ASSEMBLER 8086 8088

di Pierluigi Panunzi

Registri - Flag Indirizzamento

Dopo aver visto nelle prime puntate di questa rubrica le caratteristiche interne dei due microprocessori in esame e dopo aver introdotto il concetto fondamentale della programmazione in 8086/8088 e cioè la «segmentazione della memoria», in questa puntata inizieremo a vedere più da vicino quelle che sono le caratteristiche in un certo senso comuni a qualsiasi microprocessore.

Ci stiamo riferendo ai registri in generale, ai flag ed ai vari modi di indirizzamento, tre caratteristiche che differenziano ancora di più un microprocessore da un altro.

Prima di iniziare l'analisi, teniamo a sottolineare un aspetto fondamentale dei due microprocessori in esame, e che abbiamo più volte ricordato: salvo rarissime ed esplicite indicazioni, da questo momento in poi e cioè per tutto quello che riguarda la programmazione i due microprocessori 8086 ed 8088 sono perfettamente identici e perciò compatibili (veramente al 100%!): tutto quanto diremo per un micro vale per l'altro. Per questo motivo d'ora in poi ci riferiremo all'uno e all'altro con il generico termine «86/88», che ogni volta ci ricorderà la perfetta identità di programmazione.

I registri generali

Appartengono a questo insieme i già più volte citati AX, BX, CX e DX i quali possono partecipare senza alcuna limitazione a tutte le operazioni logico-algebriche previste dall'86/88, sotto forma di registri a 16 bit (AX, BX, CX e DX) oppure come registri ad 8 bit (AL e AH da AX, BL e BH da BX, CL e CH da CX ed infine DL e DH da DX): non vi sono limitazioni, nel senso che una certa operazione che può essere eseguita tra due registri qualunque può essere effettuata tra altri due registri secondo tutte le possibili combinazioni. Evidentemente non si potranno usare nella stessa istruzione due registri, uno a 16 bit e l'altro ad 8: non avrebbe senso infatti effettuare l'AND tra un dato ad 8 bit ed uno a 16 bit.

Alcune altre istruzioni, che vedremo in dettaglio nel seguito, invece utilizzano solo alcuni dei registri in modo ben preciso (ad esempio la moltiplicazione, gli shift multipli e le operazioni su blocchi di dati).

In generale in questi ultimi casi i registri in questione assumono un nome mnemonico che aiuta a comprenderne la funzione; in particolare si ha la seguente corrispondenza:

Registro	Funzione
AX	Accumulatore (Accumulator)
BX	Base (Base)
CX	Contatore (Counter)
DX	Dato (Data)

Per quanto riguarda l'accumulatore si ha un'altra caratteristica che lo contraddistingue dagli altri tre registri, caratteristica che se vogliamo è un «retaggio» dell'8088 e dell'8085: in particolare si ottengono istruzioni più brevi in termine di numero di byte se si usa l'accumulatore come destinazione di un calcolo logico o aritmetico. Tra l'altro, ma è di ben poco ausilio, in questo caso anche l'istruzione dal punto di vista mnemonico può essere abbreviata: ad esempio un'istruzione:

```
MOV AX,PIPPO
```

che, a seconda se PIPPO è una costante oppure l'indirizzo di una cella di memoria, pone in AX il valore della costante PIPPO oppure il contenuto della cella di memoria di etichetta PIPPO, dicevamo, questa istruzione può essere abbreviata con:

```
MOV ,PIPPO
```

sottintendendo così l'uso dell'accumulatore nell'istruzione: non ce la sentiamo però di affermare che ciò comporti un aiuto per il programmatore, specie se alle prime armi, in quanto si ottiene un'istruzione praticamente illeggibile.

I restanti quattro registri sono esclusivamente a 16 bit, indipendentemente dal fatto che o la parte alta o la parte bassa non vengano utilizzate, ad esempio perché nulle.

I Registri Puntatori ed Indice

Appartengono, a questo gruppo di registri «irriducibili» a 16 bit, i registri SP, BP, SI e DI ed hanno la caratteristica di contenere in generale la base o l'offset usati nel calcolo dell'indirizzo fisico di una certa locazione di memoria nell'ambito di un certo segmento.

Analogamente ai registri precedenti i registri Puntatori ed Indice possono prendere parte a tutte le operazioni logico-aritmetiche a 16 bit (avete mai fatto l'OR dello Stack Pointer con il contenuto di una locazione di memoria? Lo raccomandiamo ai programmatori con tendenze suicide...).

In particolare ha senso invece effettuare operazioni su due «Index register» SI e DI in quanto partecipano in maniera massiccia ad operazioni su vettori, tabelle o matrici, nonché su stringhe. La differenza fondamentale tra i registri BP ed SP da un lato e SI e DI dall'altro consiste nel fatto che i primi due, quando contengono un offset di una certa locazione, la considerano appartenente allo Stack Segment attuale, mentre viceversa offset contenuti in SI e DI vengono abbinati, nel calcolo dell'indirizzo fisico, al valore corrente del Data Segment, ad eccezione però di alcune particolari istruzioni di gestione delle stringhe che invece considerano per il registro DI l'Extra Segment nel calcolo dell'indirizzo. Riassumiamo perciò nella seguente tabellina questo ennesimo con-

cetto fondamentale legato alla scelta dei costruttori e progettisti di segmentare la memoria in quattro parti. In particolare si ha dunque che:

Offset contenuto in	Segmento di default
BP	SS
SP	SS
SI	DS
DI	DS
	(ES in alcuni casi)

Come tanti altri concetti fondamentali che abbiamo sin qui incontrati e ripetuti, anche questo dei segmenti associati per default ad alcuni registri è di fondamentale importanza e come tale dovrà essere tenuto ben in conto tutte le volte che in un programma si utilizzerà uno di questi quattro registfi, in particolar modo i due «Pointer Register».

Tra l'altro, anche se il tutto può sembrare ancora una volta macchinoso, appare alquanto logico che lo Stack Pointer (SP) inteso come offset debba fare riferimento allo Stack Segment (SS), altrimenti a che cosa servirebbe quest'ultimo segmento?!

Inoltre sia SI che DI, a parte i casi particolari di cui ci occuperemo nel seguito, sono abbastanza «innocui» in quanto anche in questo caso non è innaturale associare il loro campo d'azione al Data Segment, così come senza accorgercene abbiamo fatto per i registri generali. Quello che invece è da imparare bene è l'associazione di BP allo Stack Segment, che in un primo tempo può lasciare tra lo stupito e l'indifferente: in pratica, per esperienza diretta, possiamo dire che si tratta di un registro poco usato nei programmi generici, mentre viceversa viene usato pesantemente in ben precise applicazioni, per ottenere risultati praticamente impossibili per i microprocessori ad 8 bit: ci stiamo riferendo alla multiprogrammazione ed a quelle routine usate da più processi, dette «rientranti» («Reentrant»), che possono essere chiamate «contemporaneamente» (sì, è proprio così!) da processi differenti. Su questi argomenti torneremo un poco alla volta, dal momento che necessitiamo di altre conoscenze che apprenderemo man mano.

Con questo abbiamo terminato la descrizione dei singoli registri: ritorneremo sull'argomento quando inizieremo a parlare dei modi di indirizzamento dell'86/88. Ora ci occupiamo di un altro argomento importante ed in un certo senso più semplice.

I Flag

Non ci soffermeremo certo in questa sede sul concetto di «flag» in quanto ben noto ai programmatori, ma andiamo ad analizzare quali e quanti sono i flag a nostra disposizione, vedendo in quali casi si ha un funzionamento analogo agli altri microprocessori ed in quali casi siamo di fronte a delle innovazioni.

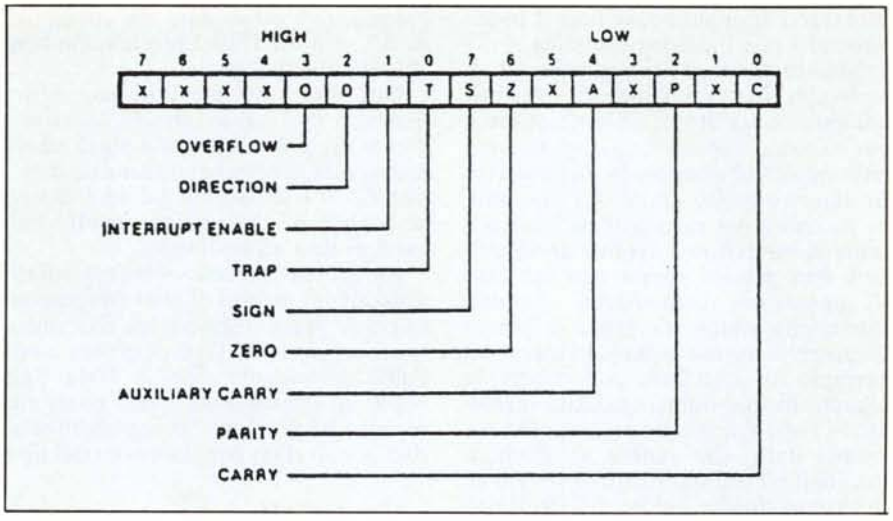


Figura 1 - I Flag dei microprocessori 8086/8088 sono contenuti in una word, i cui bit significativi sono indicati in figura. I bit indicati con «X» sono «non significativi» (don't care) e come tali devono essere ignorati.

Innanzitutto i Flag sono 9 e trovano posto all'interno di un registro a 16 bit detto «Flag Register»: 5 di questi sono equivalenti a quelli presenti nell'8080 e riguardano lo stato del processore a seguito di operazioni logico aritmetiche; i restanti 4 invece sono una novità dell'86/88 e si riferiscono ad altrettante condizioni operative. Ma andiamo con ordine analizzando innanzitutto la loro posizione nell'ambito dei 16 bit del Flag Register. Facendo riferimento alla figura 1, vediamo di quali flag si tratta, iniziando dal bit meno significativo.

Dapprima incontriamo il carry (CF), il ben noto bit settato quando un'operazione algebrica richiede il riporto, oppure usato durante le operazioni di shift: in generale assume la funzione di un 17° bit nelle operazioni coinvolgenti dati a 16 bit, oppure il 9° per calcoli coinvolgenti byte.

Subito dopo troviamo il flag di parità (PF, Parity Flag) usato tutte quelle volte in cui si deve analizzare la parità di un byte o di una word, intesa come numero di bit pari ad «1».

L'Auxiliary Carry (AF) invece è un flag (poco) usato in operazioni coinvolgenti quantità espresse in BCD (Binary Coded Decimal, decimale espresso in binario), laddove ci sia un riporto tra i primi quattro bit ed i successivi quattro di un byte.

Il flag successivo è l'arcinoto «Zero» (ZF), che è settato quando il risultato di una certa operazione è appunto nullo, secondo quella strategia che lascia sconcertati i programmatori alle prime armi, dal momento che sembra un gioco di parole: «il flag di zero vale uno se il risultato è zero, mentre vale zero se il risultato è diverso da zero»...

Sul quinto flag «ordinario», il flag di «Segno» (SF, Sign Flag), non vi è nulla di nuovo da dire se non ricordare che è sempre uguale al bit più significativo del risultato di una certa operazione: in particolare ricordiamo qui la convinzione «naturale» della logica complementata che considera negativi i numeri avendo il bit più significativo settato (il 16° o l'8° a seconda del tipo degli operandi), dotati perciò di «segno negativo», appunto indicato dal flag omonimo. I flag successivi sono come detto nuovi ed ora andremo a conoscerli: nel corso del nostro studio dell'assembler e delle sue istruzioni ne incontreremo alcuni parecchie volte, mentre per altri ci dovremo riferire ad applicazioni particolari.

Il primo che incontriamo è il «Trap Flag» (TF) il quale, se settato, consente al microprocessore di lavorare in «single-step» utilissimo in fase di debugging di un programma: il procedimento non è però così semplice come sembra, in quanto bisogna prevedere un apposito programma di gestione del single-step, cosa che si ottiene con i cosiddetti «monitor».

Il secondo è l'«Interrupt-enable Flag» (IF) che consente di abilitare o disabilitare gli interrupt provenienti dall'esterno: su questo argomento, è inutile dirlo, torneremo ampiamente nei prossimi numeri.

Il terzo flag invece è di più immediata utilizzazione: si tratta del «Direction Flag» (DF) e serve nelle istruzioni di gestione delle stringhe per auto-incrementare o auto-decrementare i puntatori subito dopo l'esecuzione dell'istruzione; ciò consente di usare un'unica istruzione sia nel caso di incremento che di decremento automatico.

L'ultimo dei quattro flag aggiunti è l'«Overflow Flag» (OF), il cui nome

può trarre in inganno, se non si pensa bene alla sua funzione specifica.

Mentre di solito nei linguaggi ad alto livello una condizione di overflow compare a seguito di un'operazione il cui risultato supera il valore massimo consentito (ad esempio la divisione tra un numero molto grande ed uno molto piccolo), nel caso dell'86/88 il termine di «overflow» assume un significato ben preciso soprattutto nel caso di operazioni coinvolgenti quantità numeriche dotate di segno e perciò espresse in forma complementata: un esempio di overflow può essere la somma di due numeri positivi (aventi cioè i rispettivi bit più significativi entrambi nulli) che genera un risultato avente il bit più significativo settato ed indicante cioè un valore negativo, nettamente in contrasto con quanto ci si attendeva. Come si vede anche in questo caso si tratta di un flag utilissimo per i calcoli in logica complementata, laddove prima occorreva lavorare con shift supplementari che coinvolgevano perciò il flag di Carry; tra l'altro, e di ciò ci occuperemo parlando della gestione degli interrupt, c'è persino un interrupt attivabile via software in caso di overflow.

La tecnica del «Segment Override»

Riservandoci di tornare per i dettagli quando parleremo dei modi di indirizzamento, affrontiamo ora un argomento delicato (tanto per cambiare...) che coinvolge ancora una volta i registri di segmento: vedremo come si possano cambiare alcune caratteristiche di default.

Come vedremo dunque nella prossima puntata e come d'altronde già sappiamo, quando facciamo riferimento ad una certa locazione per effettuare una certa operazione sul suo contenuto, dobbiamo distinguere, nell'atto di indirizzare la cella di memoria, il suo segmento di appartenenza dal suo «offset»: in generale per una locazione appartenente al Data Segment il «Segment Register» relativo è come noto il registro DS, mentre l'offset sarà contenuto o direttamente nell'istruzione oppure, come abbiamo visto in questa puntata, in uno dei registri generali come pure nei registri «Index». In tutti questi casi ed in particolare in quelli analizzati in questa sede, sappiamo che l'86/88 fa riferimento per default al registro DS.

Nel caso in cui invece vogliamo accedere al dato contenuto nella nostra locazione in esame con un indirizzamento «indiretto» per mezzo del registro BP (Base Pointer), allora sappiamo, dalla tabellina precedente, che a dispetto delle nostre intenzioni il nostro assemblatore andrà a cercare la locazione di offset dato dal contenuto di BP, non nel Data Segment, ma bensì nello Stack Segment.

Ora si può ovviare a questo inconveniente (nel senso che la locazione con molta probabilità non potrà essere indirizzata fisicamente) con due metodi: uno lo conosciamo già ed è di tipo se vogliamo «definitivo», mentre l'altro è di tipo «transitorio».

La scorsa puntata abbiamo infatti visto alcuni esempi di mini-programmi in cui lo Stack Segment era fisicamente sovrapposto al Data Segment a sua volta coincidente con il Data Segment: in questo caso basta porre nel registro SS il valore del segmento relativo ai dati (DS) con istruzioni del tipo

```
MOV AX,DATA
MOV DS,AX
MOV SS,AX
```

mentre bisogna avvisare l'Assembler della situazione, con una direttiva del tipo

```
ASSUME DS:DATA,SS:DATA
```

In questo modo, tornando al caso del registro BP contenente l'offset di una locazione, per default l'Assembler farà riferimento allo Stack Segment, ma questo coincide proprio con il Data Segment: la nostra variabile potrà essere correttamente indirizzata o, come suol dirsi, è «raggiungibile».

Se invece siamo nel caso in cui i quattro segmenti sono separati l'uno dall'altro o meglio non iniziano tutti allo stesso paragrafo, la locazione in esame non potrebbe essere «raggiunta» se non grazie al particolare meccanismo dell'«Override», per mezzo del quale nell'istruzione si forza l'assemblatore ad usare il Segment register che vogliamo noi: in tal modo «scavalchiamo», «sovrapponiamo» il default con un valore scelto da noi.

Vediamo dunque più da vicino la situazione con un esempio.

Supponiamo dunque di trovarci nella situazione di quattro segmenti separati e di una variabile (ALFA) posta nel Data Segment: supponiamo di voler mettere l'offset di tale locazione in un registro.

Volendo porre tale offset nel registro generale AX, allora useremo l'istruzione

```
MOV AX,OFFSET ALFA
```

ben diversa dalla

```
MOV AX,ALFA
```

che carica in AX il *contenuto* della cella di indirizzo ALFA.

Invece bisogna porre attenzione se vogliamo caricare l'offset di ALFA nel registro BP o SP con l'istruzione

```
MOV BP,OFFSET ALFA
```

oppure con

```
MOV SP,OFFSET ALFA
```

entrambe corrette dal punto di vista sintattico e che come tali non disturberanno l'assemblatore: in entrambi i casi ancora una volta viene correttamente caricato nel registro in questione l'offset della variabile ALFA, che, ricordiamolo, è posta nel Data Segment.

Quando ora andremo ad indirizzare in maniera indiretta la variabile ALFA, con un procedimento di cui parleremo la prossima puntata, ecco che ci dovremo ricordare che la variabile ALFA è nel Data Segment, mentre BP ed SP fanno per default riferimento allo Stack Segment: bisognerà allora comunicare all'Assembler la nostra intenzione di «forzare» un segmento diverso da quello di default, indicandolo semplicemente nell'istruzione.

In tal modo si genererà addirittura un cosiddetto «byte di prefisso» che verrà posto subito prima della codifica dell'istruzione senza «override».

Questo fatto comporterà poi che il microprocessore, leggendo tale prefisso, automaticamente forzerà come segmento a cui fa riferimento l'istruzione successiva, proprio quello relativo al prefisso.

In caso contrario e cioè dimenticandosi di effettuare l'«override», ancora una volta non si otterrà alcuna segnalazione di errore, ma i guai cominceranno dopo, quando il programma verrà eseguito.

In questo caso infatti verrà indirizzata una locazione di memoria avente l'offset prefissato dall'istruzione, ma appartenente allo Stack Segment e perciò posta da tutt'altra parte della memoria.

Tra l'altro in questa locazione potrebbe anche esserci un indirizzo di ritorno di una subroutine in corso di esecuzione, per cui è facile immaginarsi cosa può succedere se il valore contenuto viene alterato...

Comunque come detto precedentemente, ritorneremo più volte su questo argomento, che tra l'altro è praticamente impossibile esaurire in un paragrafo: ecco che alcuni punti rimasti oscuri verranno chiariti con piccoli esempi pratici, che ora invece non possiamo ancora proporre in quanto abbiamo bisogno di conoscere altri argomenti.

Il più importante di questi riguarda i vari modi di indirizzamento dell'86/88, dei quali ci occuperemo nel prossimo numero.

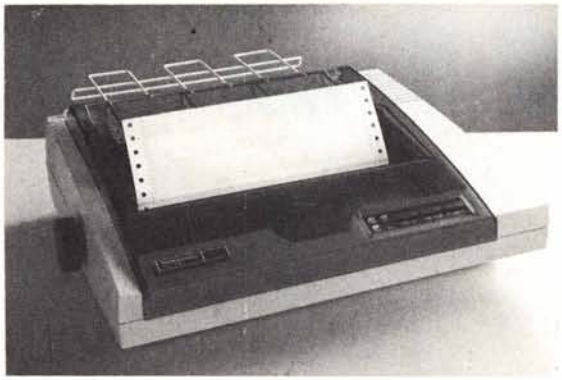
MC

MASTERBIT
MIPECO

VENDITA PER CORRISPONDENZA

INTERFACCIA PARLANTE CURRAH L. 75.000

Scrivete le parole da pronunciare "Lei" le leggerà:
LET S$ = "sAlve" enter
sentirete la parola salve dall'altoparlante del T.V.
Molti programmi prevedono già il suo uso (Birds and the Bees, Lunar jet man, maziacs, VOICE CHESS ecc.
Compreso nel prezzo manuale completo in italiano più un programma compilatore per farle pronunciare in italiano qualsiasi parola richiesta.
Parla attraverso il televisore con chiara voce sintetica.



MANNESMANN MT 80+

L. 599.000

80 col. - 100 cps - interfaccia Centronics - foglio singolo e modulo continuo - bidirezionale.

QL SINCLAIR 128K 549.000

Tutto compreso 6 mesi di garanzia

CPU MOROTOLA 68008 da 32 BIT e 2 microdrive. Ultima versione con nuovi programmi, alimentatore, manuale in inglese, manuale in italiano, 4 cartucce con i 4 programmi gestionali + 1 cartuccia con giochi originali (PIRATE, ZETA, PED, GUN, SREAKOUT, HUNT) e in regalo un ottimo copiatore per mdv e floppy di Massimo Rossi

SPECTRUM 48K PLUS 299.000

Tutto compreso 6 mesi di garanzia

con lo SPECTRUM plus **manuale in italiano** *e in regalo 5 programmi in italiano (conto corrente, grafica, funzioni, bioritmi, esapedone + il Supercopiatore di Massimo Rossi)*

QL 512 K 949.000
Espansione da 512K montata internamente, non necessita di alimentazione supplementare e lascia il connettore libero per altre periferiche.

nuovo SPECTRUM 48K + ... 299.000
manuale in italiano, cavetti alimentatore, cassette dimostrative e oltre 50.000 lire di software originale e in italiano.

CALCOLATORE SCIENTIFICO SHARP ... 49.000
EL - 506 P. Funz. Trig, BIN-OCT-HEX

10 RULLI di carta termica 29.000

MANNESMANN TALLY tutti i modelli

MT 80+ 599.000
foglio singolo e continuo, interfaccia Centronics, 100 cps vari set di caratteri - Bidirezionali.

MT85 ..... 899.000
interfaccia Centronics o seriale a scelta 180 cps 80/136 col. foglio singolo e continuo.

DISCHI 3"½ 13.000
DISCHI 3"½ 10 pezzi 100.000
garantiti doppia faccia e doppia densità

INTERFACCIA PER JOYSTICK UNA PRESA 29.000
tipo Kempston, per tutti i joystick stand. 9 PIN D

INTERFACCIA PER JOYSTICK DUE PRESE 39.000
tipo Kempston, per tutti i Joystick stand. P PIN D.

JOYSTICK STANDARD 9 PIN D 15.000

CONVERTITORE 99.000
Da RS232 a Centronics per interfaccia 1 o per QL **cavi e connettori speciali compresi.**

INTERFACCIA CENTRONICS SPECTRUM 99.000
senza software tutto su Rom compreso il copy

8 CARTUCCE x MICRODRIVE 49.000

TRISLOT 27.000
presa tripla per connettore Spectrum

MANUALE IN ITALIANO x SPECTRUM 16.000
«Come usare il tuo Spectrum»

ROM «JS» NUOVO TIPO (256K + 128K) . 99.000
trasforma il tuo QL in un «JS».

INTERFACCIA PARLANTE CURRAH 75.000
manuale completo in italiano.

ESPANSIONE + 32K x SPECTRUM 59.000
issue 2 o 3 specificare, facilissima da montare, istruzioni dettagliate in italiano con fotografie, porta il VS. Spectrum da 16 K a 48 K. Montaggio gratis.

TASTIERA DELLO SPECTRUM PLUS 85.000
Kit per trasformare lo Spectrum normale in PLUS

DISK DRIVE 3"½ x INTERF. x QL 619.000
Oltre 700K formattati

DISK DRIVE 3"½ INTERF. x SPECTRUM 519.000
Oltre 700K formattati

KIT DI ESPANSIONE x QL A 512 ..... 249.000
Si monta all'interno del QL, si consiglia l'assistenza di un tecnico specializzato.

ESPANSIONE DEL VOSTRO QL A 512K 349.000
Montata all'interno del Vostro QL e collaudata con garanzia di 3 mesi spedite il Computer solo dopo aver avuto un contatto telefonico.

TOOLKIT II x QL SU ROM 89.000

MASTERBIT
MIPECO

VENDITA PER CORRISPONDENZA

PARTI DI RICAMBIO PER SPECTRUM E QL

GARANZIA 48H: oltre la normale Garanzia di 6 mesi per i Computer e di 3 mesi per gli accessori, la MASTERBIT MIPECO si impegna a sostituire tutto il materiale trovato malfunzionante, entro 48 ore dal ricevimento.

AVVERTENZE - tutti i prezzi sono comprensivi di IVA e spese postali - per ordini inferiori alle 50.000 lire aggiungere L. 5.000 per contributo spese di spedizione — pagamento contrassegno al ricevimento del pacco — è gradito un contatto telefonico — **sconti quantità.**
Listino prezzi aggiornato anche su richiesta telefonica.

MASTERBIT
MIPECO
Viale dei Romagnoli 35
00121 OSTIA LIDO RM - CAS. POST. 3016

ORDINI TELEFONICI (ore 8.30/19.30): 06/5611251