di Pierluigi Panunzi

Il set di istruzioni

quarta parte

Proseguiamo in questa puntata l'analisi del set di istruzioni del 386, dando dapprima una rapidissima occhiata a quelle che implementano le istruzioni aritmetiche

Le istruzioni aritmetiche

Essendo il 386 completamente compatibile verso il basso (8086 e 80286), ecco che ritroviamo, nell'ambito delle istruzioni aritmetiche, il set completo che già ben conosciamo: ci sono le istruzioni di addizione con e senza riporto (**ADD** e **ADC** rispettivamente), le sottrazioni con e senza prestito (**SBB** e **SUB** rispettivamente), l'istruzione di «compare» (**CMP**), l'istruzione che effettua il complemento a 2 dell'operando (**NEG**), nonché l'incremento ed il decremento (**INC** e **DEC** rispettivamente).

Fin qui nulla di nuovo, ovviamente, in quanto c'è poco da reinventare riguardo queste semplici operazioni: invece con il 386 è stata aggiunta una caratteristica legata al fatto che questo microprocessore è a 32 bit.

In particolare tutte le istruzioni citate sono state estese al caso di operandi (registri e/o locazioni di memoria) a 32 bit, lasciando ancora il particolare privilegio per il registro AX (accumulatore), che diventa EAX se esteso a 32 bit, di avere delle istruzioni dedicate, codificate con una forma breve.

Vediamo dunque alcuni esempi, ottenuti come al solito per mezzo del TASM della Borland:

```
40                  inc ax
66 40               inc eax
81 D3 04D2          adc bx,1234
66 81 D3 12345678   adc ebx,12345678h
83 FE 01            cmp si,1
66 81 FE FEDCBA98   cmp esi, 0fedcba98h
```

Come già abbiamo visto nelle scorse puntate, l'estensione a 32 bit avviene, nella maggior parte dei casi (ma non come regola fissa), anteponendo al codice «normale» il prefisso **66H**, che indica, come abbiamo già avuto modo di sottolineare, non tanto l'estensione a 32 bit, ma piuttosto il cambiamento di un valore di default: senza ritornare sui dettagli già discussi, ricordiamo in breve che, supponendo di lavorare con quantità «a 16 bit», allora il prefisso commuta il microprocessore a considerarle a 32 bit e, nel caso di registri, a prendere in considerazione i registri estesi.

Altre istruzioni comuni ai «vecchi» microprocessori sono le ben note:

AAA ASCII Adjust after Addition
AAS ASCII Adjust after Subtraction
AAD ASCII Adjust before Division
AAM ASCII Adjust after Multiply
DAA Decimal Adjust after Add
DAS Decimal Adjust after Subtraction

che comunque continuano ad operare sull'accumulatore, inteso come **AL** oppure come coppia **AH AL**.

Visto che siamo in tema di istruzioni rimaste inalterate, citiamo anche le istruzioni di set e reset dei singoli flag:

CLC CLear Carry flag
STC SeT Carry flag
CMC CoMplement Carry flag
CLD CLear Direction flag (... Clear Decrement flag)
STD SeT Direction flag (... SeT Decrement flag)
CLI CLear Interrupt enable flag
STI SeT Interrupt enable flag

nonché le istruzioni di caricamento e memorizzazione dei flag da e verso il registro **AH** (**SAHF** e **LAHF**, rispettivamente «Store AH into Flags» e «Load AH with Flags»).

Due istruzioni che hanno invece avuto un'estensione a 32 bit sono le **CBW** («Convert Byte to Word») e **CWD** («Convert Word to Doubleword»): in particolare la prima, che ricordiamo estendere **con il segno** il contenuto di AL nel registro AX, con l'estensione a 32 bit si chiama **CWDE** («Convert Word to Doubleword with sign Extension») ed in particolare estende **con il segno** il valore posto in AX nel registro EAX.

Questa istruzione non deve essere confusa con la già citata **CWD**, la quale viceversa estende, sempre con il segno, la quantità posta in AX nella coppia di registri **DX:AX** : è proprio questa istruzione, estesa a 32 bit a diventare **CDQ** («Convert Doubleword to Quadword»), la quale estende, ancora una volta con il segno, la quantità posta in **EAX** in una quantità a 64 bit contenuta nella coppia di registri estesi **EDX:EAX**. Ancora una volta, al livello codice operativo troviamo che le istruzioni estese sono precedute dal byte di prefisso:

```
98          cbw
66 98       cwde
99          cwd
66 99       cdq
```

detto questo, passiamo a due importanti operazioni aritmetiche, la moltiplicazione e la divisione, che hanno risentito particolarmente dei benefici dell'estensione a 32 bit delle rispettive istruzioni «normali»: piuttosto che segnalare solo le nuove possibilità (sia rispetto all'8086 che rispetto al 286), decriveremo tutti i casi coinvolgenti tutti i tipi di operandi.

Le istruzioni di moltiplicazione

Iniziamo dalla moltiplicazione, che nel caso del 386 viene eseguita per mezzo di un particolare algoritmo estremamente veloce che consente di ottenere tempi di esecuzione contenutissimi: al massimo 14 e 17 cicli di clock per registri e memorie ad 8 bit, 22 e 25 cicli per registri e memorie a 16 bit ed appena 38 e 41 cicli di clock (ripetiamo, **al massimo!**) per registri e memorie a 32 bit (cioè con entrambi gli operandi a 32 bit...).

Innanzitutto ci occupiamo dalla **MUL**, che, come sappiamo già, fornisce un risultato di tipo «unsigned».

In particolare abbiamo che in tutti i casi il primo fattore è l'accumulatore (AL nel caso ad 8 bit, AX nel caso a 16 bit ed EAX nel caso a 32 bit), mentre l'altro fattore potrà essere una quantità analogamente ad 8, 16 e 32 bit, rispettivamente.

Il risultato invece sarà posto in AX per gli 8 bit, nella coppia di registri DX:AX nel caso a 16 bit e nella coppia di registri estesi EDX:EAX nel caso di moltiplicazioni a 32 bit.

Alcuni esempi di tale istruzione, con la relativa codifica, sono i seguenti (ALFA è una word in memoria, BETA è una doubleword mentre GAMMA è un byte):

```
F6 E1           mul cl
F6 26 0000      mul gamma
F7 E1           mul cx
F7 26 0001      mul alfa
66 F7 E1        mul ecx
66 F7 26 0003   mul beta
```

Per quanto riguarda l'istruzione **IMUL**, la moltiplicazione di tipo «signed» e che cioè tiene conto del segno, c'è da dire che offre una maggiore elasticità nelle possibilità di scelta degli operandi.

In particolare abbiamo, oltre ai sei modi visti per la MUL, altre innumerevoli possibilità, che abbiamo sintetizzato nella tabella 1.

In questa tabella vediamo dunque che è possibile svincolarsi dall'obbligo di ottenere il risultato comunque nell'accumulatore ed infatti è possibile moltiplicare il contenuto di un registro per un valore immediato e porre il risultato in un altro registro.

Ci sarebbe piaciuto pure poter moltiplicare tra loro due registri diversi e porre il risultato in un terzo registro, ma forse saremo accontentati solo dall'80486 ... chissà.

Le istruzioni di divisione

Per quanto riguarda le istruzioni di divisione, c'è subito da dire che non si hanno tutte quelle possibilità che abbiamo con la IMUL, ma abbiamo solamente l'estensione a 32 bit di quanto già conosciamo: inoltre c'è ancora una volta da notare la fenomenale velocità di esecuzione in termini di cicli di clock, che arriva a 22 cicli di clock se il divisore è una memoria ad 8 bit, 30 cicli per una memoria a 16 bit ed appena 46 cicli se dividiamo una quantità a 64 bit (si veda più sotto) una per memoria a 32 bit!

Senza distinguere la **IDIV** (la divisione di tipo «signed») dalla **DIV** (la divisione di tipo «unsigned»), abbiamo che l'operazione ha sempre a che vedere con l'accumulatore: in particolare si può effettuare la divisione tra:

— una quantità a 16 bit ed una ad 8 bit e cioè dividere il contenuto di **AX** per un byte (registro o cella di memoria ad 8 bit, ma non un valore immediato) per ottenere il quoziente nel registro **AL** ed il resto nel registro **AH**.

— Una quantità a 32 bit ed una ad 16 bit e cioè dividere il contenuto della coppia di registri **DX:AX** per una word (registro o cella di memoria a 16 bit, ma non un valore immediato) per ottenere il quoziente nel registro **AX** ed il resto nel registro **DX**.

— Una quantità a 64 bit ed una a 32 bit e cioè dividere il contenuto della coppia di registri estesi **EDX:EAX** per una doubleword (registro esteso o cella di memoria a 32 bit, ma non un valore immediato) per ottenere il quoziente nel registro **EAX** ed il resto nel registro **EDX**.

Come esempio di quanto detto, vediamo le seguenti istruzioni, dove ancora una volta ALFA è una word, BETA è una doubleword e GAMMA è un byte.

```
F6 F9            idiv cl
F6 36 0000       div gamma
F7 F9            idiv cx
F7 36 0001       div alfa
66 F7 F9         idiv ecx
66 F7 36 0003    div beta
```

Le funzioni logiche

In questo caso, accanto ad istruzioni perfettamente identiche a quelle conosciutissime, ma con estensione a 32 bit, abbiamo altre istruzioni del tutto nuove: iniziamo dalle prime...

In particolare si tratta delle ben note **AND, OR, XOR, TEST** e **NOT** sul cui significato non ci soffermeremo certo, ma delle quali ribadiamo il concetto che quanto è stato finora possibile compiere con quantità ad 8 e 16 bit ora può essere esteso a quantità a 32 bit, senza eccessive complicazioni: solo a scopo di completezza indichiamo un esempio di ognuna delle istruzioni citate e riferite a quantità a 32 bit.

```
66 83 E0 02                 and eax,2
66 81 0E 0002 12345678      or beta,12345678h
66 33 CE                    xor ecx, esi
66 85 26 0002               test beta,esp
66 F7 D7                    not edi
```

A questo gruppo (di istruzioni già note) appartengono anche le istruzioni di **shift** e **rotate** che ora vengono estese a quantità a 32 bit.

Sappiamo già che si tratta delle istruzioni **SAL, SAR, SHL** e **SHR**, nonché le **ROL, ROR, RCL,** e **RCR**: sappiamo inoltre che con l'8086 era possibile effettuare queste operazioni di shift o per una volta sola oppure per un numero di volte contenuto nel registro **CL**.

Con l'80186 è stata poi introdotta la possibilità ulteriore di effettuare lo shift per un numero di volte contenuto in un byte immediato.

Ora con il 386 queste tre possibilità sono state estese, come più volte ripetuto, a quantità a 32 bit: in più c'è una differenza sostanziale nel funzionamento, che comporta che ora il numero di cicli di clock richiesti per eseguire le istruzioni è pari a 9 o 10 cicli per le RCR e RCL (operando di tipo registro oppure locazione di memoria), mentre in tutti gli altri casi si ottiene lo shift di un registro (**indifferentemente** ad 8, 16 o 32 bit!) in appena 3 clock e di una locazione di memoria in 7 cicli di clock, in tutti quanti i casi indipendentemente dal numero di shift da compiere.

Ciò avviene grazie ad un apposito circuito hardware («Barrel shifter») posto all'interno del 386 che si occupa dello shift di dati.

Prima di passare alle nuove istruzioni logiche vediamo alcuni esempi delle istruzioni di shift relative a quantità a 32 bit:

```
66 C1 E0 15      shl eax,21
66 D3 CE         ror esi,cl
```

```
      moltiplicazione            esempio
   ------------------------------------------
     AX <- AL    * r8        imul cl
     AX <- AL    * m8        imul gamma
  DX:AX <- AX    * r16       imul cx
  DX:AX <- AX    * m16       imul alfa
EDX:EAX <- EAX   * r32       imul ecx
EDX:EAX <- EAX   * m32       imul beta

    r16 <- r16   * r16a      imul cx,dx
    r16 <- r16   * m16       imul cx,alfa
    r16 <- r16   * i8        imul dx,53
    r16 <- r16   * i16       imul dx,0c1a0h

    r32 <- r32   * r32a      imul esi,ebp
    r32 <- r32   * m32       imul edi,beta
    r32 <- r32   * i8        imul edx,0abh
    r32 <- r32   * i32       imul ecx,0dec0deh

    r16 <- r16a  * i8        imul bx,cx,23h
    r16 <- r16a  * i16       imul dx,ax,1234h
    r16 <- m16   * i8        imul di,alfa,5
    r16 <- m16   * i16       imul di,alfa,5678h

    r32 <- r32a  * i8        imul ebx,eax,34h
    r32 <- r32a  * i32       imul edx,ebx,12345678h
    r32 <- m32   * i8        imul ebp,beta,0fh
    r32 <- m32   * i32       imul ebx,beta,9abcdef0h

dove :
  r8            registro ad 8 bit
  r16,r16a      registri (differenti) a 16 bit
  r32,r32a      registri (differenti) a 32 bit
  m8            cella di memoria ad 8 bit (GAMMA)
  m16           cella di memoria a 16 bit (ALFA)
  m32           cella di memoria a 32 bit (BETA)
  i8            valore immediato ad 8 bit
  i16           valore immediato a 16 bit
  i32           valore immediato a 32 bit
```

Tabella 1 - Elenco di tutte le possibilità di fattori per la moltiplicazione «unsigned» IMUL.

```
66 D1 E5              sal ebp,1
66 C1 1E 0002 0A      rcr beta,10
```

Le istruzioni di shift nuove si chiamano **SHLD** e **SHRD** e cioè rispettivamente «SHift Left in Double precision» e «SHift Right in Double precision» e sono una particolare estensione delle SHL e SHR.

L'istruzione SHLD in particolare agisce su due registri entrambi a 16 o a 32 bit effettuando lo shift verso sinistra di tante volte quanto indicato dal terzo operando dell'istruzione stessa, contemporaneamente però rimpiazzando i bit da destra con i bit provenienti dal secondo registro, anche lui shiftato a sinistra: nella figura 1 vediamo una rappresentazione schematica di quanto detto a parole.

In particolare tutto va come se i due registri fossero un'unica quantità a 32 o 64 bit, da shiftare insieme verso sinistra di «n» volte, con la sola differenza che la parte «meno significativa» rimane inalterata.

Vediamo ancora meglio con un esempio cosa succede: supponiamo perciò che nel registro **ECX** ci sia posto il valore 12345678H, mentre in **ESI** ci sia il valore, ancora a 32 bit, 0FEDCBA98H.

Ora vediamo che cosa succede per effetto dell'istruzione

```
shld ecx,esi,4
```

Innanzitutto «leggiamo» tale istruzione: vogliamo shiftare a sinistra di 4 bit il contenuto di ECX «spingendo» da destra i 4 bit che si ottengono shiftando ESI a sinistra, uno alla volta.

In un certo senso è una generalizzazione della SHL laddove si ricorda che in questo caso il bit che viene «spinto» da destra è sempre uno 0 (infatti la SHL serve a moltiplicare per 2 e potenze di 2 l'operando): nel caso della SHLD invece non si immette da destra sempre e solo un bit pari a 0, ma viceversa vengono estratti più bit, uno per uno, da sinistra ancora, dal secondo registro, il tutto un certo numero di volte, posto come valore immediato oppure nel registro CL.

Tornando al nostro esempio, il valore finale per ECX sarà
2345678FH
mentre il registro ESI rimarrà inalterato.

Tra l'altro se si desiderasse avere anche ESI shiftato di 4 bit, per mantenerlo in passo con ECX, allora basterà eseguire l'istruzione
shl esi,4
come è ovvio...

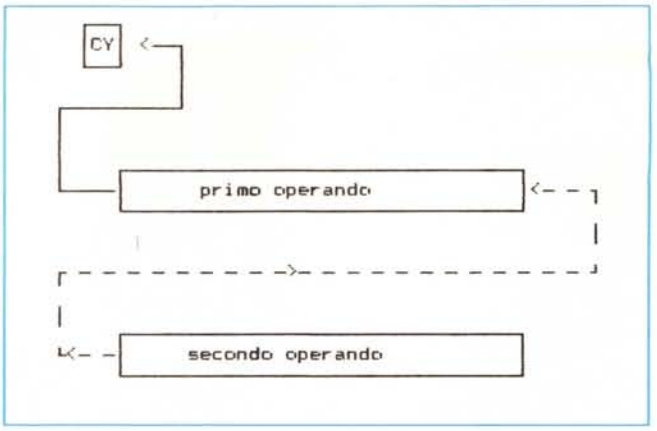
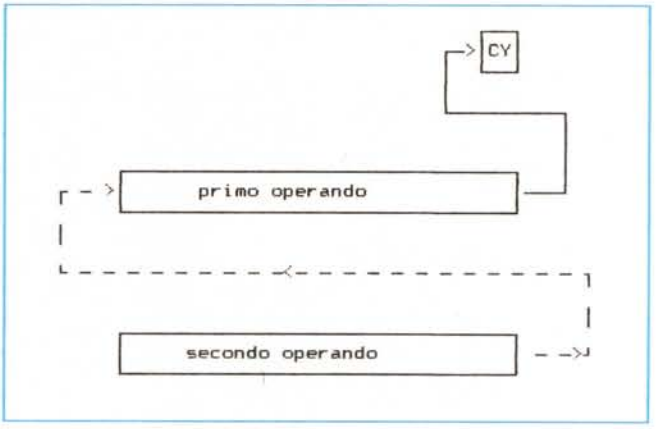


*Figura 1 - Disegno schematico delle operazioni svolte dalla funzioni **SHLD**: il primo operando viene shiftato a sinistra di «contatore» posizioni, ogni volta inserendo come bit meno significativo il bit proveniente dallo shift verso sinistra del secondo operando, che però rimane sempre inalterato.*

*Figura 2 - Disegno schematico delle operazioni svolte dalla funzione **SHRD**: il primo operando viene shiftato a destra di «contatore» posizioni, ogni volta inserendo come bit più significativo il bit proveniente dallo shift verso destra del secondo operando, che però rimane sempre inalterato.*



Lo stesso discorso si può ripetere pari pari per l'istruzione **SHRD**, sostituendo tutti i riferimenti a «destra» con «sinistra» e viceversa (si veda in merito la figura 2): anche in questo caso il secondo operando rimarrà inalterato.

In definitiva dunque la sintassi delle due istruzioni è la seguente:

```
shld reg1,reg2,contatore
shrd reg1,reg2,contatore
```

dove appunto **reg1** può essere o un registro o una locazione di memoria a 16 o 32 bit, mentre **reg2** potrà essere solo un registro (con un numero di bit pari a quelli del primo operando); infine **contatore** potrà essere un valore immediato ad 8 bit oppure il registro CL: in entrambi i casi, analogamente a quanto succede per tutte le altre istruzion di shift, il valore «contatore» verrà preso comunque «modulo 32».

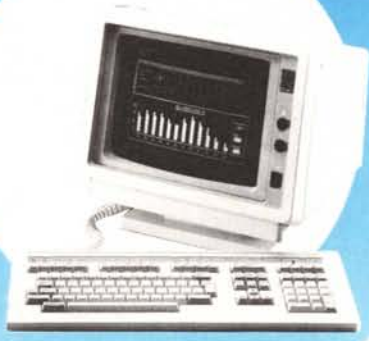
Nella tabella a fianco riportiamo i soliti esempi di istruzioni che in questo caso abbiamo volutamente ingigantito per ciò che riguarda le locazioni di memoria, ricordando che esistono parecchi modi di indirizzamento.

Con questo terminiamo la puntata e diamo appuntamento alla prossima, nella quale continueremo l'analisi delle istruzioni del 386. MC

Esempio di istruzioni in Assembler 80386

```
0F  A4  DA  0B                    shld dx,bx,11
0F  A5  DA                        shld dx,bx,cl
66  0F  A4  DA  0B                shld edx,ebx,11
66  0F  A5  DA                    shld edx,ebx,cl
66  67  0F  A5  B4  AE  00000002  shld beta[esi+ebp*4],esi,cl
66  67  0F  A4  20  16            shld dword ptr [eax],esp,22
67  0F  A5  0C  13                shld word ptr [ebx + edx],cx,cl
0F  AC  DA  0B                    shrd dx,bx,11
0F  AD  DA                        shrd dx,bx,cl
66  0F  AC  DA  0B                shrd edx,ebx,11
66  0F  AD  DA                    shrd edx,ebx,cl
66  67  0F  AD  B4  AE  00000002  shrd beta [esi+ebp*4],esi,cl
66  67  0F  AC  20  16            shrd dword ptr [eax],esp,22
67  0F  AD  0C  13                shrd word ptr [ebx + edx],cx,cl
```

parliamo di videoterminali a colori



Cresce l'uso dei videoterminali a colori

La notevole diffusione di programmi per PC muniti di monitors a colori ha abituato l'utente a farsi aiutare dal colore e dalla grafica per mettere in evidenza le informazioni sullo schermo.

Al punto da far diventare il monitor a colori un bene irrinunciabile. Con la diffusione dei programmi a colori per sistemi operativi multiutenza Unix e Xenix, aumenta anche la diffusione dei videoterminali a colori. La Microvitec, conscia di ciò, ha lanciato sul mercato i Microcolour, una serie di videoterminali a colori ANSI compatibili.

Ora anche l'utente di videoterminali può farsi aiutare dal colore per organizzare meglio i suoi dati sullo schermo.

Il terminale alfanumerico M2100 è oggi sicuramente il più economico sul mercato, al punto da poter competere con molti terminali monocromatici. Il modello M3220TV offre prestazioni superiori ed è totalmente compatibile VT320. Il modello M4305MU, con grafica compatibile Tektronix 4105, dispone di un kit che gli permette di essere usato con i più diffusi Office Automation in circolazione.

Lo standard di costruzione denota la solida tecnologia posseduta dal costruttore britannico. Del resto la Microvitec, maggior costruttore europeo di videoterminali e monitors a colori, ha sempre offerto prodotti sicuri. Ma come è tradizione nostra abbiamo proceduto anche questa volta ad una attenta verifica di ogni unità consegnata. E questo per avere più della sicurezza.

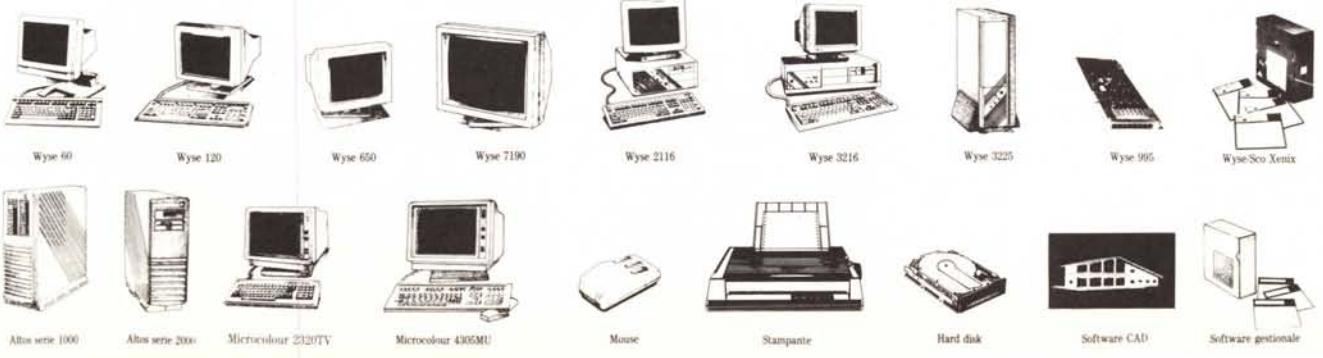
Ecco perchè continuiamo a dire che tutto ciò che è distribuito dalla Ready Informatica è **informatica pronta.**

MICROVITEC

Via Provinciale, 67
22068 Monticello Brianza
Tel. (039) 9202108
Fax (039) 9206738

Ready INFORMATICA

Milano Tel. (02) 26410625
Verona Tel. (045) 7155590
Firenze Tel. (055) 319321
Roma Tel. (06) 4386886



Wyse 60 · Wyse 120 · Wyse 650 · Wyse 7190 · Wyse 2116 · Wyse 3216 · Wyse 3225 · Wyse 995 · Wyse Sco Xenix

Altos serie 1000 · Altos serie 2000 · Microcolour 2320TV · Microcolour 4305MU · Mouse · Stampante · Hard disk · Software CAD · Software gestionale