L'Assembler x86

Istruzioni aritmetiche

M. Rebaudengo - M. Sonza Reorda

Politecnico di Torino
Dip. di Automatica e Informatica

Istruzioni aritmetiche

Si suddividono in:

- **istruzioni per il calcolo binario**
- **istruzioni per il calcolo tra numeri BCD (*non trattate*).**

Formato dei dati

L’8086 può eseguire operazioni aritmetiche su numeri nei seguenti formati:

- **numeri binari senza segno, su 8 o 16 bit**
- **numeri binari con segno, su 8 o 16 bit**
- **numeri decimali *packed*: ogni byte contiene due numeri decimali codificati in BCD; la cifra più significativa sta nei 4 bit superiori**
- **numeri decimali *unpacked*: ogni byte contiene un solo numero decimale BCD nei 4 bit inferiori; i 4 bit superiori devono essere a zero se il numero è usato in un’operazione di moltiplicazione o divisione.**

Le istruzioni ADD e SUB

Formato:

```
ADD  dest, sorg
SUB  dest, sorg
```

Funzionamento:

L'istruzione ADD esegue un'addizione tra l'operando *dest* e l'operando *sorg* e scrive il risultato nell'operando *dest*; l'operando *sorg* rimane immutato.

L'istruzione SUB esegue una sottrazione tra l'operando *dest* e l'operando *sorg* e scrive il risultato nell'operando *dest*; l'operando *sorg* rimane immutato.

Le istruzioni ADD e SUB modificano il valore di tutti i flag (AF, PF, CF, SF, OF, ZF**).**

Restrizioni sulle istruzioni ADD e SUB

- **Gli operandi devono essere dello stesso tipo (o entrambi byte o entrambi word).**
- **L'operando destinazione può essere un registro, oppure una locazione di memoria.**
- **L'operando sorgente può essere un registro, una locazione di memoria, oppure un valore immediato.**
- **Non è lecito eseguire l'istruzione tra due locazioni di memoria.**

```
ADD   VAL1, VAL2        ; ERRORE !!!
```

Si può sostituire con:

```
MOV   AH, VAL2
ADD   VAL1, AH
```

L'istruzione CBW

Formato:

CBW

Funzionamento:

L'istruzione CBW permette di convertire un byte nella word equivalente.

L'istruzione CBW esegue l'estensione del segno del contenuto del registro AL a tutto il registro AH:

- **se AL contiene un numero positivo, AH è caricato con il valore 00H;**
- **se AL contiene un numero negativo, AH è caricato con il valore FFH.**

L'istruzione CBW (II)

L'istruzione CBW risulta utile quando si vuole eseguire un'operazione di addizione o sottrazione tra un numero con segno memorizzato in un byte e un numero con segno memorizzato in una word.

Esempio

```
MOV   AL, VALORE
CBW
ADD   SI, AX
```

L'istruzione ADC

Formato:

ADC dest, sorg

Funzionamento:

L'istruzione ADC somma al contenuto dell'operando *dest* il contenuto dell'operando *sorg* ed il valore del flag CF.

L'istruzione ADC ha il seguente comportamento:

- **se CF vale 0 l'istruzione ADC si comporta come un'istruzione ADD;**
- **se CF vale 1 l'istruzione ADC aggiunge 1 al risultato ottenuto con un'istruzione ADD.**

Somma tra numeri interi su 32 bit

Per eseguire le operazioni aritmetiche di somma tra numeri di tipo doubleword occorre sommare coppie di word, cominciando da quella meno significativa.

Le operazioni da eseguire sono:

- **si sommano le due word meno significative utilizzando l'istruzione `ADD`**
- **si sommano le due word più significative utilizzando l'istruzione `ADC`.**

Somma di 2 numeri su 32 bit

```
        .STACK
        .DATA
NUMA    DD      ?
NUMB    DD      ?
NUMC    DD      ?
        ...
        .CODE
        ...
        MOV     AX, WORD PTR NUMA       ; somma tra le 2 word
        ADD     AX, WORD PTR NUMB       ; meno significative
        MOV     WORD PTR NUMC, AX
        MOV     AX, WORD PTR NUMA+2     ; somma tra le due word
        ADC     AX, WORD PTR NUMB+2     ; più significative + CF
        MOV     WORD PTR NUMC+2, AX
        ...
```

Somma di 2 numeri su 64 bit

```
            .DATA
NUMA        DQ          ?
NUMB        DQ          ?
NUMC        DQ          ?
            ...
            .CODE
            ...
            CLC         ; azzeramento del flag CF
            LEA         SI, WORD PTR NUMA
            LEA         DI, WORD PTR NUMB
            LEA         BX, WORD PTR NUMC
```

```
            MOV  CX, 4
ciclo:      MOV  AX, [SI]
            ADC  AX, [DI] ; [DI] + [SI] + CF
            MOV  [BX], AX
            INC  SI
            INC  SI
            INC  DI
            INC  DI
            INC  BX
            INC  BX
            DEC  CX
            JNE  ciclo
            ...
```

L'istruzione SBB

Formato:

SBB dest, sorg

Funzionamento:

L'istruzione SBB esegue la sottrazione tra l'operando *dest* e l'operando *sorg;* il valore del flag CF viene sottratto al risultato ed il valore ottenuto viene copiato nell'operando *dest*; l'operando *sorg* rimane immutato.

L'istruzione SBB ha il seguente comportamento:

- **se CF vale 0 l'istruzione SBB si comporta come un'istruzione SUB**
- **se CF vale 1 l'istruzione SBB sottrae 1 al risultato ottenuto con un'istruzione SUB.**

Differenza tra numeri su 64 bit

```
        .MODEL     small
        .STACK
        .DATA
NUMA    DQ         ?     ; definisce una var su 64 bit
NUMB    DQ         ?
NUMC    DQ         ?
        ...
        .CODE
        ...
        CLC               ; azzeramento del flag CF
        LEA   SI, WORD PTR NUMA
        LEA   DI, WORD PTR NUMB
        LEA   BX, WORD PTR NUMC
```

```
            MOV  CX, 4; 4 iterazioni
ciclo:      MOV  AX, [SI]
            SBB  AX, [DI]  ; [SI] - [DI]- CF
            MOV  [BX], AX
            INC  SI
            INC  SI
            INC  DI
            INC  DI
            INC  BX
            INC  BX
            DEC  CX
            JNE  ciclo
            ...
```

Le istruzioni INC e DEC

Formato:

```
INC   operando
DEC   operando
```

Funzionamento:

L'istruzione *INC* incrementa *operando* di un'unità e copia il risultato in *operando* stesso.

L'istruzione *DEC* decrementa *operando* di un'unità e copia il risultato in *operando* stesso.

Le due istruzioni aggiornano tutti i flag di stato tranne il flag `CF`.

L'istruzione NEG

Formato:

NEG operando

Funzionamento:

L'istruzione *NEG* cambia il segno di *operando*, che si assume rappresentato in complemento a 2.

L'operando puè essere un registro oppure il contenuto di una locazione di memoria.

L'istruzione *NEG* aggiorna tutti i flag di stato.

Calcolo del modulo di un vettore

Si vuole realizzare un programma che calcola il modulo del contenuto di tutte le celle di un vettore di interi.

```
main()
{
 int i, vett[10];
 ...
 for (i=0 ; i < 10 ; i++)
      if (vett[i] < 0)
          vett[i] *= -1;
 ...
}
```

Soluzione Assembler

```
LUNG        EQU         10
            .MODEL      small
            .STACK
            .DATA
      VETT  DW          LUNG DUP (?)
            ...
            .CODE
            ...
      MOV         SI, 0
      MOV         CX, LUNG
```

```
ciclo:    CMP  VETT[SI], 0     ; elemento < 0 ?
          JNL  continua  ; No: va a continua
          NEG  VETT[SI]  ; Sì: calcola il modulo
continua: ADD  SI, 2     ; scansione del vettore
          DEC  CX
          CMP  CX, 0
          JNE  ciclo
          ...
```

Carry e Overflow

Quando la ALU esegue un'istruzione aritmetica, aggiorna i flag di Carry e Overflow (CF e OF).

Si ricorda che la ALU

- **non sa se i due operandi e il risultato sono rappresentati con segno o senza**
- **aggiorna CF e OF sulla base di semplici regole**
  - **CF: nel caso della somma è forzato a 1 se il bit di carry generato dai due MSB è 1**
  - **OF: viene messo a 1 se la somma di due numeri con lo stesso segno produce un numero con segno diverso.**

Il programmatore deve

- **guardare il CF se lavora su operandi senza segno**
- **guardare l'OF se lavora su operandi con segno.**

Le istruzioni MUL e IMUL

Formato

```
MUL     operando
IMUL    operando
```

Uso

Permettono di eseguire l'operazione di moltiplicazione tra numeri interi senza segno (`MUL`) e con segno (`IMUL`).

Funzionamento

L'*operando* può essere un registro oppure una locazione di memoria; il suo tipo può essere BYTE oppure WORD.

Non è ammessa la moltiplicazione per un valore immediato.

Le istruzioni MUL e IMUL

(segue)

Il processore salva il risultato della moltiplicazione in un operando di lunghezza doppia rispetto ai fattori.

I due casi possibili sono:

- **se si specifica un operando di tipo BYTE, il processore**
  - **esegue la moltiplicazione tra l'operando e il contenuto del registro AL**
  - **copia il risultato nel registro AX**
- **se si specifica un operando di tipo WORD, il processore**
  - **esegue la moltiplicazione tra l'operando e il contenuto del registro AX**
  - **copia il risultato nei registri DX (word più significativa) ed AX (word meno significativa).**

Le istruzioni `MUL` e `IMUL`

(segue)

Le istruzioni di moltiplicazione aggiornano i flag `CF` ed `OF` in modo da segnalare se la parte più significativa del risultato è nulla:

- **in una moltiplicazione tra byte, i flag `CF` ed `OF` valgono `0` se il registro `AH` è nullo;**
- **in una moltiplicazione tra word, i flag `CF` ed `OF` valgono `0` se il registro `DX` è nullo.**

Calcolo del quadrato

```
        .MODEL      small
        .STACK
        .DATA
NUM     DW          ?
RES     DD          ?
        ...
        .CODE
        ...
        MOV     WORD PTR RES+2, 0
        MOV     AX, NUM             ; AX = NUM
        MUL     AX                  ; DX,AX = NUM * NUM
        MOV     WORD PTR RES, AX
        JNC     esce                ; word alta = 0 ?
        MOV     WORD PTR RES+2, DX
esce:   ...
```

Moltiplicazione tra dati di tipo diverso

Le istruzioni MUL ed IMUL permettono di eseguire solo la moltiplicazione tra dati dello stesso tipo (o entrambi byte o entrambi word).

È possibile moltiplicare un byte per una word utilizzando opportunamente l'istruzione CBW (nel caso di numeri con segno).

Esempio:

```
        .DATA
BVAL    DB   ?
WVAL    DW   ?
        .CODE
        ...
        MOV  AL, BVAL
        CBW
        IMUL WVAL
```

Le istruzioni DIV e IDIV

Formato

```
DIV     operando
IDIV    operando
```

Uso

Permettono di eseguire l'operazione di divisione tra numeri interi senza segno (`DIV`) e con segno (`IDIV`).

Funzionamento:

L'*operando* può essere un registro oppure una locazione di memoria.

Non è ammessa la divisione per un valore immediato.

Le istruzioni `DIV` e `IDIV`

Le istruzioni di divisione possono eseguire due tipi di operazioni:

- **divisione tra un operando di tipo word e un operando di tipo byte**
- **divisione tra un operando di tipo doubleword e un operando di tipo word.**

Le istruzioni `DIV` e `IDIV`

Le istruzioni di divisione restituiscono due risultati

- ***quoziente***
- ***resto*.**

Il comportamento delle istruzioni di divisione è diverso a seconda del tipo di operazione:

- **se l'operando è di tipo `BYTE`, il processore**
  - **esegue la divisione tra il contenuto del registro `AX` (dividendo) e il contenuto dell'*operando* (divisore)**
  - **scrive il quoziente nel registro `AL` e il resto nel registro `AH`**
- **se l'operando è di tipo `WORD`, il processore**
  - **esegue la divisione tra il contenuto dei registri `DX,AX` (dividendo) e il contenuto dell'*operando* (divisore)**
  - **scrive il quoziente nel registro `AX` ed il resto nel registro `DX`.**

Le istruzioni `DIV` e `IDIV`

Le istruzioni di divisione non aggiornano i flag.

Nel caso in cui il divisore sia troppo piccolo, il processore rileva l'errore e salta alla *procedura di gestione dell'interruzione causata da una divisione per 0* (*interrupt di tipo 0*).

Esempio

```
MOV   AX, 1024
MOV   BL, 2
DIV   BL          ; 1024 / 2  = 512
                  ; non sta su un byte
                  ; overflow di divisione
```

Media di un insieme di numeri

Si vuole realizzare un frammento di codice che calcola il valor medio dei numeri positivi memorizzati in un vettore di numeri interi con segno.

```
main()
{int i, count=0, somma=0, avg, vett[10];
 ...
 for (i=0 ; i<10 ; i++)
     if (vett[i] > 0)
      {     count++;
            somma += vett[i];
      }
 avg = somma/count;
 ...
}
```

Soluzione Assembler

```
LUNG        EQU  10
            .MODEL    small
            .STACK
            .DATA
VETT        DW   LUNG DUP (?)
COUNT       DB   ?              ; numero di positivi
AVG         DB   ?
            ...
            .CODE
            ...
            MOV  CX, LUNG
            MOV  SI, 0
            MOV  BX, 0; somma totale
            MOV  COUNT, 0
```

```
ciclo:      CMP   VETT[SI], 0       ; VETT[] > 0 ?
            JNG   continua          ; No: va a continua
            INC   COUNT             ; Sì: incrementa il
                                    ; contatore
            ADD   BX, VETT[SI]      ; BX = BX + VETT[SI]
            JC    errore
continua:   ADD   SI, 2             ; scansione del vettore
            DEC   CX
            CMP   CX, 0
            JNZ   ciclo
            MOV   AX, BX            ; copia in AX del
                                    ; dividendo
            DIV   COUNT             ; BX / COUNT
            MOV   AVG, AL           ; copia in AVG del
                                    ; quoziente
            ...
            JMP dopo_err
errore:     ...                     ; codice per la gestione
                                    ; dell'overflow nella
                                    ; somma
dopo_err:
```

L'istruzione CWD

Formato

CWD

Funzionamento:

L'istruzione CWD permette di convertire una word nella doubleword equivalente.

L'istruzione CWD esegue l'estensione del segno del contenuto del registro AX a tutto il registro DX:

- **se AX contiene un numero positivo, DX è caricato con il valore 0000H**
- **se AX contiene un numero negativo, DX è caricato con il valore FFFFH.**

L'istruzione CWD (II)

L'istruzione CWD risulta utile ad esempio quando si vuole eseguire un'operazione di divisione tra due numeri con segno su 16 bit.

Esempio

```
MOV   AX, CX      ; carico il dividendo in AX
CWD               ; estensione a DX:AX
IDIV  BX          ; divisione
```