di Pierluigi Panunzi

Il set di istruzioni

A partire da questa puntata inizieremo l'analisi del set di istruzioni del 386, con particolare riguardo innanzitutto alle istruzioni introdotte proprio con l'80386 e poi in seconda battuta agli effetti, sulle «vecchie» istruzioni, delle innovazioni del microprocessore, quali i nuovi indirizzamenti e l'utilizzazione di quantità (registri e memoria) a 32 bit.
Prima però di iniziare tale analisi, ritorniamo un istante a quel piccolo frammento di programma di cui abbiamo parlato nella scorsa puntata e che riportiamo in figura 1

Quantità a 16 o a 32 bit

Abbiamo detto più volte che il 386 può manipolare quantità a 16 (e sottintendiamo con questa dizione anche 8 bit, come siamo abituati con l'8086 e l'80286) nonché a 32 bit: queste quantità possono essere registri (i registri «Extended») oppure locazioni di memoria, quelle definite con una direttiva DD («Define Doubleword») o a cui si fa riferimento con «DWORD PTR».

Per non essere costretti a riprogettare daccapo il set di istruzioni, che avrebbe comportato la completa incompatibilità verso il basso, i progettisti dell'Intel hanno pensato di creare due nuove entità logiche che indicano la presenza di quantità a 32 bit: la prima entità riguarda un bit all'interno del descrittore del segmento interessato e l'altra entità è una coppia di prefissi, che perciò nella codifica dell'istruzione appaiono prima del codice operativo dell'istruzione stessa. Il bit in questione è un nuovo campo («D») di un segment descriptor (sul quale ritorneremo fra un po' di tempo...) che indica, se settato, la presenza di quantità a 32 bit: in particolare se si tratta di un «code segment descriptor» allora significa che tutte le istruzioni in esso contenute faranno riferimento per default ai registri estesi a 32 bit, mentre se si tratta di un «data segment descriptor» allora ciò significa che tutti i dati presenti in quel segmento sono per default a 32 bit.

Viceversa (anche se ciò ora sembra lapalissiano, subito dopo se ne comprenderà il motivo...) il bit «D» posto a 0 significa che per default le istruzioni (nel caso del code segment) oppure i dati (nel caso di un data segment) sono a 16 bit.

A tal proposito, il settaggio o meno di questo bit rientra nelle operazioni da compiere prima di entrare in Modo Protetto e perciò sarà compito delle apposite routine di sistema eseguite al massimo livello di privilegio: dal punto di vista dell'utente invece ci si troverà in un ambiente per default a 32 oppure a 16 bit.

Ecco perciò che il significato dei due nuovi prefissi è analogo a quello di tanti altri prefissi e cioè quello di «override» (cambiamento di ciò che è settato per default): in particolare il prefisso 67H comparirà laddove si faccia uso di registri con un numero di bit diverso da quello del default (si usano registri estesi in ambiente di codice a 16 bit che è il caso dell'esempio, oppure registri a 16 bit in un ambiente «puro 386» a 32 bit) ed analogamente accadrà per l'altro prefisso (66H) stavolta relativo a quantità poste in memoria.

Anche per quanto riguarda quest'ultimo prefisso, nell'esempio (relativo ad un ambiente a 16 bit) vediamo che il 66H appare tutte le volte in cui si fa riferimento a quantità a 32 bit (la variabile ALFA).

A titolo di curiosità riportiamo in figura 2 il disassemblato dello stesso programma di figura 1, però ottenuto con il ben noto DEBUG, che viceversa non riconosce i codici operativi (e tantomeno i prefissi) del 386: a parte infatti i vari 66H e 67H che in ogni modo vengono «ignorati» dal disassemblato con altrettante DB («Define Byte»), si possono vedere delle differenze tra come vengono interpretate le istruzioni nell'8086 rispetto al loro significato del 386.

Nella prima istruzione vediamo che viene effettuato l'incremento della word posta all'offset ABCDH: correttamente nel 386 il passaggio tra la word e la doubleword avviene semplicemente anteponendo il prefisso 66H all'opcode.

Ma già nella seconda istruzione vediamo che il tutto non è così semplice e soprattutto gratuito ed automatico: il codice operativo FFH 03H nell'8086 corrisponde a

INC WORD PTR [BP + DI]

mentre col prefisso 67H (registri estesi) non significa

INC WORD PTR [EBP] [EDI]

ma bensì

```
1   0000                            .MODEL TPASCAL
2                                   .386
3   0000                            .DATA
4                                           ORG 0ABCDH
5   ABCD  ????????                  ALFA    DD ?
6   ABD1                            .CODE
7   0000  66: FF 06 ABCDr           START:  INC ALFA
8   0005  67: FF 03                         INC WORD PTR [EBX]
9   0008  67: FF 83 00004FF2                INC WORD PTR [EBX + 4FF2H]
10  000F  66: 67: 03 92         +           ADD EDX,ALFA[EDX]
11        0000ABCDr
12  0017  66: 67: 8B 0C 5D      +           MOV ECX,[EBX * 2]
13        00000000
14  0020  66: 67: 8B 1C 1B                  MOV EBX,[EBX][EBX]
15  0025  66: 67: 8B 1C 5B                  MOV EBX,[EBX * 2][EBX]
16  002A  67: C7 04 83 0001                 MOV WORD PTR [EBX][EAX * 4],1
17  0030  67: C7 04 FE 0002                 MOV WORD PTR [EDI * 8][ESI],2
18  0036  66: 67: FF 8C 99      +           DEC ALFA[EBX * 4][ECX + 1000H]
19        0000BBCDr
20
21                                          END START
```

Figura 1 - Listato del programmino di prova proposto la scorsa puntata, scritto in TASM (Turbo Assembler) con lo scopo di vedere come vengono codificate le istruzioni che utilizzano i nuovi modi di indirizzamento del 386 e le quantità a 32 bit.

```
0000   66              DB 66
0001   FF06CDAB        INC WORD PTR [ABCD]
0005   67              DB 67
0006   FF03            INC WORD PTR [BP+DI]
0008   67              DB 67
0009   FF83F24F        INC WORD PTR [BP+DI+4FF2]
000D   0000            ADD [BX+SI],AL
000F   66              DB 66
0010   67              DB 67
0011   0392CDAB        ADD DX,[BP+SI+ABCD]
0015   0000            ADD [BX+SI],AL
0017   66              DB 66
0018   67              DB 67
0019   8B0C            MOV CX,[SI]
001B   5D              POP BP
001C   0000            ADD [BX+SI],AL
001E   0000            ADD [BX+SI],AL
0020   66              DB 66
0021   67              DB 67
0022   8B1C            MOV BX,[SI]
0024   1B6667          SBB SP,[BP+67]
0027   8B1C            MOV BX,[SI]
0029   5B              POP BX
002A   67              DB 67
002B   C7048301        MOV WORD PTR [SI],0183
002F   0067C7          ADD [BX-39],AH
0032   04FE            ADD AL,FE
0034   0200            ADD AL,[BX+SI]
0036   66              DB 66
0037   67              DB 67
0038   FF8C99CD        DEC WORD PTR [SI+CD99]
003C   BB0000          MOV BX,0000
```

Figura 2 - Disassemblato (in 8086) del programmino di figura 1 ottenuto con il DEBUG, che non riconosce il 386 e perciò le sue istruzioni ed i prefissi.

INC WORD PTR [EBX]

Questo fatto che la scrittura [BP + DI] con il prefisso si trasforma in [EBX] si vede pure nella terza istruzione...

Per chi fosse interessato diciamo che il codice operativo dell'istruzione

INC WORD PTR [EBP] [EDI]

è dato (a parte i prefissi) da FFH 44H 3DH 00H e perciò proprio tutta un'altra cosa...

Saltiamo all'istruzione

MOV EBX,[EBX] [EBX]

il cui codice operativo, a parte i prefissi, è 8BH 1BH 1CH e cioè un opcode a tre byte: trasportato all'8086 invece vediamo che 8BH 1BH generano un'istruzione

MOV BX,[SI]

che non è nemmeno lontana parente ed ancor peggiore è il fatto che quell'1CH rimasto va a combinarsi con i prefissi successivi per creare altre istruzioni che non c'entrano niente.

Da ciò possiamo dunque trarre due indicazioni importanti:

— le istruzioni del 386 non sono (e non dovevano essere!) compatibili verso il basso (8086 e 80286), per cui riteniamo superfluo sconsigliare la prova di esecuzione di tale frammento di programma non in ambiente 386.

— Gli opcode delle istruzioni del 386 sono costruiti solo in prima analisi a partire da quelli dei microprocessori precedenti: in presenza di prefissi i codici operativi (che possono già, come visto, essere a 3 byte) vengono generati seguendo regole molto più complesse di quelle che governano i micro precedenti. Siccome non ne abbiamo parlato a suo tempo (se non per brevi cenni) per nessuno dei due predecessori, a maggior ragione non ne parleremo in questa sede, soprattutto per la grande complessità intrinseca.

Per quanto riguarda il primo punto c'è da aggiungere che il tentativo di eseguire semplicemente il codice 66H della prima istruzione, su di un AT, è catastrofico e deludente allo stesso tempo: non è che si hanno effetti visivi, beep o altro... ma semplicemente il computer rimane bloccato...

D'altronde bisogna ricordarsi che il valore 66H, per il microprocessore 80286, è un «Invalid Opcode», per cui ben difficilmente ci si può aspettare qualcosa di buono: per quanto riguarda l'8086 (o 8088) non abbiamo fatto prove in merito e lasciamo il compito ai lettori interessati.

Fatta dunque questa doverosa premessa, sulla quale per forza di cose torneremo più volte, iniziamo ad analizzare il set di istruzioni del 386, partendo dalle istruzioni generalmente utilizzabili dall'utente: per quanto riguarda le istruzioni privilegiate possiamo dire che ne parleremo fra non poco.

```
cs:0100 0FBEC3          movsx   ax,bl
cs:0103 660FBEDD        movsx   ebx,ch
cs:0107 660FBFFE        movsx   edi,si
cs:010B 0FB6C3          movzx   ax,bl
cs:010E 660FB6DD        movzx   ebx,ch
cs:0112 660FB7FE        movzx   edi,si
```

Figura 3 - Esempio di nuove funzioni del 386, ottenute con il fenomenale Turbo Debugger: dalla codifica si nota la ricca presenza di prefissi 66H di override.

Le istruzioni di trasferimento dati

Con tale termine già sappiamo che si intendono quelle istruzioni che, in un modo o nell'altro, effettuano uno spostamento fisico di informazioni (dati) da un luogo all'altro ed in particolare dai registri del microprocessore alla memoria e viceversa. In particolare per «memoria» si intenderà non solo la memoria dati ma anche, perché no?!, lo stack.

Iniziamo subito dalla **MOV**, non certo per parlare della sua semantica, ma per vedere che questa istruzione prevede il caricamento di locazioni e/o registri a 32 bit, come naturale estensione delle istruzioni dei microprocessori precedenti.

Ecco che perciò all'istruzione

MOV BX,BETA

si affiancherà l'omologa

MOV EBX,GAMMA

dove sappiamo che EBX è l'«Extended BX» e GAMMA è una double word posta in memoria.

Analoghi parallelismi sono facili da immaginare, mentre veramente nuove sono le istruzioni che si occupano del caricamento dei registri di segmento: sappiamo che possiamo utilizzare due nuovi registri di segmento (FS e GS) e perciò sarà possibile caricarne il contenuto con apposite istruzione di MOV, una delle quali è ad esempio

MOV FS,ALFA

Accanto a queste istruzioni di MOV ne esistono altre particolari, che servono a caricare in modi nuovi i registri a 16 e 32 bit: si tratta del «caricamento con estensione del segno» (**MOVSX**, «MOVe Sign eXtended») e del «caricamento con estensione dello zero» (**MOVZX**, «MOVe Zero eXtended»).

In particolare si tratta di istruzioni che consentono di estendere una quantità ad 8 bit verso una a 16 o 32 bit, nonché una quantità a 16 bit verso una a 32 bit, sia mantenendo il segno della quantità originale, sia trascurandolo e forzandolo a zero.

A tal proposito in figura 3 abbiamo portato l'esempio delle sei possibilità offerte dalle MOVSX e MOVZX: nella

codifica vediamo ancora una volta presentarsi il prefisso 66H che a questo punto conosciamo molto bene.

Ad esempio se il registro CH contiene il valore 86H, allora dopo una

MOVSX EBX,CH

il registro EBX conterrà il valore 0FFFFFF86H e cioè il valore 86H (che ad 8 bit possiede il bit più significativo posto ad 1 e che rappresenta il valore −122) viene riportato come quantità a

```
cs:0100 50          push   ax
cs:0101 6650        push   eax
cs:0103 6A01        push   0001
cs:0105 683412      push   1234
cs:0108 06          push   es
cs:0109 0F          db     0F
cs:010A A00FA8      mov    al,[A80F]
cs:010D 60          pusha
cs:010E 6660        pushad
cs:0110 58          pop    ax
cs:0111 6658        pop    eax
cs:0113 07          pop    es
cs:0114 0F          db     0F
cs:0115 A10FA9      mov    ax,[A90F]
cs:0118 61          popa
cs:0119 6661        popad
```

Figura 4 - Esempio che riassume le istruzioni PUSH e POP estese e non: per l'inaspettata coppia di istruzioni disassemblate con «db 0F» si veda il testo.

32 bit che mantiene il valore di −122.

Invece con una

MOVZX EBX,CH

il valore finale contenuto in EBX sarà 00000086H (abbreviabile senz'altro con 86H!), che viceversa vale 134. Interessante è il fatto che queste istruzioni, caso alquanto raro, possiedono due operandi di tipo e grandezza differente, quasi a contravvenire la viceversa ferrea legge che impone uno stretto controllo delle quantità su cui si opera: ad esempio un'istruzione di caricamento di un registro a 16 bit con uno da 8 è vietata...

Per quanto riguarda le istruzioni di gestione dello stack (PUSH e POP, semplici e generalizzate), vediamo in figura 4 un altro frammento di programma contenente una miscellanea delle possibilità di istruzioni, nuove oppure estensioni di vecchie.

Diciamo subito che abbiamo scoperto un «bug» all'interno del Turbo Debugger, che lo porta a disassemblare in modo errato alcune istruzioni: in particolare i codici operativi 0FH A0H e 0FH 0A8H appartengono in realtà alle istruzioni **PUSH FS** e **push gs**, come pure i codici 0FH 0A1H e 0FH 0A9H appartengono alle duali **POP FS** e **POP GS**: peccato...

C'è da menzionare che accanto alle **PUSHA** e **POPA**, introdotte a partire dall'80186, appaiono le versioni estese e cioè **PUSHAD** e **POPAD** (rispettivamente «PUSH All Doubleword» e «POP All Doubleword») le quali rispettivamente salvano e ripristinano il set completo di registri a 32 bit anziché quelli a 16 bit usuali.

Altre istruzioni analoghe alle precedenti sono le **PUSHF** e **POPF** con le loro «estese» a 32 bit che sono le **PUSHFD** e **POPFD**.

Per quanto riguarda l'istruzione **XCHG**, oltre a citare le ovvie estensioni a 32 bit di quanto ben conosciamo, c'è da segnalare un fatto nuovo, implementato nel 386 durante l'esecuzione di tale istruzione: in particolare quando uno dei due operandi è una cella di memoria, allora viene reso attivo il segnale LOCK del microprocessore, indipendentemente dalla presenza o meno del prefisso «LOCK» (l'istruzione presente già nell'8086), per ottenere la «chiusura» del BUS ad altri eventuali processori presenti nel sistema. Comunque torneremo su questo argomento quando sarà più opportuno.

Altre istruzioni di trasferimento dati sono quelle che consentono di caricare direttamente un puntatore (quantità a 32 bit) in una coppia di registri, un registro di segmento per la parte più significativa ed un altro registro per quella meno significativa.

Si tratta delle istruzioni **LES** e **LDS**, che hanno in questo caso delle interessanti estensioni: in particolare, partendo dalle istruzioni.

```
LES AX,DWORD PTR ALFA
LDS BX,BETA
```

ecco apparire le

```
LES EAX,DWORD PTR ALFA
LDS EBX,BETA
```

nonché un'altra serie di istruzioni che coinvolge i registri di segmento FS, GS nonché SS: anche per questi tre casi si hanno i due sottocasi in cui l'altro registro è a 16 oppure a 32 bit.

Comunque in tutti questi casi è interessante scoprire il funzionamento nel caso che il secondo registro sia un registro esteso.

Vediamo ad esempio cosa accade a seguito delle istruzioni

```
LGS BX,ALFA
LFS EAX,ALFA
```

dove i byte, a partire dalla cella identificata da ALFA, valgono

11H 22H 33H 44H 55H 66H 77H...

Ecco che, subito dopo la prima istruzione, il registro BX conterrà il valore 2211H ed il registro GS conterrà il valore 4433H; invece, dopo la seconda istruzione, il contenuto di EAX (a 32 bit) varrà 44332211H, mentre il contenuto del registro FS sarà 6655H: perciò nel caso normale viene caricato un valore a 16 bit nel registro ed un valore a 16 bit nel registro di segmento, per un totale di 4 byte a partire dalla cella interessata dall'istruzione, mentre nel secondo caso viene caricato un valore a 32 bit nel registro ed un valore a 16 bit nel registro di segmento, per un totale di 6 byte a partire dalla cella, con un comportamento a prima vista alquanto strano.

Altra istruzione che rappresenta un'estensione di una già nota è l'istruzione **LEA** («Load Effective Address») con la quale si può caricare sia un registro a 16 bit sia un registro esteso con l'indirizzo a 16 o 32 bit di una determinata cella di memoria.

In particolare, accanto al caricamento di un indirizzo a 16 bit in un registro a 16 bit (funzione ben nota) c'è il caricamento di tale valore in un registro esteso, nel qual caso la parte più significativa del registro in esame verrà posta a 0.

Viceversa il caricamento di un indirizzo a 32 bit di una cella di memoria avviene senza problemi se il registro di destinazione è uno esteso (e perciò a 32 bit), mentre nel caso in cui il registro sia a 16 bit allora solo parte meno significativa dell'indirizzo potrà essere caricata.

Anche riguardo questa istruzione dobbiamo sottolineare (quasi con incredulità) la presenza di un altro bug all'interno del meraviglioso Turbo Debugger il quale codifica in maniera strana istruzioni del tipo

LEA EAX,[EBX]

aggiungendo offset che non c'entrano niente: in particolare capita che impostando l'istruzione al volo (così come nel caso già citato dell'istruzione PUSH FS), non si ottenga viceversa un'istruzione debuggata identica, ma un qualcosa di strano... Attendiamo fiduciosi la versione 2 di tale meraviglioso programma...

Come ultima coppia di istruzioni di trasferimento dati che analizziamo, parliamo delle istruzioni **IN** ed **OUT**, che tutto sommato non hanno risentito molto dell'estensione a 32 bit, se non per il registro che riceve o invia il dato attraverso la porta, che in questo caso, oltre che AL ed AX, può essere anche EAX, fermo restando il fatto che le porte possono essere indirizzate o in modo immediato (valori tra 0 e 255) oppure per mezzo del registro DX (da 0 a 65535).

MC

INTERNATIONAL SHOW ROOM

COMMUNICATION

3Com

3Plus Share
3Plus Open

Software operativi di rete in ambiente MS/DOS e OS/2. In entrambi gli ambienti operativi sono disponibili moduli per la gestione di posta elettronica e collegamenti remoti.

3S400
3Station

Server di rete basato su CPU 386, Workstation diskless

Etherlink
Tokenlink

Adattori Ethernet e Token Ring

ALCOM

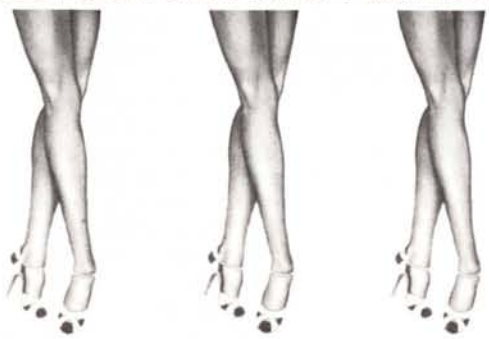
LAN FAX/10
Easy Gate

Gateway di comunicazione per il collegamento di reti locali verso Telex, Fax e sistemi pubblici di posta elettronica.

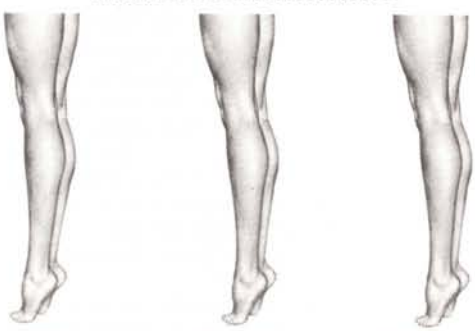
Rabbit SOFTWARE

Rabbit Station
Rabbit Gate

Emulatori IBM 3270 per PCstand alone o collegati in rete locale. Disponibili in versione Remote, DFT, X.25.



CAD/CAM



Cadkey

Software di progettazione CAD 3D con modellazione geometrica di tipo wireframe.
Quotatura automatica standard ANSI-ISO.

CADKEY 3

Mastercam

Modulo CAD/CAM integrato per controllo Frese, Torni, EDM fino a tre assi. Post-processor standard ISO.

CNC Software Inc.

- Schede grafiche ad alta risoluzione 1280x1024 16 o 256 colori
- Compatibili con i software di grafica e CAD più usati: Cadkey, Autocad, Personal Designer, Dr. Halo.

IMAgraph

ADD-ON E PERIFERICHE

LOGITECH

Mouse in versione seriale, bus e per PS/2 risoluzione standard 320 dpi.

ScanMan

Hand scanner in versione standard bus, MicroChannel e Macintosh, risoluzione fino a 400 dpi.

Finesse

Pacchetto software di Desk Top Publishing.

Jasmine

DirectDrive - DirectPrint
DirectServer

Hard disk esterni e removibili per Macintosh PLUS/SE/II.
Laser printer 300 dpi, 3Mb RAM, compatibile Post-script.
AppleShare file server per reti Macintosh Local Talk compatibile AFP.

SOTA

Schede Super VGA, risoluzione massima 1024x768 16 colori

Schede acceleratrici compatibili con IBM PS/2 mod. 30, PC, XT Compaq Desk-Pro, Olivetti M24 e con la maggior parte delle macchine 8088 e 8086 con clock fino a 10 Mhz.

DATA BASE SOFTWARE

Fox Software

Fox Base +
Fox Base + Mac

Pacchetto di Data-Base relazionale compatibile con dBase III Plus.
Disponibile sia in versione MS/DOS che Macintosh.
Esiste in Versione Single User e Multiuser, 386 MS/DOS, nonché il modulo di Run Time per un numero illimitato di installazioni.

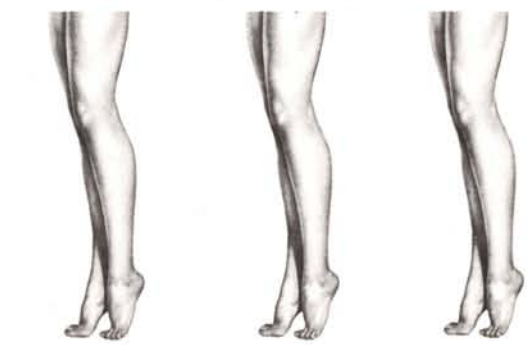
Nantucket

Clipper

Il compilatore per dBase III Plus.
Un ambiente di sviluppo completo, aggiunge più di 30 funzioni al linguaggio dBase III, comprende un efficiente debugger, permette di agganciare routine scritte in C o in Assembler.

Sycero

Generatore di programmi in linguaggio dBase III/Clipper.



PROTEZIONI SOFTWARE

LOGIKEY

Dispositivo per la protezione del software. Si applica sulla porta seriale, non inibendone comunque l'utilizzo. Viene personalizzato per ogni cliente.

Fermatevi un attimo davanti a una vetrina di prodotti che non conosce frontiere geografiche. Apprezzerete hardware e software selezionati fra le migliori firme internazionali. Novità esclusive assolutamente in linea con le esigenze del mercato italiano. Una collezione di prodotti che abbina tecnologia e prezzo all'internazionalità dell'esperienza Algol. Troverete professionalità, competitività e risparmio. Fermatevi ancora un attimo: il nostro servizio di telemarketing è a vostra disposizione per parlarvi di soluzioni ma anche di prezzi, avviamento e assistenza. Un consiglio: tuffatevi.

MILANO: Via Feltre 28/6 - Tel. 02/26411411 (r.a.) - Fax 02/2154629
FILIALI: Roma Tel. 06/5913971-5919479 - Fax 06/5918745 - Ivrea Tel. 0125/424541-2
CENTRO SVILUPPO PROGETTI E SELEZIONE PRODOTTI: Palo Alto (California) USA Tel. 001 415 493-3800

ALGOL