L'Assembler x86

Istruzioni per il trasferimento dei dati

M. Rebaudengo - M. Sonza Reorda

Politecnico di Torino
Dip. di Automatica e Informatica

Istruzioni per il trasferimento dei dati

- **MOV**
- **XCHG**
- **LEA**
- **XLAT**
- **PUSH e POP**
- **PUSHA e POPA**
- **PUSHF e POPF**
- **IN e OUT**

Istruzione MOV

L'istruzione `MOV` copia un dato da una posizione ad un'altra.

Il suo formato in codice sorgente è il seguente:

`MOV destinazione, sorgente`

I dati vengono letti dall'operando `sorgente` e memorizzati nell'operando `destinazione`.
Il dato nell'operando sorgente non viene modificato.

L'operando sorgente può essere

- **un registro, oppure**
- **una locazione di memoria, oppure**
- **un valore immediato**

L'operando destinazione può essere

- **un registro, oppure**
- **una locazione di memoria.**

Combinazioni non ammesse da MOV

Valgono le seguenti regole:

- **il tipo dell'operando sorgente deve essere lo stesso dell'operando destinazione**

```
MOV BL, DX ; ERRORE !!
```

- **il registro IP non può essere né sorgente né destinazione**
- **il registro CS non può essere destinazione.**

Combinazioni non ammesse da MOV (II)

- **memoria ⇒ memoria**
  Si deve passare attraverso un registro general-purpose:

```
MOV AX, PIPPO
MOV PLUTO, AX
```

- **segment register ⇒ segment register**

Si deve passare attraverso un registro general-purpose:

```
MOV AX, ES
MOV DS, AX
```

Oppure si usa lo stack:

```
PUSH ES
POP  DS
```

Combinazioni non ammesse da MOV (III)

- **segment register ⇐ immediato**

  Si deve passare attraverso un registro general-purpose

```
MOV AX, DATA_SEG
MOV DS, AX
```

Istruzione XCHG

L'istruzione `XCHG` permette di eseguire lo scambio tra due registri o tra un registro ed una locazione di memoria. Il suo formato è il seguente:

`XCHG operando1, operando2`

Dopo l'esecuzione di questa istruzione il contenuto di `operando1` è pari al precedente valore di `operando2` e viceversa.

Esempi

```
XCHG   AX, BX
XCHG   MEMORY, AX
```

Restrizioni nell'uso di XCHG

- **Gli operandi devono avere la stessa lunghezza**
- **Nessuno dei due operandi può essere un registro di segmento**
- **Non è possibile scambiare il contenuto di due locazioni di memoria**

  Per eseguire quest'operazione si può utilizzare un registro temporaneo:

```
MOV     AH, DATO2
XCHG    AH, DATO1
MOV     DATO2, AH
```

Esempio

Si desidera invertire l'ordine degli elementi di un vettore.

```
#define LUNG 150
main()
{
int i;
char vett[LUNG], temp;
...
for (i=0 ; i < (LUNG/2) ; i++)
      {
      temp = vett[LUNG-1-i];
      vett[LUNG-1-i] = vett[i];
      vett[i] = temp;
      }
...
}
```

Soluzione Assembler

```
LUNG        EQU   150
            .STACK
            .DATA
VETT        DB    LUNG DUP (?)
            .CODE
            ...
            MOV   SI, 0         ; SI punta al primo elemento
            MOV   DI, LUNG-1    ; DI punta all'ultimo elemento
            MOV   CX, LUNG/2
ciclo:      MOV   AH, VETT[SI]; scambio del contenuto
            XCHG  AH, VETT[DI]
            MOV   VETT[SI], AH
            INC   SI            ; aggiornamento degli indici
            DEC   DI
            DEC   CX
            CMP   CX, 0
            JNE   ciclo
            ...
```

Istruzione LEA

Formato:

```
LEA dest, sorg
```

Funzionamento:

L'offset dell'operando *sorg* viene copiato nell'operando *dest*.

Applicazione

L'istruzione `LEA` trasferisce l'effective address dell'operando sorgente nell'operando destinazione.

Esempio

```
LEA  AX, VAR
```

Copia nel registro `AX` l'offset della variabile `VAR`.

Esempio

Si desidera copiare un vettore di interi in un altro.

```
#define LUNG 500
main()
{
int i;
int sorg[LUNG], dest[LUNG];
...
for (i=0 ; i < LUNG ; i++)
      dest[i] = sorg[i];
...
}
```

Soluzione Assembler

```
LUNG        EQU   500
            ...
            .STACK
            .DATA
SORG        DW    LUNG DUP(?)
DEST        DW    LUNG DUP(?)
            .CODE
            ...
            LEA   SI, SORG
            LEA   DI, DEST
            MOV   CX, LUNG
ciclo:      MOV   AX, [SI]
            MOV   [DI], AX
            ADD   SI, 2
            ADD   DI, 2
            DEC   CX
            CMP   CX, 0
            JNZ   ciclo
            ...
```

Istruzione XLAT

Formato:

XLAT

Funzionamento:

Durante l'esecuzione il processore esegue la somma del contenuto dei registri AL e BX, trasferendo in AL il dato avente come offset il risultato di tale somma.

Applicazione

L'istruzione XLAT si usa speso quando si deve fare accesso a tabelle di conversione (*look-up table*).

BX deve contenere l'indirizzo di partenza della tabella e AL l'offset al suo interno. Al termine dell'esecuzione dell'istruzione, AL contiene il byte puntato nella tabella.

Esempio

Sia `TAB` una sequenza di byte contenente i valori esadecimali da `30H` a `39H` e da `41H` a `46H` corrispondenti al codice ASCII delle 16 cifre della rappresentazione esadecimale:

```
        .DATA
TAB     DB    30H,31H,32H,33H,34H ;01234
        DB    35H,36H,37H,38H,39H ;56789
        DB    41H,42H,43H,44H,45H,46H ;ABCDEF
        .CODE
        MOV   AL, 10
        MOV   BX, OFFSET TAB
        XLAT
```

L'istruzione copia il valore della locazione di memoria avente offset `TAB+10` nel registro `AL`.

Limiti di `XLAT`

- **I dati memorizzati nella tabella di conversione devono essere di tipo byte (per poter essere correttamente copiati in `AL`)**
- **il massimo numero di elementi in tabella è pari a 256.**

Esempio

Si realizzi un programma che esegua la conversione in codifica Gray di 100 numeri binari compresi tra 0 e 15.

Si ricorda che la codifica Gray è un particolare metodo di rappresentazione dei numeri interi, la cui caratteristica è quella di garantire che le codifiche di numeri decimali che differiscono di un'unità differiscono di un solo bit.

Soluzione Assembler

```
LUNG        EQU         100
            .MODEL      small
            .STACK
            .DATA
            ; tabella di conversione da
            ; numero decimale a codice Gray
TAB   DB    00000000B, 00000001B, 00000011B, 00000010B
            00000110B, 00000111B, 00000101B, 00000100B
            00001100B, 00001101B, 00001111B, 00001110B
            00001010B, 00001011B, 00001001B, 00001000B
NUM   DB    LUNG DUP (?)
GRAY  DB    LUNG DUP (?)
            ...
```

```
      .CODE
            ...
      LEA   SI, NUM     ; copia dell'offset di NUM
      LEA   DI, GRAY    ; copia dell'offset di GRAY
      MOV   CX, LUNG
      LEA   BX, TAB     ; copia dell'offset di TAB
lab:  MOV   AL, [SI]    ; copia di NUM in AL
      XLAT              ; conversione
      MOV   [DI], AL    ; copia di AL in GRAY
      INC   SI          ; scansione di NUM
      INC   DI          ; scansione di GRAY
      DEC   CX
      CMP   CX, 0
      JNE   lab
            ...
```

Lo stack

L'8086/8088 prevede alcune strutture e meccanismi hardware per la gestione di uno *stack*.

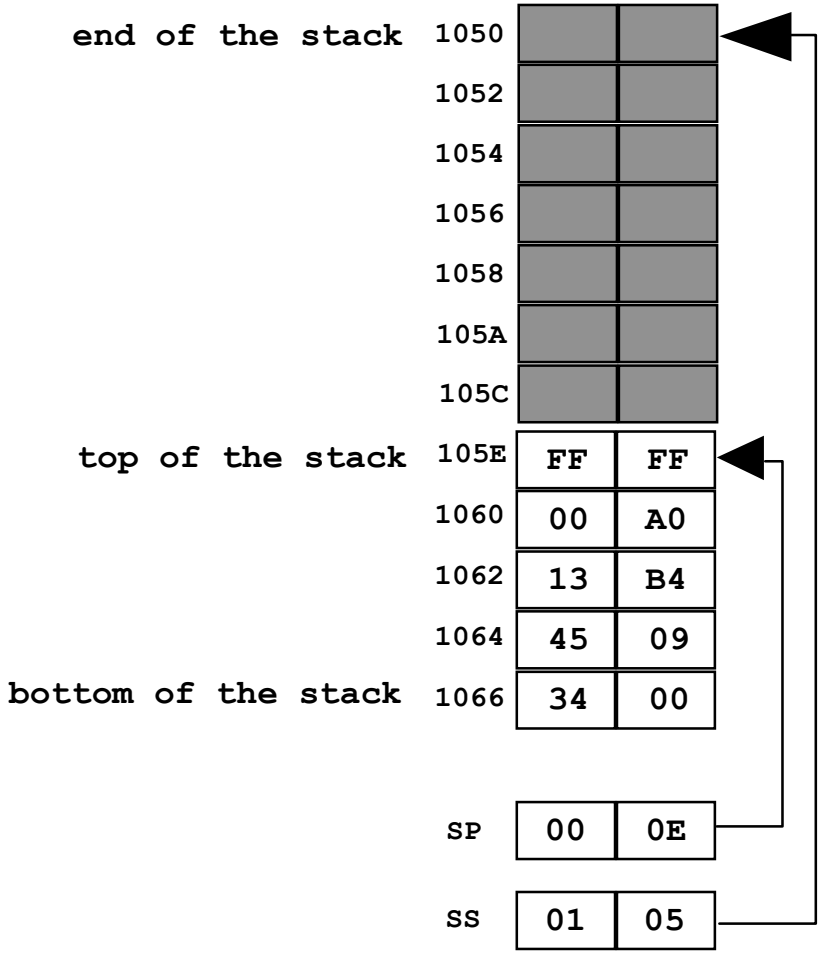
Lo stack corrisponde al segmento di memoria la cui testa è puntata da SS. Il *top* dello stack (locazione riempita per ultima) è puntato da SP.

Lo stack cresce dalle locazioni di memoria con indirizzo maggiore verso quelle ad indirizzo minore.

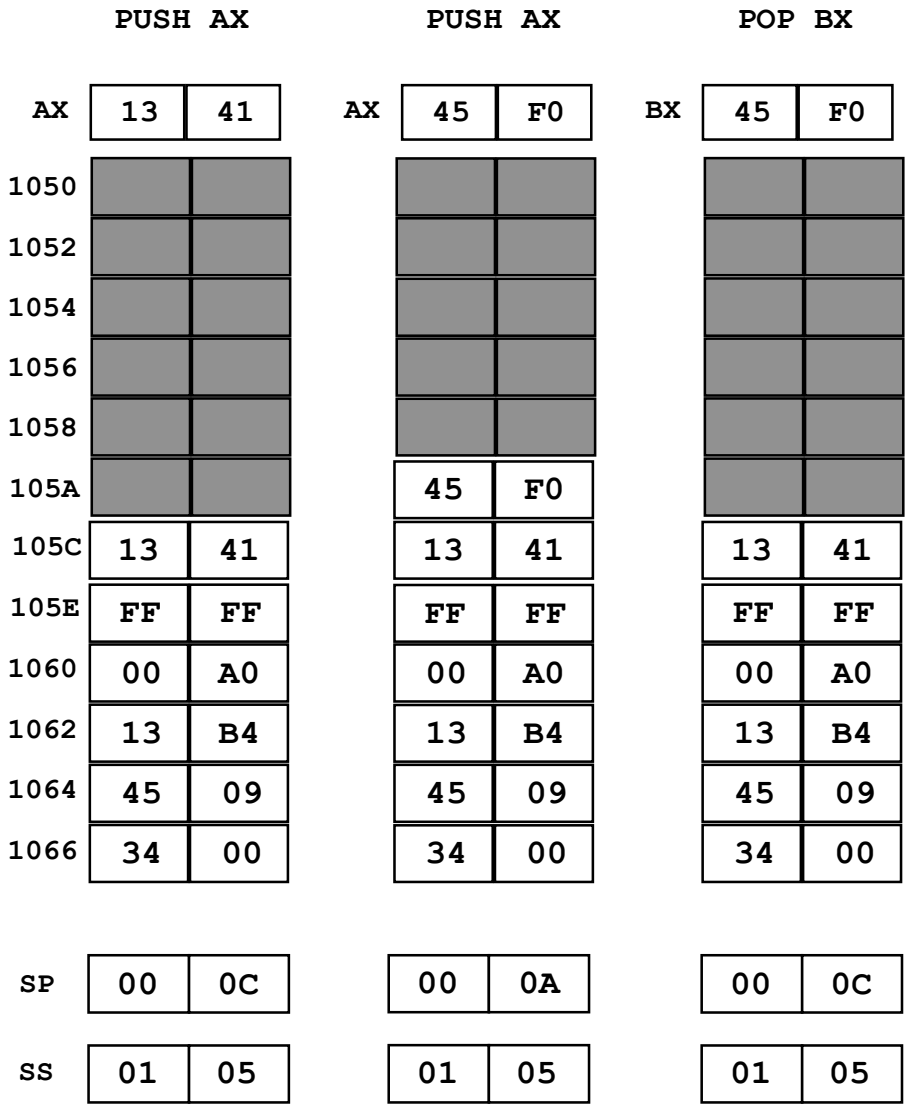
Ogni operazione di PUSH decrementa di 2 unità SP e scrive una parola nella locazione da questo puntata.

Ogni operazione di POP estrae una parola dalla locazione puntata da SP, e successivamente incrementa SP di 2 unità.

Esempio di stack



Operazioni sullo stack



Istruzioni PUSH e POP

Formato:

```
PUSH sorgente
POP destinazione
```

Funzionamento:

L'istruzione PUSH decrementa il valore di SP di 2 unità e trasferisce una word dall'operando sorgente all'elemento dello stack indirizzato da SP.

L'istruzione POP trasferisce una word dall'elemento dello stack indirizzato da SP all'operando destinazione e incrementa il registro SP di 2 unità.

Istruzioni PUSH e POP

(segue)

Le istruzioni `PUSH` e `POP` lavorano su operandi a 16 bit e possono essere utilizzate per copiare nello stack il contenuto di registri general purpose, registri di segmento e locazioni di memoria.

L'8086 non permette di eseguire la `PUSH` di un operando immediato; tale operazione è stata introdotta nel linguaggio a partire dal 80186.

Esempi

```
POP     VAL
PUSH    AX
PUSH    7        (valida dal 80186)
```

Istruzioni PUSHA e POPA

Sono state introdotte a partire dal 80186

Formato:

PUSHA

POPA

Funzionamento:

Le istruzione PUSH e POPA eseguono le operazioni di push e pop di tutti i registri *general purpose* (AX, BX, CX, DX, SP, BP, SI, DI), a partire dall'effective address memorizzato nel registro SP.

Istruzioni PUSHF e POPF

Formato:

PUSHF

POPF

Funzionamento:

Permettono di salvare e di ripristinare i 16 bit della parola di stato (PSW).

L'istruzione PUSHF decrementa il valore di SP di 2 unità e trasferisce la PSW nell'elemento dello stack indirizzato da SP.

L'istruzione POPF trasferisce la parola indirizzata da SP nella PSW e incrementa il registro SP di 2 unità.

Istruzioni IN e OUT

Formato:

```
IN   registro, porta
OUT  porta, registro
```

Funzionamento:

Attraverso le istruzioni IN e OUT, il processore scambia i dati con le periferiche di I/O.

Per leggere da un dispositivo si usa l'istruzione IN; per scrivere su un dispositivo si usa l'istruzione OUT.

Il campo *registro* può essere o AX o AL; il campo *porta* è una costante su 8 bit, oppure il registro DX, e rappresenta l'indirizzo della periferica cui si vuole accedere.