Algoritmi di salto (selezione e iterazione) Assembly 8086

Docente: Prof. Mario Perna **Email:** prof.perna.mario@darzo.net **Revisione:** 04.04.2025

Introduzione sulle Istruzioni di confronto

In queste slides studieremo le istruzioni di controllo di un programma.

La **selezione e l'iterazione** sono controllate dai **salti condizionati e incondizionati** che, insieme **con il controllo dei flag**, forniscono al programmatore un metodo per ottenere cicli e selezioni, analogamente a quanto avviene con i linguaggi evoluti.

L’istruzione di confronto CMP

Permette di confrontare tra loro due valori espressi da due operandi, effettua una **sottrazione** tra il secondo operando e il primo senza memorizzare il risultato.

Il suo scopo è quello di alterare i **flag** per utilizzarli nelle istruzioni di salto condizionato che seguono l’istruzione CMP stessa.

```
CMP operando1,operando2
```

Il **primo operando** può essere un **registro,** una **cella di memoria** indirizzata in modo diretto, indiretto.

il **secondo operando** può consistere in un **registro,** una **cella di memoria** indirizzata in modo diretto, indiretto o un **dato immediato**.

Alterazione dei flag in funzione agli operandi

Sebbene tutti i flag vengano alterati, i più significativi sono ZF e CF:

- **ZF** = **0** quando i due operandi sono diversi, cioè quando **operando1 <> operando2**.

- **ZF** = **1** quando i due operandi sono uguali, cioè quando **operando1 = operando2**.

- **CF** = **0** quando il secondo operando è minore o uguale al primo, cioè quando **operando1 >= operando2**.

- **CF** = **1** quando il secondo operando è maggiore del primo, cioè quando **operando1 < operando2**.

Modalità di indirizzamento

Sebbene tutti i flag vengano alterati, i più significativi sono ZF e CF:

- CMP **registro**,**registro**
- CMP **registro**,**memoria**
- CMP **registro**,**immediato**
- CMP **memoria**,**registro**
- CMP **memoria**,**immediato**

Flag alterati: **ZF, CF, SF, OF, PF**

Esempio di utilizzo

```
CMP AL,BL              ;confronta due registri (8 bit)
                       ;se ZF = 1 e CF = 1 allora significa che AL = BL
                       ;se ZF = 1 e CF = 0 allora significa che AL = BL
                       ;se ZF = 0 e CF = 1 allora significa che AL < BL
                       ;se ZF = 0 e CF = 0 allora significa che AL > BL
CMP CX,BX              ;confronta il registro CX con BX (16 bit) identiche modalità
                       ;viste sopra (16 bit)
CMP DL,02h             ;confronta il registro DL con il dato immediato 02h (8 bit)
```

Istruzione di salto incondizionato (JMP)

I salti incondizionati (JMP, jump) saltano all'etichetta indicata senza verificare il contenuto dei flag (potremmo dire che saltano qualunque cosa accada).

Tecnicamente, un'istruzione di salto copia l'operando indicato, rappresentato da un'etichetta tradotta in un indirizzo, nel registro **IP**.

Al posto dell'etichetta è anche possibile indicare un **indirizzo fisico di memoria**, chiamato **spiazzamento.**

Le **etichette o label** possono essere poste in un punto qualunque del programma e devono terminare con i due punti, per esempio "**ciclo:**".

Istruzione di salto incondizionato (JMP)

A cosa serve quindi l'etichetta ?

Mediante un'etichetta si realizza un salto **relativo**, in quanto il salto riguarderà un numero di **celle** successive o precedenti indicate dall'etichetta.

Possiamo anche effettuare un salto **assoluto** indicando espressamente **l'indirizzo** di offset della cella da raggiungere mediante una cella di memoria oppure un registro a 16 bit.

```
            JMP etichetta oppure spiazzamento
            ...
etichetta:
```

Istruzione di salto incondizionato (JMP)

Modalità di indirizzamento:

- JMP **etichetta**
- JMP **[registro]**
- JMP **memoria**

Flag alterati: **nessuno**

Esempio di utilizzo

```
JMP ciclo                ;salto incondizionato relativo all'etichetta chiamata ciclo:
JMP SI                   ;salto incondizionato assoluto all'istruzione di indirizzo indicato
                         ;dal registro SI
JMP Word Ptr[DI]         ;salto incondizionato assoluto all'istruzione di indirizzo indicato
                         ;dal registro DI
infinito: NOP            ;salto incondizionato di un ciclo infinito
JMP infinito
```

Istruzione di salto condizionato (J)

L’istruzione di salto condizionato **J** determina una modifica nella sequenza di esecuzione del programma a seconda dell'esito di una determinata condizione:

- se quest'ultima è soddisfatta il controllo passa alla riga in cui è presente l’etichetta indicata come operando dell’istruzione;
- in caso contrario il controllo passa all'istruzione successiva.

Le condizioni valutate dall’istruzione sono relative al **contenuto del flag** indicato; esistono infatti numerose istruzioni di salto condizionato, una per ogni flag che deve essere verificato.

Istruzione di salto condizionato (J)

Per effettuare una condizione che verifichi, per esempio, se un registro contiene zero è necessario anteporre all'istruzione di salto condizionato un'istruzione CMP tra i due valori, in modo che il flag di zero venga settato se i due valori sono uguali:

```
CMP AL,0h      ;se AL = 0 viene settato il flag di zero
JZ salto       ;se il flag di zero è settato il controllo passa all'etichetta salto:
```

La notazione **0h** rappresenta **il numero zero in formato esadecimale**.

Istruzione di salto condizionato (J)

```
JA    ;Jump  if above (salta se CF=0 e ZF=0, cioè se op1>op2)*
JAE   ;Jump  if above or equal (salta se CF=0, cioè se op1>=op2)*
JB    ;Jump  if below (salta se CF=1, cioè se op1<op2)*
JBE   ;Jump  if below or equal (salta se CF=1 or ZF=1, cioè se op1<=op2)*
JC    ;Jump  if carry (salta se CF=1)
JE    ;Jump  if equal (salta se ZF=1, cioè se op1=op2)
JG    ;Jump  if greater (salta se ZF=0 and SF=OF, cioè se op1>op2)**
JGE   ;Jump  if greater or equal (salta se SF=OF, cioè se op1>=op2)**
JL    ;Jump  if less (salta se SF<>OF, cioè se op1<op2)**
JLE   ;Jump  if less or equal (salta se ZF=1 or SF<>OF, cioè se op1<=op2)**
JNA   ;Jump  if not above (salta se CF=1 or ZF=1, cioè se op1 non è > op2)*
JNAE  ;Jump  if not above or equal (salta se CF=1, cioè se op1 non è >= op2)*
JNB   ;Jump  if not below (salta se CF=0, cioè se op1 non è < op2)*
JNBE  ;Jump  if not below or equal (salta se CF=0 and ZF=0, cioè se op1 non è <= op2)*
JNC   ;Jump  if not carry (salta se CF=0)
JNE   ;Jump  if not equal (salta se ZF=0, cioè se op1<>op2)
JNG   ;Jump  if not greater (salta se ZF=1 or SF<>OF, cioè se op1 non è > op2)*
JNGE  ;Jump  if not greater or equal (salta se SF<>OF, cioè se op1 non è >= op2)*
JNL   ;Jump  if not less (salta se SF=OF, cioè se op1 non è < op2)*
JNLE  ;Jump  if not less or equal (salta se ZF=0 and SF=OF, cioè se op1 non è <= op2)*
JNO   ;Jump  if not overflow (salta se OF=0)
JNP   ;Jump  if not parity (salta se PF=0)
JNS   ;Jump  if not sign (salta se SF=0)
```

Istruzione di salto condizionato (J)

```
JNZ   ;Jump  if not zero (salta se ZF=0)
JO    ;Jump  if overflow (salta se OF=1)
JP    ;Jump  if parity (salta se PF=1)
JPE   ;Jump  if parity even (salta se PF=1)
JPO   ;Jump  if parity odd (salta se PF=0)
JS    ;Jump  if sign (salta se SF=1)
JZ    ;Jump  if zero (salta se ZF = 1)

*Opera su numeri senza segno.
**Opera su numeri con segno.
```

Istruzione di incremento (INC) e decremento (DEC)

Le istruzioni di INC e DEC permettono, come suggeriscono le parole, l'incremento o il decremento di un registro o di una cella di memoria.

La sintassi utilizzata è la seguente:

Per l'incremento:

INC destinazione

Per il decremento:

DEC destinazione

destinazione = registro o variabile di memoria.

Flag alterati: **ZF, SF, OF, PF**

Flag non alterato: **CF (Carry Flag)**

Quando usarli ?

- Per contare
- Per creare cicli
- Per aggiornare indici

ESEMPIO

```
MOV AX, 5    ; AX = 5
INC AX       ; AX = 6

MOV BX, 10   ; BX = 10
DEC BX       ; BX = 9
```

Cicli con istruzione LOOP

L'istruzione **LOOP** consente di effettuare un ciclo a contatore ripetuto un numero di volte pari al contenuto del registro **CX**. A ogni istruzione **LOOP** avviene un decremento di **CX**; **se CX è zero il ciclo termina, altrimenti effettua un salto all'etichetta indicata.**

La sintassi è la seguente:

```
LOOP ciclo      ;salta se CX <> 0
LOOPZ ciclo     ;salta se CX <> 0 e ZF = 1
LOOPNZ ciclo    ;salta se CX <> 0 e ZF = 0
```

Modalità di indirizzamento:

- LOOP etichetta

Flag alterati: **NESSUNO**

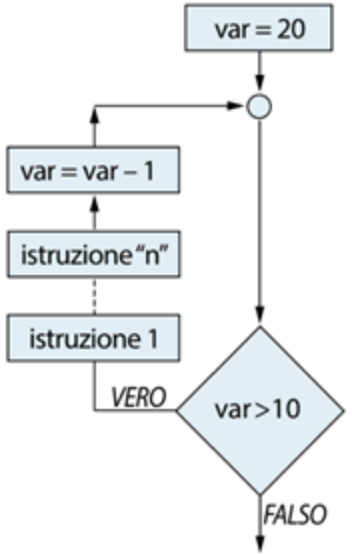
Esempio Pratico

In questo esempio vediamo come applicare l'istruzione **LOOP** per ripetere un ciclo 20 volte.

```
;ciclo ripetuto per 20 volte
                MOV CX,14h        ;inizializzazione del contatore CX
ciclo:          ...               ;istruzioni da ripetere
                LOOP ciclo        ;salta se CX <> 0

;il ciclo sopra equivale alle seguenti istruzioni realizzate senza costrutto LOOP
                MOV CX,14h        ;inizializzazione del contatore CX
ciclo:          ...               ;istruzioni da ripetere
                DEC CX            ;decrementa il contenuto di CX di 1 unità
                JNZ ciclo         ;salta se CX <> 0
```

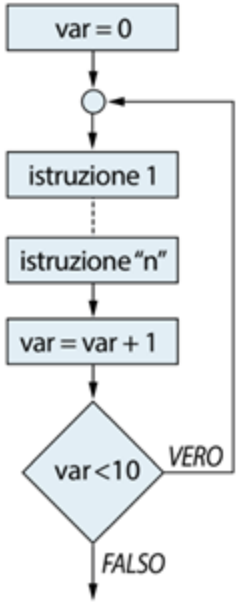
Il ciclo while

CODIFICA IN LINGUAGGIO C	CODIFICA IN ASSEMBLY			FLOW-CHART
...	.DATA			
int var;	var	DB 14h	;dichiarazione variabile di tipo ;Byte e inizializzazione	
var=20;			;a 20	
	.CODE			
while (var>10)	ciclo:	CMP var,0Ah	;confronto con 0	
{		JBE fine	;se var<=10 salta ;all'etichetta fine	
var=var-1;		DEC var	;decremento variabile di 1 unità	
...		...		
istruzioni blocco		istruzioni blocco		
		...		
...		JMP ciclo	;salto incondizionato che ;ritorna a "while"	
}	fine:	...	;prosecuzione programma	

Il ciclo while

Nella codifica in Assembly il ciclo viene ripetuto fino a quando la variabile non risulta essere minore o uguale a 10. Il controllo iniziale prevede una condizione invertita rispetto a quella presentata nella codifica in linguaggio C; infatti il salto condizionato JBE opera un salto quando la variabile contiene un numero minore o uguale a 10, esattamente al contrario rispetto al linguaggio evoluto.

Il ciclo do while

CODIFICA IN LINGUAGGIO C	CODIFICA IN ASSEMBLY	FLOW-CHART
...	.DATA	
int var=0;	var DB 0h ;dichiarazione variabile di ;tipo Byte e inizializzazione	
do {	.CODE	
... var=var+1;	ciclo: INC var ;incremento variabile di 1 unità	
...	...	
istruzioni blocco	istruzioni blocco	
...	...	
} while (var<10)	CMP var,0Ah ;confronto con 10	
	JB ciclo ;se var<10 salta all'etichetta ciclo	
...	... ;prosecuzione programma	

Il ciclo do while

Il codice esegue un ciclo post condizionato per vero ripetuto mentre la variabile è minore di 10: all'interno del ciclo essa viene incrementata, in tal modo il ciclo verrà ripetuto 10 volte (da 0 a 9). Il costrutto post condizionato svolto in linguaggio Assembly viene paragonato al relativo costrutto scritto in un linguaggio C. Il programma esegue il blocco di istruzioni mentre la condizione è verificata, cioè se la variabile è minore di 10, altrimenti esce.

ESERCIZIO LOOP

```
TITLE loop1
.MODEL small
.STACK 200h

.DATA
   str1 DB 13,10,'Ciao mondo','$' ; CR + LF + testo + terminatore

.CODE
.STARTUP

   MOV CX, 10    ; contatore per il ciclo
```

SEGUE…

```
ciclo:
   MOV AH, 09h
   ; DS:DX deve puntare alla stringa
   MOV DX, OFFSET str1

   INT 21h

   LOOP ciclo

   MOV AH, 4Ch
   INT 21h

END
```