L'Assembler 8086

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 nelle Istruzioni Aritmetiche

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.**

3

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).

4

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
```

5 **M. Rebaudengo, M. Sonza Reorda**

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`.**

6 **M. Rebaudengo, M. Sonza Reorda**

L'istruzione CBW

(segue)

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

Le operazioni da eseguire sono:

- **caricare il dato di tipo byte in `AL`**
- **eseguire la conversione da byte a word con l'istruzione `CBW`**
- **eseguire l'operazione aritmetica desiderata tra `AX` ed il dato di tipo word.**

Esempio

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

7

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`.**

8

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`.**

9 M. Rebaudengo, M. Sonza Reorda

Somma tra numeri su 32 bit

```
      .MODEL    small
      .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
      ...
```

10 M. Rebaudengo, M. Sonza Reorda

Somma tra numeri su 64 bit

```
            .MODEL    small
            .STACK
            .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
              LOOP 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        ?
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
          LOOP      ciclo
          ...
```

15

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.

16

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 lo stato di tutti i flag di stato.

17 M. Rebaudengo, M. Sonza Reorda

Calcolo del modulo di un vettore

Specifiche:

Si realizzi un programma che calcoli 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;
 ...
}
```

18 M. Rebaudengo, M. Sonza Reorda

Soluzione Assembler

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

19 M. Rebaudengo, M. Sonza Reorda

```
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
          LOOP ciclo
          ...
```

20 M. Rebaudengo, M. Sonza Reorda

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)

I fattori della moltiplicazione devono essere dello stesso tipo (o entrambi byte o entrambi word).

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 ed il contenuto del registro AL e copia il risultato nel registro AX.**
- **se si specifica un operando di tipo WORD, il processore esegue la moltiplicazione tra l'operando ed il contenuto del registro AX e 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 per una costante

Istruzioni introdotte a partire dall'80186.

Formato

```
IMUL    op, cost
IMUL    dest, op, cost
```

Uso

Permettono di eseguire l'operazione di moltiplicazione tra un numero intero ed un valore immediato.

Funzionamento:

Gli operandi *op* e *dest* possono essere uno dei registri general purpose. Il processore moltiplica l'operando *op* per la costante; se è specificato il campo *dest* l'istruzione copia il risultato nel registro *dest*, altrimenti lo copia nel registro *op*.

25

Moltiplicazione per una costante

(segue)

Il risultato del prodotto è memorizzato comunque su 16 bit.

Occorre fare attenzione alla correttezza del risultato; se il risultato dell'operazione è rappresentabile su una word, l'istruzione azzera i flag `CF` ed `OF`; in caso contrario, l'istruzione pone a 1 i flag `CF` ed `OF` (*overflow* nella moltiplicazione).

26

Conversione di valuta

Specifiche:

Si realizzi un frammento di programma che converte il costo di un prodotto da franchi francesi in lire italiane.

```
#define FFRANCO 295
main()
{
int f_costo, it_costo;
...
   it_costo = f_costo*FFRANCO;
...
}
```


Soluzione Assembler

```
FFRANCO     EQU   295
            .386
            .MODEL      small
            .STACK
            .DATA
F_COST      DW    ?
IT_COST     DW    ?
ERR_MSG     DB    "Overflow nella moltiplicazione",0DH,0AH,"$"
            .CODE
            ...
            MOV   AX, F_COST
            IMUL  AX, FFRANCO       ; AX = AX * 297
            JNC   ok                ; CF = 1 ?
            LEA   DX, ERR_MSG       ; Si: messaggio di errore
            MOV   AH, 09H
            INT   21H
            JMP   esci
ok:         MOV   IT_COST, AX       ; No
esci:       ...
```

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 od entrambi word).

È possibile moltiplicare un byte per una word utilizzando opportunamente l'istruzione CBW.

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`

(segue)

Le istruzioni di divisione possono eseguire due tipi di operazioni:

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

Le istruzioni `DIV` e `IDIV`

(segue)

Le istruzioni di divisione restituiscono due risultati: il *quoziente* ed il *resto* della divisione.

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) ed il contenuto dell'*operando* (divisore); come risultato dell'operazione l'istruzione restituisce il quoziente nel registro `AL` ed 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) ed il contenuto dell'*operando* (divisore); come risultato dell'operazione l'istruzione restituisce il quoziente nel registro `AX` ed il resto nel registro `DX`.**

Le istruzioni `DIV` e `IDIV`

(segue)

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

Specifiche

Si realizzi un frammento di codice che calcoli 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
```

35 M. Rebaudengo, M. Sonza Reorda

```
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]
continua: ADD  SI, 2          ; scansione del vettore
          LOOP ciclo
          MOV  AX, BX         ; copia in AX del
                              ; dividendo
          IDIV COUNT          ; BX / COUNT
          MOV  AVG, AL        ; copia in AVG del
                              ; quoziente
          ...
```

36 M. Rebaudengo, M. Sonza Reorda

L'istruzione CWD

Formato

CWD

Funzionamento:

L'istruzione CWD permette di convertire un 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.**

37

L'istruzione CWD

(segue)

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

Le operazioni da eseguire sono:

- **caricare il dividendo in AX;**
- **eseguire la conversione in doubleword con l'istruzione CWD;**
- **eseguire l'operazione di divisione.**

Esempio

```
MOV  AX, CX
CWD
IDIV BX
```

38