L'Assembler 8086

Istruzioni per il controllo del flusso

M. Rebaudengo - M. Sonza Reorda

Politecnico di Torino
Dip. di Automatica e Informatica

Istruzioni di salto

Le istruzioni vengono normalmente eseguite sequenzialmente nell'ordine in cui appaiono nel programma.

Utilizzando invece le istruzioni di salto (*jump*) è possibile forzare il processore a continuare l'esecuzione senza eseguire l'istruzione successiva, ma saltando ad una diversa istruzione.

Tipi di salto

I salti possono essere di tipo `NEAR` o di tipo `FAR` a seconda che l'istruzione a cui si salta appartenga o no allo stesso segmento di codice.

- **Per salti di tipo `NEAR`, il processore modifica il contenuto dell'`IP` con l'offset dell'istruzione a cui si salta.**
- **Per salti di tipo `FAR`, il processore modifica sia il contenuto dell'`IP`, sia il contenuto del registro di segmento `CS`, scrivendo in essi i valori corrispondenti all'indirizzo (offset e segmento) dell'istruzione a cui saltare.**

Istruzioni di salto

Le istruzioni di salto si dividono in due gruppi:

- **le istruzioni di *salto condizionato***
- **le istruzioni di *salto incondizionato*.**

Istruzione di salto incondizionato

Formato:

```
JMP  destinazione
```

Funzionamento:

Un salto incondizionato è un salto che viene sempre eseguito, senza il controllo di alcuna condizione.

L'operando destinazione contiene l'indirizzo dell'istruzione a cui il processore salta.

Tale indirizzo può essere espresso sotto forma di:

- **etichetta di un'istruzione**
- **indirizzamento indiretto.**

Tipi di salti incondizionati

L'istruzione `JMP` permette di gestire due tipi di salto:

- **i *salti diretti***
- **i *salti indiretti.***

Salti diretti

Nei salti diretti l'indirizzo destinazione specifica l'indirizzo a cui saltare.

Esistono tre tipi di salti diretti:

- **di tipo *short***
- **di tipo *near***
- **di tipo *far*.**

Salti diretti
(segue)

- **Nei salti diretti di tipo *short* l'istruzione macchina è codificata in modo che in un solo byte è contenuto il *displacement* tra il contenuto attuale dell'`IP` e l'*offset* dell'istruzione a cui saltare.**
- **Nei salti di tipo *near* l'istruzione macchina è codificata in modo che il *displacement* tra il contenuto attuale dell'`IP` e l'*offset* dell'istruzione a cui saltare è contenuto in una word.**
- **Nei salti di tipo *far* l'istruzione macchina è codificata in modo da contenere su due word l'*indirizzo intero* dell'istruzione a cui saltare (offset e registro di segmento). I salti di tipo *far* causano la modifica sia del registro `IP` sia del registro `CS`.**

Salti indiretti

Nei *salti indiretti* l'indirizzo *destinazione* non specifica l'indirizzo, ma fornisce un puntatore all'istruzione a cui saltare.

Esempi

```
JMP   AX
JMP   WVAR
JMP   TABLE[BX]
```

Esempio di salto indiretto

```
        . code
        mov ax, salto
        jmp ax
        mov dl, '1'
        jmp fine

salto:
        mov dl, '2'

fine:
        mov ah, 2
        int 21h
```

Costrutti di tipo CASE

I *salti indiretti* possono essere utilizzati per implementare i costrutti di tipo `CASE`.

Esempio

```
switch(var)
        {
        case '1':codice_1;
                        break;
        case '2':codice_2;
                        break;
        case '3':codice_3;
                        break;
         }
```

Soluzione Assembler

```
        .DATA
TAB     DW   lab_1
        DW   lab_2
        DW   lab_3
        ...
        .CODE
        ...
        DEC  VAR      ; decremento il valore di VAR
        MOV  BX, VAR  ; BX assume un valore
                      ; compreso tra 0 e 2
        SHL  BX, 1    ; BX = BX * 2
        JMP  TAB[BX]  ; salto ad uno degli indirizzi
                      ; contenuti in TABELLA
                      ; a seconda del valore di BX
```

```
lab_1:    ...
          JMP continue
lab_2:    ...
          JMP continue
lab_3:    ...
          JMP continue
          ...
continue:
```

Codifica dell'indirizzo di salto in JMP

L'assemblatore permette la massima libertà nel tipo di salto effettuato attraverso l'istruzione `JMP`.

Nel caso di salti diretti l'assemblatore ottimizza il tipo di salto generando l'istruzione macchina opportuna.

L'assemblatore calcola innanzitutto il *displacement* tra l'istruzione successiva a quella di salto e l'istruzione destinazione:

- **se la distanza (in byte) è compresa tra -128 e +127 genera un'istruzione macchina di salto di tipo *short*;**
- **se la distanza è maggiore, ma l'istruzione destinazione appartiene allo stesso segmento di codice genera un'istruzione di salto di *tipo near*;**
- **se l'istruzione destinazione appartiene ad un diverso segmento di codice genera un'istruzione macchina di salto di *tipo far*.**

Istruzioni di salto condizionato

Le istruzioni di salto condizionato si dividono in tre gruppi:

- **quelle in cui la condizione è relativa ad un singolo flag;**
- **quelle in cui la condizione è relativa al risultato di un confronto**
- **quelle in cui la condizione riguarda il contenuto del registro CX.**

Process Status Word (PSW)

È composta da 16 bit, ma solo 9 di questi sono usati. Ogni bit corrisponde ad un flag.

I flag si dividono in:

- **flag di condizione**
- **flag di controllo.**

				OF	DF	IF	TF	SF	ZF		AF		PF		CF

Flag di condizione

Vengono automaticamente scritti al termine di varie operazioni:

- **SF (*Sign Flag*): coincide con il MSB del risultato dopo un'istruzione aritmetica**
- **ZF (*Zero Flag*): vale 0 se il risultato è nullo, 1 altrimenti**
- **PF (*Parity Flag*): vale 1 se il numero di 1 negli 8 bit meno significativi del risultato è pari, 0 altrimenti**
- **CF (*Carry Flag*): dopo le istruzioni aritmetiche vale 1 se c'è stato *riporto* (somma) o *prestito* (sottrazione); altre istruzioni ne fanno un uso particolare**
- **AF (*Auxiliary Carry Flag*): usato nell'aritmetica BCD; vale 1 se c'è stato *riporto* (somma) o *prestito* (sottrazione) dal bit 3**
- **OF (*Overflow Flag*): vale 1 se l'ultima istruzione ha prodotto un *overflow*.**

Flag di controllo

Possono venire scritti e manipolati da apposite istruzioni, e servono a regolare il funzionamento di talune funzioni del processore:

- **DF (*Direction Flag*): utilizzato dalle istruzioni per la manipolazione delle stringhe; se vale 0 le stringhe vengono manipolate partendo dai caratteri all'indirizzo minore, se vale 1 a partire dall'indirizzo maggiore**
- **IF (*Interrupt Flag):* se vale 1, i segnali di Interrupt mascherabili vengono percepiti dalla CPU, altrimenti questi vengono ignorati**
- **TF (*Trap Flag*): se vale 1, viene eseguita una *trap* al termine di ogni istruzione.**

L'istruzione CMP

Permette il settaggio dei flag necessari per l'esecuzione di una successiva istruzione di salto condizionato.

Formato:

CMP destinazione, sorgente

Funzionamento:

L'istruzione `CMP` permette di eseguire il confronto tra l'operando destinazione e l'operando sorgente.

L'istruzione `CMP` tratta gli operandi come numeri ed esegue la sottrazione tra l'operando *`destinazione`* e l'operando *`sorgente`*, senza restituirne il risultato.

L'istruzione `CMP` aggiorna opportunamente tutti i flag di condizione.

L'istruzione CMP

(segue)

L'istruzione CMP permette i seguenti confronti:

- **tra due registri**
- **tra una locazione di memoria ed un registro**
- **tra un valore immediato ed un registro**
- **tra un valore immediato ed una locazione di memoria.**

I vincoli da rispettare sono i seguenti:

- **gli operandi del confronto devono avere la stessa lunghezza**
- **non è ammesso il confronto tra due locazioni di memoria**
- **l'operando immediato non può essere l'operando destinazione, ma solo l'operando sorgente.**

Istruzioni di salto condizionato

Formato:

`Jxxx label`

Funzionamento

`xxx` è un suffisso che specifica la *condizione sui flag*.

Il flusso di esecuzione delle istruzioni dipende dal risultato della condizione:

- **se la *condizione è vera*, il processore continua l'esecuzione saltando all'istruzione etichetta dalla `label`;**
- **se la *condizione è falsa*, l'esecuzione del programma continua con la successiva istruzione in sequenza.**

Istruzioni di salto condizionate da un singolo flag

Per ogni flag esistono due istruzioni di salto condizionato:

- **una che *esegue il salto se il flag vale 1***
- **una che *esegue il salto se il flag vale 0*.**

Istruzioni di salto condizionate da un singolo flag

(segue)

JZ	**Salta se ZF = 1**
JNZ	**Salta se ZF = 0**
JS	**Salta se SF = 1**
JNS	**Salta se SF = 0**
JO	**Salta se OF = 1**
JNO	**Salta se OF = 0**
JC	**Salta se CF = 1**
JNC	**Salta se CF = 0**
JP o JPE	**Salta se PF = 1**
JNP o JPO	**Salta se PF = 0**

Istruzioni di salto condizionate da un singolo flag

(segue)

L'eccezione è data dal flag `AF` (*auxiliary carry*): non esiste nessuna istruzione che *testa* il suo valore: le istruzioni per l'aritmetica BCD, che usano il flag `AF`, *testano* automaticamente il suo valore ed il programmatore non ha dunque bisogno di alcuna istruzione per controllarne il valore.

Le istruzioni di salto condizionato dal risultato di un confronto

Formato

```
CMP destinazione, sorgente
Jxxx label
```

Funzionamento

Il salto è condizionato dall'esito del confronto.

Anche queste istruzioni verificano una condizione sui flag; in generale tale condizione coinvolge più di un flag.

Le istruzioni di salto condizionato dal risultato di un confronto

(segue)

Esistono due insiemi di istruzioni di salto condizionato dipendenti dal risultato di un confronto:

- **un insieme per il *confronto tra numeri con segno***
- **un insieme per il *confronto tra numeri senza segno*.**

Confronto tra numeri con segno

JL o JNGE	Salta se *destinazione < sorgente*
JG o JNLE	Salta se *destinazione > sorgente*
JLE o JNG	Salta se *destinazione <= sorgente*
JGE o JNL	Salta se *destinazione >= sorgente*
JE	Salta se *destinazione = sorgente*
JNE	Salta se *destinazione <> sorgente*

Confronto tra numeri senza segno

JB o JNAE	Salta se *destinazione < sorgente*
JA o JNBE	Salta se *destinazione > sorgente*
JBE o JNA	Salta se *destinazione <= sorgente*
JAE o JNB	Salta se *destinazione >= sorgente*
JE	Salta se *destinazione = sorgente*
JNE	Salta se *destinazione <> sorgente*

Istruzioni di salto condizionato dal contenuto di CX

L'istruzione JCXZ controlla il contenuto del registro CX.

Formato:

```
JCXZ        label
```

Funzionamento

L'istruzione esegue il salto all'istruzione avente etichetta `label` se il contenuto del registro CX è pari a 0.

Codifica dell'indirizzo di salto nel caso di salto condizionato

Un'istruzione di salto condizionato può eseguire un salto solo ad indirizzi nello stesso segmento di codice e con una distanza limitata dall'istruzione di salto.

Per ragioni di efficienza, l'indirizzo dell'istruzione di salto è infatti codificato in linguaggio macchina in *un unico byte*, nel quale è memorizzato *un numero con segno* che indica la distanza in byte tra l'istruzione successiva a quella di salto e l'istruzione destinazione.

Poiché in un byte con segno il numero può variare da -128 a +127, l'istruzione a cui salta un'istruzione di salto condizionato deve stare all'interno di questo intervallo.

Salto condizionato di tipo *lungo*

Nel caso in cui ci si trovi in condizione di effettuare un salto ad un'istruzione localizzata ad una distanza superiore a quanto rappresentabile in un unico byte occorre eseguire il salto di tipo lungo mediante un salto incondizionato.

Esempio

```
controllo:JC    inv_data   (errore !!)
          ...
inv_data:
```

Invece:

```
controllo:JNC   good_data
          JMP   inv_data
good_data:...
inv_data: ...
```

Esercizio

Ricerca del massimo in un vettore di interi positivi.

```
#define LUNG 10
main()
{ int i, massimo = 0;
  unsigned int vett[LUNG];
  ...
  for (i=0 ; i < LUNG ; i++)
   if (vett[i] > massimo)
       massimo = vett[i];
...
}
```

Soluzione Assembler

```
LUNG        EQU   10
            ...
            .DATA
VETT        DW    LUNG DUP (?)
max         DW    ?
            .CODE
            ...
            LEA   SI, VETT    ; copia dell'offset di VETT
            MOV   AX, 0
            MOV   CX, LUNG
ciclo:      CMP   AX, [SI]    ; VETT[SI] > AX ?
            JAE   scansione   ; No: va a scansione
            MOV   AX, [SI]    ; Sì: aggiornamento di max
scansione:  ADD   SI, 2
            DEC   CX
            CMP   CX, 0
            JNE   ciclo
            MOV   max, AX
```

Esercizio

Calcolo del numero di lettere minuscole in una stringa.

```
#define LUNG 20
main()
{ int i;
  char vett[LUNG], minuscole = 0;
  for (i=0 ; i< LUNG ; i++)
   if ((vett[i] >= 'a') && (vett[i] <= 'z'))
       minuscole++;
}
```

Soluzione Assembler

```
car_a      EQU "a"
car_z      EQU "z"
           ...
           .DATA
VETT       DB   "Nel mezzo del cammin di nostra... "
LUNG       EQU  $-VETT
MINUSCOLE  DW   ?
           .CODE
```

```
      ...
      MOV  SI, 0
      MOV  AX, 0   ; contatore di minuscole
      MOV  CX, LUNG
ciclo:CMP  VETT[SI], car_a  ; VETT[SI] >= 'a' ?
      JB   salta            ; No: va a salta
      CMP  VETT[SI], car_z  ; Sì: VETT[SI] <= 'z' ?
      JA   salta            ; No: va a salta
      INC  AX      ; Sì: carattere minuscolo
salta:INC  SI
      DEC CX
      CMP CX, 0
      JNE  ciclo        MOV  MINUSCOLE, AX
      ...
```

Istruzioni di iterazione

Le istruzioni *LOOP*, *LOOPE* e *LOOPNE* facilitano l'implementazione di costrutti di tipo iterativo.

L'istruzione LOOP

L'istruzione LOOP permette di gestire in maniera semplice una sequenza di istruzioni ripetuta un numero definito di volte.

Formato:

LOOP label

Funzionamento:

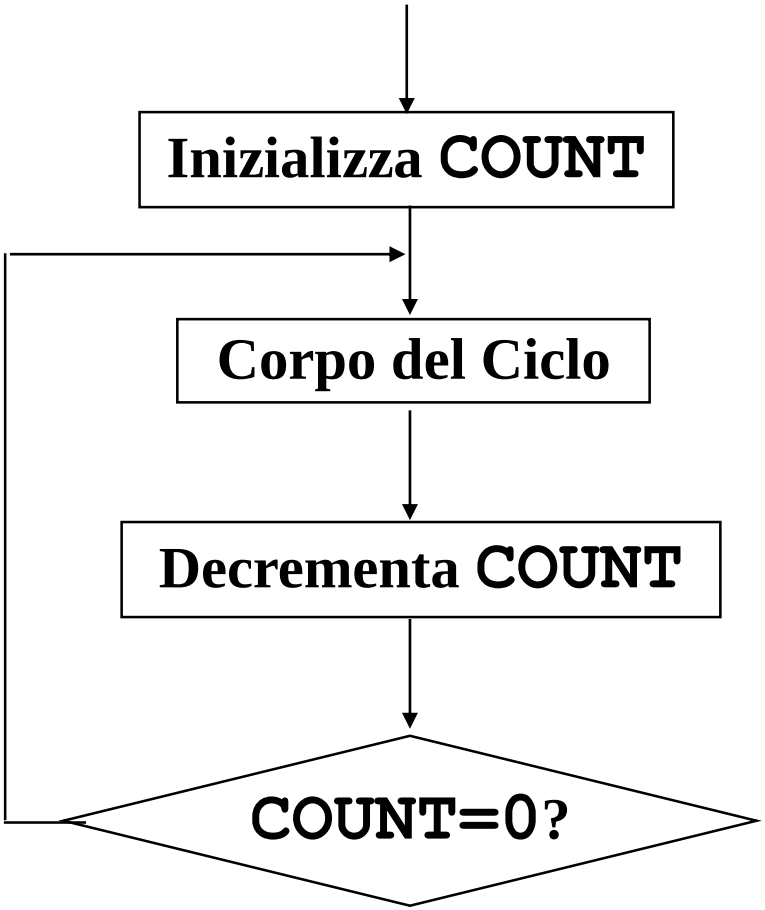
All'atto dell'esecuzione dell'istruzione LOOP, il processore esegue le seguenti operazioni:

- **decrementa di una unità il registro CX**
- **esegue il controllo sul contenuto del registro CX:**
  - **se il valore di CX è diverso da 0, salta all'istruzione avente etichetta *label*;**
  - **altrimenti esegue l'istruzione successiva.**

LOOP: Esempio

```
        MOV     CX, N
BEGIN:  ...
        DEC     CX
        JNZ     BEGIN
```

```
        MOV     CX, N
BEGIN:  ...
        LOOP BEGIN
```



L'istruzione LOOPE

Formato:

LOOPE label

Funzionamento:

All'atto dell'esecuzione dell'istruzione LOOPE, il processore esegue le seguenti operazioni:

- **decrementa di un'unità il registro CX;**
- **esegue il controllo sul registro CX e sul flag ZF:**
  - **se il valore di CX è diverso da 0 *AND* il flag ZF è uguale a 1, salta all'istruzione avente etichetta *label*;**
  - **altrimenti (se il valore di CX è uguale a 0 *OR* il flag ZF è uguale a 0) allora esegue l'istruzione successiva.**

L'istruzione LOOPNE

Formato:

LOOPNE label

Funzionamento:

All'atto dell'esecuzione dell'istruzione `LOOPE`, il processore esegue le seguenti operazioni:

- **decrementa di un'unità il registro `CX`;**
- **esegue il controllo sul registro `CX` e sul flag `ZF`:**
  - **se il valore di `CX` è diverso da `0` *AND* il flag `ZF` è uguale a `0`, salta all'istruzione avente etichetta *label*;**
  - **altrimenti (se il valore di `CX` è uguale a `0` *OR* il flag `ZF` è uguale a 1) allora esegue l'istruzione successiva.**

Esercizio

Ricerca di un numero all'interno di una tabella

```
#define LUNG    100
#define NUM     -1
main()
{
int i;
int vett[LUNG];
for (i = 0 ; i < LUNG ; i++)
      if (vett[i] == NUM) break;
}
```

Soluzione Assembler

```
LUNG   EQU 100
NUM    EQU -1
       ...
       .DATA
VETT   DW  LUNG DUP (?)
       ...
       .CODE
       ...
       MOV SI, -2
       MOV CX, LUNG
ciclo:ADD SI, 2
       CMP VETT[SI], NUM ; VETT[SI] = NUM ?
       LOOPNE ciclo      ; No e CX<>0: =>
                         ; ripeti ciclo
       ...
```

Esercizio

Realizzare un programma che scandisca un vettore di interi alla ricerca del primo numero diverso da 0.

```
#define LUNG 100
main()
{
int i;
int vett[LUNG];
...
for (i = 0 ; i < LUNG ; i++)
     if (vett[i]) break;
}
```

Soluzione Assembler

```
LUNG   EQU 100
       ...
VETT   DW  LUNG DUP (?)
       ...
       MOV SI, -2
       MOV CX, LUNG
ciclo:ADD SI, 2
       CMP VETT[SI], 0; VETT[SI] = 0 ?
       LOOPE ciclo     ; Sì e CX <> 0 ripeti il ciclo
       ...
```

Le istruzioni che modificano i flag

Il processore mette a disposizione alcune istruzioni che permettono di modificare il valore dei flag.

In generale non è necessario modificare i flag di stato, in quanto essi servono per essere letti per controllare gli effetti di un'istruzione.

In alcuni casi può essere necessario forzare il valore del flag `CF`.

Esistono tre istruzioni che permettono di modificare il valore di `CF`:

```
STC   ; fissa CF a 1
CLC   ; fissa CF a 0
CMC   ; complementa il valore di CF
```

Le istruzioni che modificano i flag

(segue)

È possibile modificare il valore dei flag di controllo `DF` (*direction flag*) e `IF` (*interrupt flag)*.

```
STD   ; fissa DF a 1
CLD   ; fissa DF a 0
STI   ; fissa IF a 1
CLI   ; fissa IF a 0
```