Controllo del flusso di istruzioni all'interno di un programma

Per poter modificare la sequenza di esecuzione delle istruzioni all'interno di un programma in dipendenza dei risultati duna elaborazione si possono utilizzare le istruzioni di salto.

Un'istruzione di salto consente di costringere il program counter di un microprocessore a saltare ad un indirizzo di memoria diverso da quello che contiene per autoincremento e quindi di eseguire un nuovo blocco di istruzioni in luogo di quello che sarebbe stato svolto perché memorizzato nelle locazioni di memoria immediatamente successive a quella in cui risiede l'istruzione corrente.

Per effettuare un salto bisognerebbe conoscere l'indirizzo della locazione di memoria in cui risiede l'istruzione cui si vuole saltare. Ciò non è strettamente necessario. Infatti possiamo identificare, nel programma, l'istruzione che ci interessa mediante un'etichetta.

Un'etichetta è un nume che si associa ad una istruzione con il seguente formato

Nome: istruzione

La scelta dell'etichetta è del tutto libera. L'unica limitazione è chiaramente che per istruzioni diverse si debbono utilizzare etichette diverse.

Sarà il programma assemblatore a calcolare l'indirizzo della istruzione cui si vuole saltare e a sostituirlo all'etichetta.

Un primo esempio di istruzione di salto è l'istruzione di salto incondizionato

JMP etichetta

Vediamone subito un'applicazione

```
dati segment
numero1 dw 10
numero2 dw 10
dati ends
codice segment
assume cs:codice, ss:stack, ds:dati
inizioprogramma:
mov ax, dati
mov ds,ax
;questo programma sommerà ax e dx
;all'infinito per l'eternità
;non serve ad altro che a far vedere l'effetto
;di un salto incondizionato
mov ax,numero1
mov dx,numero2
cicloinfinito:
add ax,dx
jmp cicloinfinito
mov ah,4ch
int 21h
codice ends
end inizioprogramma
```

Questo programma è chiaramente un pretesto il cui scopo è soltanto quello di far vedere l'effetto di un salto incondizionato. Abbiamo cerato un loop infinito che durerà, cioè, per l'eternità, poiché l'istruzione di salto incondizionato fa in modo che si ritorni sempre ad eseguire l'istruzione di somma, per cui non verranno mai eseguite le istruzioni che

consentono di terminare il programma e restituire il controllo al sistema operativo. Per vedere un uso più razionale dei salti incondizionati dovremo attendere di aver illustrato altri elementi.

vedi simulazione avi

vedi simulazione autoeseguibile

Nella maggior parte dei casi il salto ad un altro blocco di programma non deve avvenire in ogni caso ma soltanto se si è verificata una particolare condizione nell'evoluzione del programma. In tal caso si utilizzano le istruzioni di salto condizionato.

Tali istruzioni eseguono il salto all'istruzione individuata mediante un'etichetta soltanto se la condizione che essi esprimono è verificata.

Tali condizioni si basano sul valore di uno dei flag. Se la condizione non è verificata il salto non viene effettuato e il programma prosegue con l'istruzione che segue immediatamente in memoria l'istruzione di salto.

Un esempio sono le istruzioni di salto condizionato dal valore del flag di zero

JZ salta se il flag di zero si trova ad 1

JNZ salta se il flag si trova a 0.

Un esempio è il seguente programma

```
dati segment
quantevolte db 100
dati ends
codice segment
assume cs:codice, ss:stack, ds:dati
inizioprogramma:
mov ax, dati
mov ds,ax
;questo programma non fa assolutamente niente
;decrementa al più volte e termina soltanto
;quando al va a zero
mov al,quantevolte
ciclo:
dec al
jnz ciclo
mov ah,4ch
int 21h
codice ends
end inizioprogramma
```

Questo programma effettua continui decrementi di al e si deve bloccare quando al giunge a zero. Poiché tale evento sarà segnalato dal flag di zero che assumerà il valore 1, noi effettuiamo un ciclo mediante un'istruzione di salto condizionato che consente di tornare ad eseguire l'istruzione di decrementa se il flag di zero si trova a zero (cioè al è ancora diverso da zero). Quando AL giunge a zero, il flag si porta ad 1. A quel punto la condizione non è più verificata, il salto non viene più eseguito e si possono eseguire le

istruzioni seguenti che interrompono il programma e restituiscono il controllo al sistema operativo.

vedi simulazione avi

vedi simulazione autoeseguibile

Altre condizioni possibili sono sul flag di segno

JS salta se il flag di segno S è ad 1

JNS salta se il flag di segno è a zero

Oppure sul flag di overflow

JO salta se il flag di overflow è ad 1

JNO salta se il flag di overflow è a 0

JC salta se il flag di carry è ad 1

JNC salta se il flag di carry è a 0

JP salta se il flag di parità è ad 1

JNP salta se il flag di parità è a 0

I flag vengono influenzati da operazioni aritmetiche ma anche da operazioni di confronto fra dati che si effettuano con l'istruzione CMP

CMP operando1, operando2

Tale istruzione confronta i due operandi facendone la sottrazione. Si tratta però di una sottrazione fittizia nel senso che il suo risultato non viene memorizzato e i due operandi non vengono in realtà modificati. Vengono però influenzati i flag.

Tenendo presente ciò, se si fa precedere i codici mnemonici per i salti possono essere anche

JE (Jump If Equal) cioè salta se operando1=operando2 (naturalmente prima ci deve essere stata un'istruzione di confronto), cioè salta se flag di zero=1 il che è equivalente all'istruzione JZ;

JNE (Jump If Not Equal) cioè salta se operando1<>operando2, cioè salta se flag di zero=0 il che è equivalente all'istruzione JNZ;

per il confronto fra numeri con segno

JG (Jump If Greater) o equivalentemente JNLE (Jump If Not Less Or Equal) cioè salta se operando1>operando2 (naturalmente prima ci deve essere stata un'istruzione di confronto), cioè salta se flag di zero=0 (perché i due operandi non devono essere uguali) oppure è verificata la condizione che il flag di segno coincida con l flag di overflow, infatti se il primo operando è maggiore del secondo operando si deve avere SF = 0 ma questo è vero se non si verifica overflow (quindi anche OF = 0): consideriamo infatti l'esempio del confronto fra 80 e –80, chiaramente il primo operando è superiore al secondo operando, ma se andiamo ad eseguire la sottrazione in binario avremo

01010000 (+80) –

10110000 (-80) =

———————

10100000

cioè nonostante il primo operando sia superiore la secondo operando si ha che il risultato è negativo cioè si ha S =1, am il problema sta nel fatto che si è avuto overflow (80 – (-80) = 80+80 = 160), ma in questo caso me ne accorgo poiché il flag di segno coincide con il flag di overflow , in definitiva i due flag devono sempre coincidere se il primo operando supera il secondo operando;

JLE (Jump If Less or Equal) o equivalentemente JNG (Jump If Not Greater) cioè salta se operando1<=operando2 (naturalmente prima ci deve essere stata un'istruzione di confronto), cioè salta se flag di zero=1 (se i due operandi sono uguali) oppure il flag di segno è diverso dal flag di overflow (se il primo operando è minore del secondo operando), infatti in quest'ultimo caso deve essere SF = 1 ma non si deve aver avuto overflow altrimenti si ricade nell'esempio precedente dove si ha SF = 1 nonostante il primo operando sia superiore al secondo;

JGE (Jump If Greater or Equal) o equivalentemente JNL (Jump If Not Less) cioè salta se operando1>=operando2 (naturalmente prima ci deve essere stata un'istruzione di confronto), cioè salta se flag di S=0 se non c'è stato overflow oppure se SF = 1 ci sia stato overflow (come nell'esempio precedente), quindi se SF = OF;

per il confronto fra numeri senza segno o carattere non si può far riferimento al flag di segno, infatti ad esempio se confronto 236 con 10 è chiaro che il primo è superiore al secondo se li considero dati senza segno ma per il computer 236 si rappresenta come 11101100 per cui quando vado a sottrarre 10 si ha

11101100 (236) –

00001010 (10)

———————

11100010

per cui analizzando il segno ne dovremmo trarre la conclusione errata che 236<10. in questo caso l'unico modo di confrontare i dati è di analizzare il flag di carry. Facendo esempi si può vedere che, nella sottrazione, si genera un prestito sul ottavo bit se il primo dato è inferiore al secondo dato. Avremo allora che

JB (Jump If Below) o equivalentemente JNAE (Jump If Not Above Or Equal) cioè salta se operando1<operando2 (naturalmente prima ci deve essere stata un'istruzione di confronto), cioè salta se flag di carry = 1 ((perché se il primo operando è minore del secondo operando, la sottrazione fa generare un prestito sull'ottavo bit);

JA (Jump If Above) o equivalentemente JNBE (Jump If Not Below Or Equal) cioè salta se operando1>operando2 (naturalmente prima ci deve essere stata un'istruzione di confronto), cioè salta se flag di zero=0 (perchè i due operandi non sono uguali) e il flag di carry è uguale a 0 (perchè il primo operando è maggiore del secondo operando);

JBE (Jump If Below or Equal) o equivalentemente JNA (Jump If Not Above) cioè salta se operando1<=operando2 (naturalmente prima ci deve essere stata un'istruzione di confronto), cioè salta se CF =1(se il primo operando è maggiore del secondo operando) oppure ZF = 1 (i due operandi sono uguali);

JAE (Jump If Above or Equal) o equivalentemente JNB (Jump If Not Below) cioè salta se operando1>=operando2 (naturalmente prima ci deve essere stata un'istruzione di confronto), cioè salta se flag di carry = 0 ((perché se il primo operando è maggiore o uguale al secondo operando, la sottrazione non fa generare un prestito sull'ottavo bit).