ASSEMBLER 8086 8088

di Pierluigi Panunzi

Il set di istruzioni
Istruzioni logiche

Seconda parte

Terminiamo in questa puntata l'analisi del gruppo di istruzioni che consentono di effettuare le operazioni logiche su operandi ad 8 o a 16 bit, operazioni che generalmente alterano alcuni dei flag principali.
La scorsa puntata abbiamo visto le istruzioni di «shift» e di «rotate» le quali, insieme alla prossima istruzione che analizzeremo (la NOT), agiscono su di un solo operando, che è perciò contemporaneamente sorgente e destinazione dell'istruzione stessa.
Anche se non l'abbiamo esplicitamente detto, dovrebbe essere chiaro che effettuando un'operazione su di una cella di memoria, in realtà l'operazione viene effettuata su di un registro interno («temporaneo»), preventivamente caricato con il contenuto della cella stessa: al termine dell'operazione il risultato verrà poi ritrasferito alla cella in questione.
Avevamo tralasciato questo «dettaglio» in quanto è prassi comune di tutti i microprocessori effettuare le operazioni all'interno della CPU, in quanto ciò risulta effettivamente più rapido, mentre l'accesso alla memoria (in generale più lento) avviene solo allorché sia necessario.

L'istruzione NOT

Si tratta ovviamente dell'istruzione che consente di ottenere il completamento ad 1 dell'operando, invertendone uno ad uno i bit (8 o 16).

Questa operazione, se vogliamo la più semplice e banale tra quelle logiche, viene effettuata all'interno del microprocessore in una maniera alquanto strana: a seconda se l'operando è ad 8 o a 16 bit il nostro 8086/88 si prepara un primo operando pari rispettivamente a 0FFH e 0FFFFH, al quale andrà a sottrarre l'operando fornito dall'istruzione stessa.

Il risultato così ottenuto verrà poi depositato all'interno dell'operando (registro o memoria che sia).

Analogamente strano in questo caso è il fatto che nessun flag viene alterato, nemmeno (ad esempio) il flag di Zero nel caso che il risultato diventi nullo.

Non riteniamo di dover fare alcun esempio di funzionamento di tale istruzione in quanto si tratta della prima istruzione logica che in genere si impara: viceversa ricordiamo che il formato generico di questa istruzione è

```
NOT operand
```

dove «operand» può essere sia un registro, sia una locazione di memoria, indirizzata sia in modo diretto che in modo indiretto, basato, indicizzato e basato-indicizzato, come ben sappiamo.

Alcuni esempi di tale istruzione sono dunque i seguenti:

```
NOT BX
NOT [BX]
NOT ALFA[SI+5]
NOT SI
```

È da notare che l'istruzione NOT non può operare sui registri di segmento, ma questa è un'affermazione probabilmente superflua in quanto non crediamo che mai possa venire in mente di invertire il contenuto (ad esempio) del Data Segment register (DS), se non per darsi la classica zappa sui piedi ...

Le istruzioni logiche a due operandi

Appartengono a questo gruppo 5 istruzioni, anche queste ben note (in quanto sono sempre le stesse!) che per l'appunto possono operare su operandi a coppia a 16 o ad 8 bit, ma non su un operando ad 8 bit e l'altro a 16 bit.

Tutte e cinque le istruzioni delle quali parleremo (nell'ordine AND,

TEST, OR, XOR e CMP) agiscono sui seguenti operandi:

1° operando	2° operando
registro	registro
memoria	registro
registro	memoria
accumulatore	immediato
registro	immediato
memoria	immediato

dove valgono le solite indicazioni riguardo i termini indicati in tabella e cioè:

— «registro» è un qualsiasi registro della CPU ad eccezione dei Segment Register.

— «memoria» è una qualsiasi locazione indirizzata direttamente, indirettamente, tramite i registri base o quelli indice oppure a tutti e due.

— «accumulatore» è AX (se il secondo operando è a 16 bit), oppure AL se il secondo operando è a 8 bit: ancora una volta si nota che l'accumulatore, come registro interno su cui effettuare i calcoli, è per antico retaggio il più privilegiato tanto è vero che viene generato un codice operativo più corto che non nel caso di un registro generico.

— «immediato» è un valore costante ad 8 o a 16 bit, conformemente al primo operando.

Analizziamo dunque una per una le cinque istruzioni logiche.

L'istruzione AND

L'istruzione in esame effettua l'AND tra i due operandi, bit a bit, ovviamente dando per risultato un bit ad «1» laddove entrambi gli operandi hanno un bit posto ad «1» ed un bit nullo negli altri casi.

Per quanto riguarda i flag, c'è da dire che vengono resettati il Carry ed il flag di Overflow, mentre la Parità, il Segno e lo Zero vengono alterati a seconda del risultato dell'operazione stessa: in generale dopo un'AND il flag che viene testato è il flag di Zero, per andare a «saggiare» il risultato del mascheramento di alcuni bit di un certo operando.

Dimenticavamo di aggiungere che il risultato dell'AND viene posto sempre nel primo operando, che perciò viene alterato: questo lo diciamo in quanto con l'istruzione AND si ha il mascheramento distruttivo dell'operando.

L'istruzione TEST

È l'istruzione che ci vuole nel caso in cui non desideriamo il mascheramento distruttivo dell'operando: infatti l'istruzione TEST è in tutto e per tutto analoga all'istruzione AND, con l'unica notevole differenza che l'AND viene effettuato in un registro interno, senza alterare in alcun modo il primo operando.

È dunque molto utile in quei casi in cui un registro, per esempio AL, contiene un valore variabile in base al quale effettuare una di un certo numero di operazioni ed ancora più in particolare in quei casi in cui è determinante la presenza di un certo determinato bit per far eseguire una routine piuttosto che un'altra: è ovvio che in ogni caso il primo test della sequenza «in cascata» non dovrà alterare il registro in esame e così pure il secondo e così via, in modo tale da poter effettuare l'«ultimo» confronto della catena con il registro immutato.

Bisogna però ricordare che l'istruzione TEST effettua l'AND tra i due operandi per cui bisogna stare attenti che uno stesso risultato si può ottenere (a parità di maschera) con parecchie combinazioni dell'operando.

Ad esempio dovendo testare se il byte contenuto in AL valga «Y» o «N» (ad esempio la risposta ad un prompt) rispettivamente pari a 59H e 4EH, non è cosa saggia scrivere un programma del tipo:

```
....
MOV AL,ANSWER
TEST AL,59H
JZ YES_ANSWER
TEST AL,4EH
JZ NO_ANSWER
JMP RETRY
....
```

in quanto ad esempio una risposta del tipo «]» (50DH in esadecimale) verrebbe interpretata come una risposta «Y», come è facile verificare.

In questi casi conviene dunque sfruttare altri procedimenti: ad esempio si può sfruttare l'istruzione CMP (come vedremo nel seguito) oppure ad esempio, in previsione di test distruttivi, si può salvare il registro «importante» da qualche parte per poterlo ripristinare nel caso in cui il primo confronto non abbia sortito esito positivo.

L'istruzione CMP

Questa istruzione effettua la «CoMPare» (comparazione) tra i due operandi e ne consente così il confronto ed è perciò utilissima in quei casi in cui è determinante l'ordine di grandezza di una certa quantità rispetto ad un'altra, quei casi in cui si testa se un operando è «maggiore», «minore», «maggiore o uguale», ecc.

L'istruzione CMP in particolare effettua la comparazione tra i due operandi sottraendo il secondo dal primo, senza calcolare il risultato (nel senso che la sottrazione viene effettuata in un registro temporaneo interno) e perciò senza alterare i due operandi, ma settando opportunamente i flag (Zero, Carry, Parity, Sign, Overflow e Auxiliary) in base al risultato della sottrazione.

Alla luce di quanto detto per l'istruzione in esame si ha che in particolare il nostro frammento di programma potrebbe essere scritto in questa maniera:

```
....
MOV AL,ANSWER
CMP AL,59H
JZ YES_ANSWER
CMP AL,4EH
JZ NO_ANSWER
JMP RETRY
....
```

dove abbiamo abilmente sfruttato il fatto che la CMP non altera gli operandi.

C'è da aggiungere per questa istruzione un'ulteriore caratteristica: in particolare nel caso di «compare» tra un registro ed un operando immediato, quest'ultimo può essere ad 8 bit anche nel caso che il registro sia a 16 bit, caso alquanto raro di convivenza di dati ad 8 e a 16 bit in un'unica istruzione, cosa di solito non gradita dall'Assembler.

Scendendo ancora più in dettaglio, nel caso in cui il primo operando è un registro o una locazione di memoria a 16 bit ed il secondo operando è un byte immediato, allora quel byte, prima di effettuare la comparazione (sottrazione), viene in alcuni casi esteso con il segno a 16 bit: se il byte poteva stare in un byte solo, allora il codice operativo generato dall'assemblatore sarà differente che non nel caso in cui il byte immediato deve essere per forza codificato con una word.

Abbiamo a tal proposito buttato giù un programmino (!) di esempio in cui abbiamo confrontato il registro BX con vari valori immediati:

```
0000                    NAME PROVA
                        CODE SEGMENT
0000  83 FB 01          CMP  BX,1
0003  83 FB FF          CMP  BX,-1
0006  83 FB 33          CMP  BX,33H
0009  83 FB CD          CMP  BX,-33H
000C  81 FB 0081        CMP  BX,129
0010  81 FB 00FF        CMP  BX,0FFH
0014  81 FB FF01        CMP  BX,-0FFH
0018  81 FB 0080        CMP  BX,80H
001C                    CODE ENDS
                        END
```

È istruttivo analizzare il modo in cui viene codificata la singola istruzione a partire da valori apparentemente casuali e innocui, e vedere come l'Assembler si preoccupa di soddisfare

due esigenze fondamentali: primo, cercare sempre di ottenere un codice più corto possibile e, secondo, non ingannare poi il microprocessore con valori differenti da quelli reali.

In particolare vediamo innanzitutto che le prime quattro istruzioni (e questo è stato davvero un caso ...) sono state codificate con i due byte di opcode (83H e FBH) e con un solo byte di operando immediato (ricordiamo che il registro interessato è BX e perciò a 16 bit): viceversa le ultime quattro hanno un opcode diverso (81H e FBH), mentre l'operando immediato è espresso a 16 bit.

Vediamone più da vicino il significato, analizzando istruzione per istruzione.

CMP BX,1: in questo caso il valore immediato «1» può essere benissimo espresso con un solo byte (criterio della massima economia) ed in più il suo segno è positivo per cui basta settare il bit 1 dell'opcode (infatti si ottiene il valore 83H, in contrapposizione con 81H che ha il bit 1 posto a «0»): ciò istruirà il microprocessore, all'atto dell'esecuzione del programma, di estendere con il segno l'operando (posto per economia su di un byte) su due byte, ottenendo così il valore (interno ...) di 0001H.

CMP BX,−1: ora il valore espresso in un solo byte (FFH) ha il bit più significativo posto ad 1 indicante che si tratta di un valore negativo, ma, mentre si sarebbe dovuto codificare il valore immediato con la word FFFFH dal momento che BX è a 16 bit, il furbo assemblatore ha lasciato un byte per la codifica del valore −1, demandando poi all'opcode il compito di istruire il microprocessore ad effettuare lui l'estensione del segno su 16 bit.

CMP BX,33H: è analogo al primo caso.

CMP BX,−33H: anche questo è analogo al secondo in quanto il valore è negativo. I valori 33 e −33 li abbiamo messi solo per riprova del comportamento con valori «minori di 127 in valore assoluto».

CMP BX,129: eccolo qui! Con un valore «positivo» (come lo è in effetti 129!) si ottiene un byte (81H) avente il bit più significativo settato e che quindi in apparenza poteva essere preso come byte negativo: ma l'accorto assemblatore NON ha certo lasciato l'81H come byte e 83H come opcode, come aveva fatto nei casi precedenti, perché in questo caso avrebbe fuorviato in seguito il microprocessore costringendolo ad estendere il segno di 81H ad una word (ottenendo un valore FF81H). Invece ha correttamente codificato il valore 129 con una word ed in più, con l'opcode 81H istruirà il microprocessore di effettuare la comparazione tra il contenuto di BX e la word 0081H.

CMP BX,0FFH: questo caso è equivalente al precedente in quanto il valore da noi impostato, anche se in esadecimale, è maggiore di 127, tantoché vale 255.

CMP BX,−0FFH: subdoli, no?! In questo caso il valore impostato (pari a −255) è già di per sé negativo e perciò l'assemblatore l'ha comodamente tradotto con la word FF01H, mentre per l'opcode valgono le considerazioni precedenti.

CMP BX,80H: anche questo caso è analogo ai precedenti e serviva solo per saggiare la bontà dell'assemblatore (sempre che ce ne fosse ancora bisogno!), che ricordiamo essere il MASM,EXE (della Microsoft).

Alla luce di questi fatti riusciranno i lettori a tradurre mentalmente in codice macchina le seguenti istruzioni?

```
CMP BX,−0
CMP BX,32768
```

È semplice! A dispetto del «−0» l'assemblatore lo interpreta ovviamente come «0» e genera il codice 83 FB 00, dove ancora una volta l'economia ha dettato legge e l'83H dice di estendere il segno (dello 0 !? boh!), mentre per il valore 32768, NON genera un errore dovuto al fatto che 32767 è il massimo valore esprimibile con una word lavorando in complemento a 2, ma bensì genera un risultato espresso dal valore 8000H, confermato dalla codifica 81 FB 00 80.

L'istruzione OR

Riguardo l'istruzione OR non c'è molto da dire, se non che, come è lecito aspettarsi, effettua l'OR logico tra i due operandi (al solito ad 8 o a 16 bit), bit a bit, e ponendo il risultato nel primo operando.

Anche in questo caso vengono alterati i flag principali: in particolare il Carry Flag (CF) e l'Overflow Flag (OF) vengono entrambi resettati, mentre viceversa i Flag Sign (SF), Parity (PF) e Zero (ZF) vengono alterati in funzione del risultato dell'operazione logica.

Largamente sfruttato è il flag di Zero che in questo caso viene settato se e solo se i due operandi dell'istruzione sono entrambi nulli: se il secondo operando coincide con il primo allora il test sul flag di Zero corrisponde a testare se l'operando è nullo.

L'istruzione XOR

Analogamente all'istruzione OR, sull'istruzione XOR non c'è molto da dire se non ricordare che l'operazione di XOR («eXclusive OR», OR esclusivo) tra due operandi ad 8 o a 16 bit fa sì che un bit venga settato se e solo se i corrispondenti bit degli operandi sono differenti (uno «0» e l'altro «1»), mentre fornisce uno «0» laddove i bit corrispondenti risultino uguali.

Anche per lo XOR, i Flag di Carry (CF) e di Overflow (OF) vengono resettati ed ancora una volta i Flag di Sign (SF), Parity (PF) e Zero (ZF) vengono alterati in funzione del risultato dell'OR esclusivo.

In questo caso il flag di Zero verrà resettato solo nel caso in cui i due operandi siano l'uno il negativo dell'altro, mentre potrà essere settato in moltissimi altri casi e perciò non necessariamente quando i due operandi coincidano.

Le istruzioni di manipolazione del Flag di Carry

Sono tre istruzioni molto utili in quanto permettono di forzare il valore del Carry Flag (CF) a quello che desideriamo noi: potremo così settarlo (con l'istruzione STC, che sta appunto per «SeT Carry»), complementarlo (e cioè invertirlo, con l'istruzione CMC, che sta per «CoMplement Carry») oppure infine resettarlo (con l'istruzione CLC, che sta per «CLear Carry»).

Particolarmente utile è l'istruzione CMC che permette come detto di complementare il Flag di Carry, il tutto ovviamente senza aver bisogno di sapere in quale stato si trovi all'inizio il Carry stesso.

Sono istruzioni che vengono eseguite in pochissimo tempo e naturalmente non alterano in alcun modo i rimanenti flag (ci mancherebbe ...)

Con questo abbiamo terminato l'analisi delle istruzioni logiche del microprocessore 8086/88: la prossima puntata affronteremo l'analisi delle istruzioni di gestione delle stringhe, che come vedremo non necessariamente saranno le stringhe così come le conosciamo e le usiamo nei linguaggi ad altro livello e cioè delle sequenze di caratteri ASCII.

Vedremo che con tale termine si intendono generici blocchi di dati, siano essi byte o word, sui quali effettuare un certo numero di operazioni primitive senz'altro interessanti.

NUOVA NEWEL sas

Attualità elettroniche e Microcomputers

20155 MILANO - Via Mac Mahon, 75
Tel.: neg. 02/32.34.92 - uff. 32.70.226

ORARI 9.00 - 12.30
15.00 - 19.00
CHIUSO IL LUNEDì

RICHIEDERE I CATALOGHI, SPECIFICANDO IL SETTORE.

VENDITA ANCHE PER CORRISPONDENZA IN CONTRASSEGNO IN TUTTA ITALIA SI SERVONO RIVENDITORI

AMSTRAD PC

512K con 2 drive 360K, mouse, nuovo processore 8086 e monitor

L. 1.590.000 DOS MICROSOFT e GEM in omaggio

PC COMPATIBILE 256K 418 MH2
2 DRIVE 360K - COMPLETO DI MONITOR

PC ISM 8088/XT
PER TE CHE ESIGI IL MASSIMO

LA SOLUZIONE IDEALE AI TUOI PROBLEMI PIÙ COMPLESSI DI AFFIDABILITÀ E COMPATIBILITÀ, LE SUE QUALITÀ SONO IMPAGABILI RICAMBI DISPONIBILI A STOCK. RIPARAZIONI ENTRO 7 GIORNI.

LETTO TUTTO?
BENE ORA TI SVELIAMO UN PICCOLO SEGRETO: HA UN DIFETTO!!
COSTA TROPPO POCO!!
E RAMMENTA, PER TE CHE ESIGI IL MASSIMO CI SARÀ SEMPRE LA MASSIMA ASSISTENZA.

A L. 1.355.000 + IVA

DISPONIBILI ANCHE VERSIONI AT E PORTATILI.

PC 256/640K

CASE AT, 1 disk drive 360 K, tasteria 84 tasti, 1 Hard Disk 20 MB NEC (o simili), monitor 12'' fosfori verdi/ambra, stampante 80 colonne 80/120 cps (in omaggio 20 dischetti BULK)

il tutto al favoloso prezzo di
L. 2.119.000 + IVA

Atari

520 STM + DRIVE A SOLE L. 900.000 IVA COMPRESA
ATARI 1040 A L. IVA COMPRESA
ATARI
SONO DISPONIBILI TUTTI GLI ULTIMI ARRIVI DI SOFTWARE A PREZZI ECCEZIONALI

MODEM

300/200 per int seriale o 64 a	L. 229.000
300 baud a	L. 118.000
Smartmodem AYES da	L. 398.000

Hard disk per IBM

20MB	L. 999.000
10MB	L. 890.000

PRESSO IL NEGOZIO PUOI TROVARE ACCESSORI DI OGNI TIPO PER PC MS/DOS A PREZZI BASSISSIMI, COME MOUSE A L. 150.000 - ESPANSIONI 640K A L. 150.000 COOPROCESSORI MATEMATICI ETC, ETC...

IMPORT

SPECIALE QL SINCLAIR - SPECTRUM

128K - INGLESE L. 299.000 + IVA
128K - ITALIANO L. 289.000 + IVA
2 x SPECTRUM PLUS L. 249.000 - IVATO
FLOPPY DISK 720K PER QL O SPECTRUM L. 449.000 - IVATO
ESPANSIONE 640K QL L. 199.000
ESPANSIONE 48K SPECTRUM L. 29.000

SONO DISPONIBILI TUTTI I TIPI DI CARTUCCE PER COMMODORE A PREZZI DI STOCK - COME OMA FINAL CARTRIDGE, FREEZE FRANE, DUPLICATORI CASSETTE, SPEEDOS, FAST DISK, TANAREGISTRATORI, PROGRAMMATORI.
NUOVISSIMA DRIVE SLIM A 5''14 PER COMMODORE 64/128/C16
PLUS4 - DOPPIA VELOCITÀ DEL 1570 - 1541 A SOLE L. 350.000
A STOCK COMMODORE, 64 - 128/D - 1541 - 1571 - MPS 1000 - MPS 1200 - 802 - 803 - OKIMATE 20 - MANNESMANN TALLY - CENTRONICS - PREZZI DA GROSSISTA - TUTTI I TIPI DI MODEM
HARD DISK PER IBM NEC - ALTRE MARCHE - ESPANSIONI - SCHEDE MULTIFUNZIONI - SCHEDE EGA - HERCULES - MOUSE COR GRAPHIC - PER TUTTI I PC/COMPATIBILI
E INOLTRE TUTTE LE ULTIME NOVITÀ SOFTWARE PER I PIÙ DIFFUSI COMPUTER.

ESPANSIONI A SCHEDE IBM

640K IBM	L. 150.000
SERIALE	L. 90.000
PARALLELA	L. 90.000
HERCULES	L. 190.000
COLOR	L. 190.000
I/O PLUS	L. 199.000
Programmatore EPROM SILK	L. 439.000

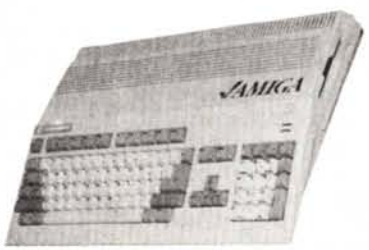
QL a L. 299.000

Spectrum Plus a	L. 229.000
Discipline Disk Interface	L. 239.000
Cartucce microdrive	L. 4.500
Programmatori EPROM 512K per QL	L. 249.000

AMIGA

AMIGA 1000 DRIVE 512K - MONITOR PAL HR - MOUSE - TASTIERA ITALIANA - GARANZIA 12 MESI COMMODORE ITALIA L. 1.900.000 + IVA
AMIGA SIDECAR (MS/DOS) L. 1.300.000 + IVA

DIGIWELL (ORIGINALE AMERICANO) L. 449.000 IVATO SCONTI PER QUANTITA CHIEDERE
DRIVE AGGIUNTO 3''1/2 SLIM L. 440.000 IVATO SCONTI PER QUANTITA CHIEDERE
MODEM COMPLETO CAVO E SOFTWARE DA L. 270.000 IN SU CHIEDERE
SONO DISPONIBILI OLTRE 500 TITOLI DI SOFTWARE - ARRIVI SETTIMANALI DAGLI USA - RICHIEDERE CATALOGO - SI FANNO SCONTI AI RIVENDITORI
IN ARRIVO AMIGA 500 - AMIGA 2500 - ESPANSIONE MEMORIA - HARD DISCK - SINTETIZZATORI



RIVENDITORE AUTORIZZATO

Amiga 500 e 2000