di Pierluigi Panunzi

ASSEMBLER 8086 8088

Il set di istruzioni

Dopo aver parlato, nel corso delle precedenti puntate, di argomenti puramente teorici quali la struttura logica di un programma redatto in Assembler 8086/88, la gestione dei segmenti, le modalità di indirizzamento e da ultimo le direttive dell'assemblatore, ecco che a partire da questa puntata inizieremo l'analisi dettagliata del set di istruzioni del nostro microprocessore.

Allo scopo dunque di classificarle ed analizzarle meglio, abbiamo perciò suddiviso le istruzioni nei sei gruppi seguenti: istruzioni aritmetiche, istruzioni di trasferimento dati, istruzioni logiche, istruzioni di gestione di stringhe, istruzioni di controllo, istruzioni avanzate.

Iniziamo dunque dal primo gruppo.

Le istruzioni aritmetiche

Appartengono a questo gruppo tutte quelle istruzioni che consentono di effettuare intanto quelle operazioni aritmetiche «primitive» che devono essere presenti in un qualunque microprocessore e poi altre operazioni «avanzate» decisamente utili ed infine altre operazioni (che ci sia concesso definire «optionals») spesso dimenticate e perciò sottoutilizzate.

Ecco dunque che il primo gruppo si è già suddiviso in tre sottogruppi: andiamo dunque ad analizzare le istruzioni aritmetiche «primitive».

Si tratta, come è facile immaginare, delle istruzioni che effettuano l'addizione e la sottrazione, rispettivamente con e senza riporto (carry, CF) e con e senza prestito (borrow, rappresentato sempre dal carry, CF).

In dettaglio si ha:

```
ADD dest, source
ADC dest, source
SUB dest, source
SBB dest, source
```

dove, seguendo la prassi comune, abbiamo indicato, dopo il cosiddetto «codice mnemonico», le quantità su cui opera la singola istruzione, rispettivamente la destinazione (dest) e la sorgente (source).

Tanto per ricordare il significato di questi semplici concetti, facciamo un esempio: la scrittura «ADD dest, source» impone al processore di sommare i contenuti di «source» e quello di «dest» e di porre il risultato in «dest» oppure, ma è esattamente lo stesso, indica al micro di porre nella «destinazione» la somma della «sorgente» e della «destinazione» stessa.

Ricordato perciò «come» si legge un'istruzione a due operandi (dimenticavamo di ricordare che quanto detto si applica in generale a tutte le istruzioni a due operandi), vediamo in dettaglio le possibili combinazioni di «dest» e «source» permesse dal microprocessore.

In particolare abbiamo riportato in tabella le possibili combinazioni.

Istruzioni ADD, ADC,SBB, SUB

	Destinazione	Sorgente
1)	registro	registro
2)	registro	memoria
3)	memoria	registro
4)	accumulatore	immediato
5)	memoria	immediato
6)	registro	immediato

Abbiamo già visto nel corso delle varie puntate cosa significano i termini riportati nella tabella, ma rinfranchiamo la memoria ricordando che:

- «registro» rappresenta in generale un qualsiasi registro interno della CPU, ad 8 o a 16 bit
- «memoria» rappresenta una qualsiasi cella di memoria indirizzata direttamente, indirettamente, in modo indicizzato e/o basato (si riveda a tal proposito la puntata relativa ai vari modi di indirizzamento della memoria da parte dell'8086/88)
- «accumulatore» è il registro AX, se l'altro operando è a 16 bit oppure AL o AH se l'altro operando è ad 8 bit
- «immediato» è un qualsiasi valore numerico.

Facciamo subito alcuni esempi relativi alle sei possibilità:

```
1) ADD BX, BP
   SBB CL, DL
2) ADC SI, ALFA
   SUB CX, BETA[BX+5][DI]
3) ADD GAMMA, AL
4) SUB AX, 1000H
5) ADD ALFA[BP][SI], 10
6) ADC CL, 4
```

Non c'è nulla di particolare da aggiungere se non che i flag che vengono alterati («affected») a seguito dell'operazione sono: OF, SF, ZF, AF, PF, CF e cioè rispettivamente i flag di overflow, di segno, di zero, l'auxiliary flag, di parità ed il carry.

Allo stesso sottogruppo appartengono le istruzioni INC, DEC e NEG, le quali rispettivamente incrementano, de-

crementano e «complementano a 2» l'operando indicato nell'istruzione: si ha infatti

```
INC dest
DEC dest
NEG dest
```

dove in questo caso si hanno le due combinazioni della tabellina seguente.

Istruzioni INC, DEC, NEG
Destinazione
1) registro
2) memoria

Per queste tre istruzioni riportiamo dunque gli esempi:

```
1) INC SI
   NEG BP

2) DEC ALFA
   NEG BETA[BP]
   INC GAMMA+3
```

Per quanto riguarda i flag abbiamo che le tre istruzioni si alterano: OF, SF, ZF, AF e PF, mentre soltanto l'istruzione NEG altera anche il CF, come dire che per effetto di un incremento o di un decremento il carry rimane inalterato.

Istruzioni aritmetiche: moltiplicazione e divisione

Al secondo sottogruppo delle istruzioni aritmetiche appartengono quattro istruzioni molto importanti e che praticamente si cominciano a trovare solo nei microprocessori a 16 bit: si tratta della moltiplicazione e della divisione, rappresentate rispettivamente dalle coppie IMUL, MUL e DIV, IDIV.

In questo caso si hanno per ognuna delle due operazioni le due possibilità (contraddistinte dalla presenza o meno della «I» iniziale) di effettuare l'operazione tra operandi dotati di segno (IMUL e IDIV) oppure senza segno (MUL e DIV), dove il segno è rappresentato dal bit più significativo del singolo operando: se tale bit è «0» allora l'operando è positivo, mentre se è «1» allora l'operando è negativo.

Analizziamo innanzitutto la moltiplicazione, facendo alcune considerazioni: la moltiplicazione tra due operandi senza segno non comporta problemi in quanto il prodotto tra quantità positive dà sempre un risultato positivo e tale verrà interpretato un risultato avente il bit più significativo pari ad «1».

Viceversa effettuando l'operazione tra quantità «segnate» allora anche il risultato sarà di tale tipo e si dovrà tener conto della ben nota «regoletta dei segni»: se i due operandi hanno lo stesso segno allora il risultato è positivo, mentre se gli operandi hanno segno opposto allora il prodotto avrà segno negativo.

In alcuni casi però si possono verificare delle condizioni in cui il prodotto tra due valori di stesso segno genera un risultato che viceversa possiede il bit più significativo pari ad «1» (indicante dunque un valore negativo) oppure casi in cui il prodotto tra una quantità positiva ed una negativa dia un risultato positivo: sono questi i casi in cui il risultato ottenuto è corretto in valore assoluto, ma manca un bit per poterlo rappresentare correttamente.

Il bit che manca è proprio quello che viene usato per la rappresentazione del segno.

Ecco che dunque nel caso dell'istruzione «IMUL» (moltiplicazione tra quantità dotate di segno), gli unici flag che vengono settati (OF e CF) contemporaneamente verranno posti ad «1» se la parte alta del risultato non è l'estensione del segno della parte bassa.

Viceversa l'istruzione «MUL» (moltiplicazione tra quantità senza segno) setterà i flag OF e CF (e solo quelli) se la parte alta del risultato è diversa da 0.

Visto dunque come le moltiplicazioni alterano i flag, vediamo il formato delle due istruzioni:

```
MUL memoria
MUL registro
IMUL memoria
IMUL registro
```

dove per «registro» e «memoria» vale quanto detto in precedenza: da aggiungere (ed è molto importante) è il seguente fatto che la moltiplicazione (con la «I» oppure senza) avviene tra l'operando indicato e l'accumulatore secondo il seguente schema:

— se l'operando (memoria o registro) è un byte allora il prodotto avviene tra l'operando e AL (parte bassa di AX) ed il risultato viene posto in AX
— se l'operando è una word allora il prodotto viene effettuato tra l'operando stesso ed AX (un'altra word) ed il risultato a 32 bit così ottenuto viene posto nella coppia DX:AX e cioè la parte più significativa in DX e quella meno significativa in AX.

Come conseguenza di queste due possibilità si ha che volendo moltiplicare un byte per una word bisogna:

1) caricare in AL il byte
2) estendere il segno di AL su tutto AX
3) moltiplicare AX per l'operando (word) per ottenere il risultato correttamente su due word (DX ed AX).

In particolare l'estensione del segno del registro AL si ottiene con l'istruzione

```
CBW
```

il cui nome deriva dalle iniziali delle parole «Convert Byte to Word»: ecco che se AL ha il bit più significativo a «0» allora AH verrà posto a 0, mentre se AL ha il bit più significativo ad «1» allora AH verrà posto al valore OFFH. Aggiungiamo infine che l'istruzione CBW non altera alcun flag.

Vediamo ora tre esempi relativi alle tre possibilità sopra esposte:

— prodotto tra due byte

```
MUL  CH      ; moltiplica AL per CH con risultato in AX
```

— prodotto tra due word

```
     ALFA    DW  1234H
     ...
     MUL  ALFA    ; prodotto tra ALFA ed AX
                  ; con risultato nella coppia DX,AX
```

— prodotto tra un byte ed una word

```
BETA     DB   55H
GAMMA    DW   5555H
...
MOV  AL,BETA  ; byte in AL
CBW           ; estensione del segno
MUL  GAMMA    ; prodotto tra "BETA esteso su word" e
              ; GAMMA con risultato in DX,AX
```

La divisione

Per quanto riguarda questa operazione possiamo dire che si può effettuare la divisione tra due operandi dotati o meno di segno, operandi che, analogamente a quanto detto per la moltiplicazione, devono essere, uno, l'accumulatore e l'altro o un «registro» o la «memoria» (sempre con il solito significato).

In questo caso il dividendo è l'accumulatore «estensibile» (AX per un'operazione ad 8 bit e la coppia DX:AX per un'operazione a 16 bit), mentre il divisore è riportato nell'istruzione stessa come operando.

È proprio l'operando a far decidere di quale tipo di divisione si tratta:

— se il divisore è un byte, allora il dividendo è solo AX, il quoziente viene posto in AL ed il resto (sì viene generato anche il resto!) in AH
— se il divisore è una word, allora il dividendo è dato dalla coppia DX:AX (dove in DX è posta la parte più significativa), il quoziente è posto in AX ed il resto è stavolta posto in DX.

Ma questo non è tutto: si tratta solo del caso in cui, i valori posti negli operandi non creano problemi di overflow.

Rimane infatti da vedere cosa accade nel caso in cui il divisore sia zero o un valore molto piccolo (è il solito problema della «divisione per 0»), nel qual caso viene addirittura generato un interrupt di tipo 0...

A questo punto bisogna però analizzare singolarmente i due casi di divisione con segno o senza segno: iniziamo dalla DIV (perciò senza segno).

In particolare viene controllato che il rapporto tra il dividendo ed il divisore non superi la massima capacità del registro che dovrà ospitare il quoziente:
— se la divisione era tra una word (AX) ed un byte (operando della DIV) allora il quoziente (AL) non può superare il valore OFFH
— se la divisione era tra una double-word (DX:AX) ed una word (operando) allora il quoziente (AX) non può superare il valore OFFFFH.

In caso contrario il nostro 8086/88 lascia indefiniti i valori dei registri dove dovrebbe porre il risultato e genera a tutti gli effetti, come detto, l'interrupt 0: fra qualche puntata, quando parleremo in dettaglio del meccanismo di gestione degli interrupt, il discorso si chiarirà decisamente.

Per adesso segnaliamo la sequenza di operazioni che il microprocessore svolge automaticamente:
— salva nello stack il registro dei flag
— azzera i flag IF (Interrupt Flag) e TF (Trace Flag)
— salva CS nello stack
— carica in CS il contenuto delle locazioni assolute 00002H e 00003H.
— salva il registro IP nello stack
— carica in IP il contenuto delle locazioni assolute 00000H e 00001H.

Appunto fra qualche puntata apparirà più chiara la connessione tra i valori contenuti nelle locazioni di indirizzo assoluto indicate e l'interrupt 0.

Per quanto riguarda l'istruzione IDIV, bisogna ricordarsi che coinvolge quantità dotate di segno e che perciò, nel calcolare se il quoziente può entrare comodamente nel registro a ciò preposto, si dovrà parlare di «valore assoluto massimo» che si può porre nel registro quoziente. In particolare viene ancora controllato che il rapporto tra il dividendo ed il divisore non superi la massima capacità del registro che dovrà ospitare il quoziente, però tenendo conto stavolta del segno:
— se la divisione era tra una word (AX) ed un byte (operando della DIV) allora il quoziente (AL), se maggiore di 0 (MSB pari a «0»), non può superare il valore 07FH (127 in decimali) e viceversa se è minore di 0 (MSB pari ad «1») non può essere inferiore ad 80H (pari a −128 in decimale)
— se la divisione era tra una double-word (DX:AX) ed una word (operando) allora il quoziente (AX), se positivo, non può superare il valore 07FFFH (pari a 32767 in decimale), mentre nel caso in cui risulti negativo (perché ha l'MSB pari ad «1») non può essere inferiore a 8000H (che rappresenta invece il valore decimale −32768).

Nel caso in cui invece il quoziente non rientra in questi range di valori, allora, analogamente al caso precedente, viene generato un interrupt 0 con la stessa sequenza di operazioni già viste per la DIV, che il microprocessore svolge automaticamente.

Vediamo ora alcuni esempi, che rispecchiano le varie possibilità offerte dall'Assembler:
— divisione tra una word (AX) ed un byte (operando)

```
ALFA      DW 2222H
...
MOV  AX,ALFA    ; dividendo
DIV  DH         ; DH e' il divisore
                ; quoziente in AL e resto in AH
```

— divisione tra un byte (AL) ed un altro byte (operando)

```
BETA      DB 100
GAMMA     DB 7
...
MOV  AL,BETA    ; dividendo ad 8 bit
CBW             ; ...esteso a 16 bit
DIV  GAMMA      ; GAMMA e' il divisore
                ; quoziente ancora in AL e resto in AH
```

— divisione tra double-word (DX e AX) ed una word (operando)

```
DIVISORE  DW 3FFFH
...
MOV  AX,PARTE_BASSA_DEL_DIVIDENDO
MOV  DX,PARTE_ALTA_DEL_DIVIDENDO
DIV  DIVISORE
                ; quoziente in AX e resto in DX
```

— divisione tra word e word (operando): in questo caso dato che l'operando è una word allora il divisore deve essere una double-word

```
WORD1     DW 4444H
WORD2     DW 33
...
MOV  AX,WORD1   ; dividendo a 16 bit
CWD             ; ... esteso a 32 bit
DIV  WORD2      ; il divisore e' GAMMA
                ; quoziente in AX e resto in DX
```

In quest'ultimo esempio abbiamo usato l'istruzione

CWD

che consente l'estensione del segno del registro AX sul registro DX: è praticamente analoga alla CBW.

In particolare se l'accumulatore è minore di 8000H e cioè se ha un bit più significativo pari a «0» allora DX viene posto a 0, altrimenti nel caso in cui l'MSB di AX sia pari ad «1», questo «1» viene esteso a tutto DX, che così viene caricato al valore OFFFFH.

Le istruzioni aritmetiche «optionals»

Appartengono a questo sottogruppo una manciata di istruzioni, sei per l'esattezza, utilizzate per scopi molti particolari quali le operazioni tra numeri espressi in BCD: si tratta di istruzioni il cui meccanismo è apparentemente complicato (nulla a che vedere a confronto dei «mostri sacri» che sono la divisione e la moltiplicazione), mentre viceversa sono «simpatiche» oltreché raramente usate.

Cominciamo dalla prima di un sotto-gruppo rappresentato da 4 istruzioni che è

AAA

che trae il nome dalle iniziali delle parole «ASCII Adjust for Addition» o meglio «Unpacked BCD ASCII Adjust ecc.».

È, come detto, ma prima di un gruppo al quale fanno parte anche la AAS, la AAM e la AAD, parenti della precedente in quanto effettuano tutte e tre un «ASCII Adjust» per rispettivamente una «Subtraction», una «Multiplication» ed una «Division».

In tutti e quattro i casi correggono il risultato della rispettiva operazione e contenuto nella coppia AH:AL (attenzione! AX in questo caso non avrebbe senso logico), in modo tale da avere in AH ed AL rispettivamente la cifra delle decine e delle unità (espresse con solo 4 bit) di un nu-

mero espresso in BCD, «unpacked» dal momento che le due cifre non sono «impacchettate» in un unico byte.

Torniamo dunque all'istruzione AAA.

Supponiamo dunque di avere effettuato un'addizione che dia come risultato due cifre BCD poste in AH ed AL: l'istruzione AAA provvede ad incrementare di 1 il valore contenuto in AH se, per effetto dell'addizione, in AL ci fosse un valore maggiore di 9 oppure nel caso in cui il flag AF (Auxiliary Flag) risulti pari ad «1».

Prima di analizzare in dettaglio ciò che viene eseguito da una parte dell'istruzione, vediamo un banale esempio di ciò che stiamo dicendo.

Dobbiamo sommare i due numeri «48» e «37» espressi in «Unpacked BCD» e perciò posti in registri separati: ad esempio «48» in AH:AL (ponendo perciò «4» in AH e «8» in AL) e «37» in BH:BL (ponendo «3» in BH e «7» in BL):

```
MOV AX, 0408H; attenzione!
MOV BX, 0307H; anche qui!
```

Ora effettuando l'addizione con

```
ADD AX,BX
```

otterremo in AX il valore 070EH che «non» è un «unpacked BCD»! Impostando perciò l'istruzione

```
AAA
```

otterremo il risultato «corretto» dato da 0805H, rappresentante di quel valore «85» che è proprio la somma dei valori assunti all'inizio.

Dunque l'istruzione AAA effettua le seguenti operazioni in sequenza:

— se AL AND OFH (i quattro bit bassi di AL) è maggiore di 9 oppure se il flag AF è pari ad «1» (vedremo dopo il significato di quest'altra condizione) allora
— somma 6 ad AL (è un vecchio trucco per i numeri in BCD!)
— incrementa AH di 1 (ma se AH ora supera il valore 9?)
— setta il flag AF (ecco la risposta! nel caso basterà salvare AL, porre AH in AL, azzerare AH e rieseguire l'istruzione AAA!)
— setta il flag CF
— pone in AL l'AND di AL con OFH.

Per quanto riguarda l'istruzione

```
AAS
```

vale praticamente quanto detto per l'istruzione AAA: abbiamo appena effettuato una sottrazione tra «unpacked BCD» ed ora vogliamo aggiustare il risultato.

In breve le operazioni che l'istruzione in esame esegue sono le seguenti:

— se AL AND OFH (4 bit più bassi di AL) è maggiore di 9 oppure se il flag AF è settato allora
— sottrae 6 da AL (sempre lo stesso trucco vecchio come il mondo...)
— decrementa AH di 1
— setta i flag AF e CF (come prima)
— pone in AL l'AND tra AL stesso ed il valore OFH. Con il che non aggiungiamo altro.

L'istruzione

```
AAM
```

corregge, come detto, il risultato di una moltiplicazione tra «unpacked BCD»: in questo caso le operazioni che compie sono comunque eseguite in quanto dopo aver effettuato una moltiplicazione saremo in generale costretti ad effettuare la correzione.

I due passi che l'istruzione AAM effettua internamente sono:

— pone in AH il valore contenuto in AL diviso per 10 (toh! chi si vede!)
— in AL pone nientemeno che il resto della divisione precedente!

Come si vede dunque l'istruzione è solo apparentemente semplice, dal punto di vista dell'esecuzione all'interno del-l'8086/88.

Per quanto concerne i flag, si ha che l'istruzione AAM altererà se necessario i flag PF (Parity), SF (Sign) e ZF (Zero).

L'ultima istruzione del sottogruppo è la

```
AAD
```

che non è altro che la controparte della precedente per quel che riguarda la divisione.

Non ci ripetiamo ancora sulle occasioni in cui si deve eseguire tale operazione, ma andiamo ad analizzare ciò che il microprocessore esegue:

— pone in AL il valore di AH diviso per 10 (ancora lui!) sommato al vecchio contenuto di AL
— pone in AH il valore 0.

Basta riflettere un attimo per comprendere che queste operazioni sono proprio quel che ci voleva.

Per quanto riguarda i flag, l'istruzione AAD altera i flag PF, SF e ZF così come faceva AAM.

Il secondo sotto-gruppo infine è rappresentato dalle due istruzioni DAA e DAS, che stavolta effettuano la correzione del risultato rispettivamente dell'addizione e della sottrazione di due numeri stavolta «packed».

Lasciamo volentieri al lettore, quasi come esercizio, il compito alquanto semplice di immaginarsi il contesto in cui si deve applicare questa coppia di istruzioni, leggermente più complicate delle «unpacked» AAA e AAS.

A proposito, i nomi DAA e DAS stanno per «Decimal Adjust for Addition» e «Decimal Adjust for Subtraction» rispettivamente.

Per quanto riguarda l'istruzione

```
DAA
```

vediamo quali sono le operazioni che il microprocessore compie:

— se AL AND OFH (i soliti 4 bit bassi di AL) è maggiore di 9 oppure se il flag AF è settato (è sempre la solita storia...)allora
— somma 6 ad AL
— setta il flag AF
— se AL è maggiore di 9FH oppure se il flag CF è settato (osservando il seguito si capisce il perché di questi test...) allora
— somma 60H ad AL
— setta il flag CF (eccolo qui il flag CF posto ad «1»).

A questo aggiungiamo che oltre al flag AF e CF già incontrati, l'istruzione DAA altera all'occorrenza anche i flag PF, SF e ZF, con il che il programmatore esperto potrà prendere le proprie decisioni.

L'ultima istruzione (anche di questa puntata!) è

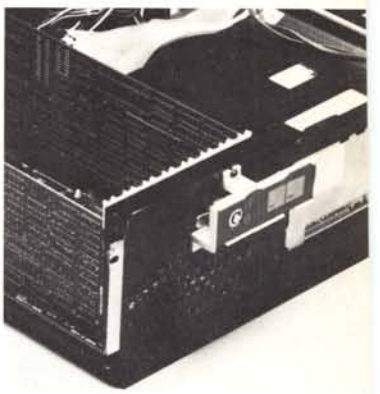
```
DAS
```

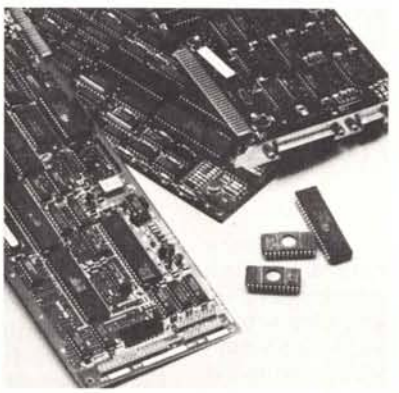
riguardo alla quale forniamo l'elenco delle operazioni effettuate internamente dal microprocessore:

— se AL AND OFH (serve ancora dire perché?!) è maggiore di 9 oppure se il flag AF è pari ad «1» (ancora...) allora
— sottrai 6 al contenuto di AL
— poni il flag AF ad «1»
— se AL è maggiore di 9FH o se il flag CF è posto ad «1» (nessun commento...) allora
— sottrai 60H dal contenuto AL
— setta il flag CF

Terminiamo dunque la puntata dicendo che quest'ultima istruzione altera i soliti flag (PF, SF e ZF) e dando l'appuntamento alla prossima dove tratteremo in dettaglio le istruzioni di trasferimento dati.

Sapete già a chi rivolgervi per la manutenzione dei vostri personal computer?

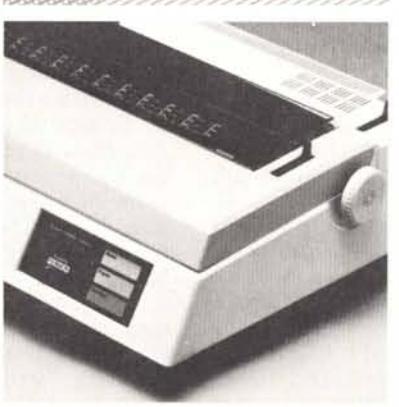


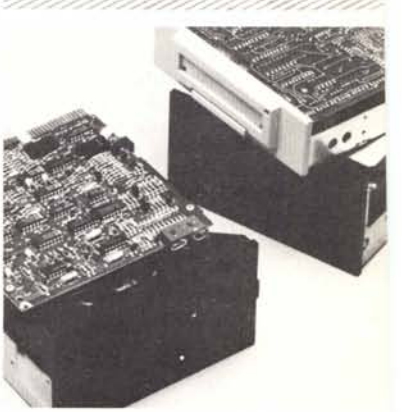


PC MAINT

Per risolvere tutti i problemi di manutenzione dei vostri computer, chiedete di PC MAINT, il centro specializzato nella riparazione di personal ed accessori.

- PC MAINT esegue in tempi brevi riparazioni o sostituzioni del materiale fuori uso.
- La costante disponibilità nel magazzino di prodotti delle migliori marche, assicura qualità e tempestività.
- Un listino prezzi garantisce i costi delle riparazioni.
- PC MAINT offre la sua assistenza anche presso di voi.
- Le riparazioni sono coperte da garanzia di 60 giorni, le sostituzioni di 180 giorni.





P.C. MAINT
PERSONAL COMPUTER MAINTENANCE

Via Bertoloni, 26 · 00197 Roma · Tel. 06/873133

MOCCIAGROUP