e per mezzo dell'Extra Segment (ES appunto).

Bisogna sempre tenere bene a mente quest'ultima particolarità, secondo la quale il dato destinazione si trova all'indirizzo formato da «ES:DI», contrapposto al dato sorgente che si trova in «(DS:)SI», dove appunto il «DS» è stato messo tra parentesi in quanto di default.

Inutile dire che una dimenticanza in tal merito comporta non già un errore sintattico o di programma, ma bensì un malfunzionamento all'atto dell'esecuzione del nostro programma, che non funzionerà come vogliamo (ma girerà comunque!)

Detto dunque questo, c'è da aggiungere il fatto che la singola operazione primitiva sul singolo dato potrà essere ripetuta per un certo numero prefissato di volte: dal momento che si ha a che fare con strutture formate da più byte o word, nasce allora l'esigenza di aggiornare il puntatore, formato nel primo caso da SI e nel secondo caso da ES:DI.

In entrambi i casi l'assembler ci consente un incremento o un decremento automatico (a nostra scelta) del puntatore, senza dover materialmente utilizzare un'istruzione di INC o di DEC, ma sfruttando un particolare flag detto di «direzione» (Direction Flag, DF), che con il suo stato indicherà all'istruzione di stringa se il o i puntatori dovranno essere incrementati o decrementati.

Altra particolarità (e poi inizieremo l'analisi vera e propria delle istruzioni) è che sarà l'istruzione stessa ad indicare se si deve operare su byte o word e perciò decidere se l'incremento o il decremento del puntatore deve essere di un'unità (nel caso del byte) o di due unità (nel caso di word): se infatti lavoriamo con word è ovvio che il puntatore deve avanzare o indietreggiare di due ogni volta altrimenti si andrebbe a puntare al byte più significativo di una word.

Le istruzioni di stringa-STOS

Iniziamo dunque dall'istruzione più semplice, la STOS, che permette di memorizzare il contenuto dell'accumulatore nella cella puntata da ES:DI e cioè in una cella della nostra stringa.

In particolare avremo le due istruzioni STOSB e STOSW se rispettivamente il dato da inizializzare sarà un byte o una word (è questa una regola generale: la «B» finale indicherà un'operazione su entità byte, mentre la «W» si riferirà ad entità word): nel caso della STOSB sarà il contenuto di AL a finire nel byte destinazione mentre è intuitivo che nel caso della STOSW sarà il contenuto di AX a finire nella word posta in ES:DI.

Nel primo caso si avrà l'incremento o il decremento di DI di un'unità mentre nel caso della STOSW il registro DI aumenterà o diminuirà di due unità.

Bisogna dunque ricordarsi di settare opportunamente il flag di direzione, con le due istruzioni di cui parleremo nel prossimo paragrafo.

Le due istruzioni qui viste (STOSB e STOSW) non necessitano di operandi in quanto l'assemblatore già presuppone che i registri siano stati caricati correttamente (in caso contrario poi il programma non ci darà risultati corretti, pur essendo esatto sintatticamente...) e si può riassumere il loro comportamento con i due schemi seguenti:

```
          Istruzione STOSB
          ----------------

ES:DI  <--  AL

if DF = 0
 then DI  <--  DI + 1
 else DI  <--  DI - 1


          Istruzione STOSW
          ----------------

ES:DI  <--  AX

if DF = 0
 then DI  <--  DI + 2
 else DI  <--  DI - 2
```

A

Aggiungiamo che, come ogni istruzione di caricamento che si rispetti, i flag non vengono alterati in alcun modo, neanche se ad esempio il valore da porre nella stringa è proprio 0. Analizziamo ora un esempio nel quale supponiamo di voler inizializzare un byte (posto alla locazione VETTORE), con il valore 55H.

Per poter sfruttare la STOSB, dob-

```
DATA    SEGMENT
ALFA    DB ?
VETTORE DB ?
        ...
DATA    ENDS

CODE    SEGMENT
        ...
        PUSH DS
        POP ES    ;trucco per caricare ES con
                  ;il contenuo di DS
        ASSUME ES:DATA
        MOV DI,OFFSET VETTORE
        MOV AL,55H
        STOSB
        ...
CODE    ENDS
```

B

```
SEGMENTO  SEGMENT
INIZIO    DW 32768 DUP (?)
SEGMENTO  ENDS

CODE      SEGMENT
          ...
          PUSH DS
          POP ES    ;come prima...
          ASSUME ES:SEGMENTO
          MOV DI,OFFSET INIZIO
          MOV CX,8000H  ;sono 32k word
          CLD       ;vogliamo incrementare DI
          XOR AX,AX  ;azzeramento dell'accumulatore
          REP STOSW  ;memorizzazione ripetuta
          ...
CODE      ENDS
```

C

biamo innanzitutto inizializzare l'Extra Segment (ES), in modo da farlo puntare al segmento che noi vogliamo (supponiamo che VETTORE si trovi nel Data Segment corrente), poi dobbiamo caricare il registro DI con l'offset della locazione di memoria in questione poi dobbiamo inizializzare l'accumulatore ed infine... dobbiamo usare la STOSB.

Un esempio di frammento di programma può essere l'esempio B.

I lettori più attenti potrebbero già brontolare notando che il tutto si poteva ottenere con una sola istruzione senza scomodare nell'ordine l'Extra Segment (ES), il registro DI, l'accumulatore (AL) e.. la STOSB: basta infatti scrivere l'istruzione:

MOV VETTORE, 55H

per ottenere esattamente la stessa cosa!

Evidentemente abbiamo fatto questo esempio solo per introdurre il concetto di «ripetizione» di un'istruzione di stringa.

Come vedremo infatti nel seguito analizzando in dettaglio il «prefisso» REP, le due istruzioni in esame possono essere ripetute per un numero di volte impostato nel registro CX, ed è inutile dire che è qui che compare tutta la potenza di questa istruzione di stringa (e anche delle altre, come vedremo).

Supponiamo dunque di voler azzerare non più un byte, ma ben 64K e cioè addirittura un intero segmento, che supponiamo chiamarsi «SEGMENTO»: in questo caso dovremo aggiungere a quanto fatto in precedenza l'inizializzazione del registro CX ed il settaggio del flag di direzione.

Inoltre possiamo usare l'istruzione STOSW che ci permette di azzerare una word alla volta fatto che consente di dimezzare il valore contenuto in CX e contemporaneamente di effettuare l'azzeramento in metà tempo.

Nell'esempio C dimostriamo come possiamo azzerare un intero segmento.

Non vogliamo usare la STOSW, ma vogliamo eseguire il programma con le istruzioni normali? Presto detto: lo stesso programma si può scrivere nel modo seguente:

```
SEGMENTO  SEGMENT
INIZIO    DW 32768 DUP (?)
SEGMENTO  ENDS

CODE      SEGMENT
          ...
          MOV DI,OFFSET INIZIO
          MOV CX,8000H  ;sono 32k word
SU:       MOV WORD PTR [DI],0
          INC DI
          INC DI
          DEC CX
          JNZ SU
          ...
CODE      ENDS
```

D

In questo caso non si scomodano altri registri, all'infuori di CX e DI, ma il rovescio della medaglia è nel tempo di esecuzione che è quasi il doppio che nel caso della STOSW: mentre infatti nel primo caso c'è solo la REP STOSW che viene eseguita ogni iterata e perciò per 32768 volte, nel secondo caso è un pacchetto di istruzioni che viene eseguito per ogni iterata.

Tanto per addentrarci in un campo non ancora affrontato, cerchiamo di quantizzare in «soldoni» le durate dei due cicli magari in termini di un clock di 4.77 MHz, tipico di un PC IBM.

Nel primo caso ogni istruzione «REP STOSW» richiede 6 + 10 cicli di clock per ogni iterata (6 per la REP e 10 per la STOSW) e perciò un totale di 16*32768 = 524288 cicli, che a 4.77 MHz significano circa 0.11 secondi! Pensate! un decimo di secondo per azzerare 64K byte di RAM... Un secondo per azzerare tutta la RAM (640K) del nostro PC! (Ma allora perché il PC ci mette così tanto all'accensione? Non è questa la sede adatta: ne parleremo nella rubrica apposita. N.d.r.).

Il secondo programmino invece richiede 15 cicli solo per l'istruzione di MOV, altri due cicli ne richiedono le singole INC e DEC (invece di INC DI - INC DI potevamo usare ADD DI,2 che impiega sempre 4 cicli), mentre infine la JNZ impiega ogni volta 8 cicli (nel caso in cui la condizione NZ è verificata e nel nostro caso 32767 volte!) contro i 4 cicli dell'ultima iterata in cui non deve effettuare il salto a ritroso: in totale abbiamo (15+2+2+2) * 32768+8 * 32767+4 cicli pari alla bellezza di 950268 cicli con durata di circa 0.2 secondi, come dire quasi il doppio del caso precedente...

Le istruzioni CLD e STD

Sono due istruzioncine semplici semplici che rispettivamente azzerano (CLD, «CLear Direction flag») e settano (STD, «SeT Direction flag»), il flag di direzione (DF), nel primo caso permettendo l'incremento automatico di uno o entrambi i registri puntatori DI e SI (a seconda di quale dei due sia richiesto dall'istruzione di stringa) e nel secondo caso abilitando il decremento automatico di DI e/o SI di una o due unità.

Volete una regoletta facile facile inventata dal redattore della rubrica?

Per ricordarsi mnemonicamente se il flag DF posto a «0» indichi un incremento o un decremento (non utilizzando molto spesso istruzioni di stringa capita di dimenticarsi questi piccoli particolari), invece di andare a sfogliare manuali (o la rivista...), allora si può aggirare l'ostacolo pensando mentalmente alla lettera «D» di CLD e STD non più relativa al «Direction flag» (la qual cosa appunto non ci è di minimo aiuto in caso di dimenticanza), ma piuttosto possiamo associare la «D» alla parola «Decrement» e così la CLD si può leggere come «CLear Decrement» (cancella il decremento e perciò attiva l'incremento...), mentre la STD si può leggere come «SeT Decrement» (setta il decremento...): semplice ma efficace, non c'è che provare!

L'istruzione LODS

Questa istruzione è in un certo senso la «duale» della STOS già analizzata, in quanto consente di caricare nell'accumulatore (AL o AX) il contenuto di una cella di memoria appartenente ad una stringa. In questo caso la cella è individuata a livello «offset» dal registro SI (che ricordiamo essere

```
  Istruzione LODSB
  ----------------

AL  <--  DS:SI

if DF = 0
 then SI  <--  SI + 1
 else SI  <--  SI - 1

E
```

```
  Istruzione MOVSB
  ----------------

ES:DI  <--  DS:SI

if DF = 0
 then SI  <--  SI + 1 ; DI  <--  DI + 1
 else SI  <--  SI - 1 ; DI  <--  DI - 1
```

```
  Istruzione LODSW
  ----------------

AX  <--  DS:SI

if DF = 0
 then SI  <--  SI + 2
 else SI  <--  SI - 2
```

```
  Istruzione MOVSW
  ----------------

ES:DI  <--  DS:SI

if DF = 0
 then SI  <--  SI + 2 ; DI  <--  DI + 2
 else SI  <--  SI - 2 ; DI  <--  DI - 2

F
```

```
PARTENZA  SEGMENT
          ...
          ORG 1234H  ;offset non nullo
START     DB 1000 DUP(?)
          ...
PARTENZA  ENDS

ARRIVO    SEGMENT
          ...
          ORG 5678H  ;altro offset non nullo
BEGIN     DB 1000 DUP(?)
          ...
ARRIVO    ENDS

CODE      SEGMENT
          ASSUME CS:CODE,DS:PARTENZA
          ...   ;caricamento del blocco di 1000 byte
          ...
          MOV AX,ARRIVO
          MOV ES,AX
          ASSUME ES:ARRIVO
          MOV SI,OFFSET START
          MOV DI,OFFSET BEGIN
          CLD
          MOV CX,1000
          REP MOVSB
          ...
G►  CODE  ENDS
```

il «Source Index») e come segment per default da DS e perciò dal segmento corrente contenente in genere i dati del programma. Analogamente alla STOS, la LODS ha insito nel suo funzionamento, l'aggiornamento automatico del puntatore SI, il quale può essere incrementato o decrementato a seconda dello stato del flag DF.

Anche in questo caso si hanno due istruzioni distinte (LODSB e LODSW), rispettivamente riferite ad un byte e ad una word, il cui contenuto viene posto rispettivamente in AL e AX.

Anche in questo caso possiamo vedere quali sono le operazioni che il microprocessore compie all'atto dell'esecuzione: in particolare si ha quanto si vede nell'esempio E.

Dal momento che le istruzioni in esame non fanno altro che caricare l'accumulatore con un dato contenuto in memoria, nel 99% dei casi non ha alcun senso effettuare la «ripetizione» dell'istruzione stessa per mezzo dell'istruzione (o meglio «prefisso») «REP» in quanto non ha molto senso caricare in accumulatore un valore e nell'istruzione dopo caricarne un altro: nell'1% dei casi in cui si ha necessità di usare la «REP LODSB» (casi che effettivamente esistono, ma sui quali non è interessante soffermarci in questo contesto) basta sapere che la coppia di istruzioni funziona egregiamente, come dire che è permesso ripetere anche le istruzioni LODS.

Come tutte le istruzioni di caricamento di registri, anche le due LODS non alterano in alcun modo i flag.

L'istruzione MOVS - come spostare blocchi di memoria

Siamo arrivati dunque alla MOVS, la quale è in un certo senso l'unione delle due istruzioni di stringa viste precedentemente ed è un'istruzione molto potente in quanto, unica nel suo genere consente di effettuare spostamenti «da memoria a memoria», cosa che finora, con la semplice «MOV» non era possibile fare.

In particolare anche in questo caso si può avere a che fare con dati sotto forma di byte o di word (e questo sarà deciso dalla presenza rispettivamente della lettera «B» o della lettera «W» nel nome dell'istruzione, che dunque si chiamerà MOVSB e MOVSW) e combinando la potenza di questa istruzione, che agisce sul singolo byte o sulla singola word, con la ripetibilità data dal prefisso REP, si ottiene una super-istruzione che effettua lo spostamento di blocchi di memoria da un certo indirizzo ad un altro.

In questo caso il byte o la word sorgente sarà indirizzata fisicamente dalla coppia formata da DS (di default come per la LODS), e da SI (come offset), mentre il byte o la word di destinazione avranno un indirizzo fisico che (è facilmente intuibile) è dato dalla coppia formata da ES (l'Extra Segment, come per la STOS) e da DI (il «Destination Index»). Ancora una volta si ha l'automatismo nell'incremento o decremento di entrambi i registri indice.

Possiamo infatti vedere nelle due tabelline di figura F il comportamento dell'istruzione nel caso di movimento di byte e di word.

In parole povere, l'istruzione MOVS trasferisce il dato puntato da DS:SI nella locazione puntata da ES:DI, aggiornando subito dopo i due puntatori a seconda dello stato del «Direction (o Decrement...) Flag».

Vogliamo ad esempio spostare 1000 byte, contenuti in un segmento, in un altro segmento, non solo «formalmente» (come sarebbe più semplice fare a livello assemblatore), ma soprattutto fisicamente.

Supponiamo di avere due segmenti di dati, chiamati rispettivamente PARTENZA e ARRIVO, contenenti l'uno i 1000 byte (riempiti non ci interessa come) e l'altro i byte «vuoti»: il blocco inizia all'etichetta START avente un certo offset (non necessariamente nullo) e deve essere spostato all'etichetta BEGIN, anch'essa dotata di offset non nullo, appartenente al segmento ARRIVO.

Vediamo dunque come è facile effettuare lo spostamento, per mezzo del frammento di programma G.

Da notare, in questo programma, che i registri SI e DI sono caricati rispettivamente con l'offset dell'etichetta START e dell'etichetta BEGIN per mezzo dell'istruzione MOV, anche se START e BEGIN appartengono l'uno al Data Segment e l'altro all'Extra Segment.

C'è da dire che per velocizzare l'operazione (in pratica si dimezza il tempo!) conviene caricare il valore 500 nel registro CX e pensare di trasferire word anziché byte per mezzo dell'istruzione MOVSW: l'assembler in questo caso non genera errori di sorta in quanto si «fida» (grazie alla lettera «W») che tanto la sorgente quanto la destinazione sono dello stesso tipo e cioè word.

Concludiamo questa prima parte dicendo che anche per l'istruzione di spostamento blocchi (o stringhe, che dir si voglia) MOVS vale la constatazione che non c'è nessuna ragione per cui i flag possano essere alterati: infatti il nostro buon microprocessore si guarda bene di effettuare scriteriate alterazioni dei poveri flag.

MC