I microprocessori x86-16 sono sostanzialmente costituiti dal 8086 e dal 8088, con la caratteristica di gestire registri a 16 bit e di poter indirizzare complessivamente fino a 1024 Kibyte, suddividendo però la memoria in segmenti da 64 Kibyte. Questa famiglia ha il limite di disporre di pochi registri per usi generali, spesso vincolati a un ruolo preciso, nell'ambito di certe istruzioni.

Dal momento che esiste una grande quantità di modelli di microprocessori compatibili con la vecchia famiglia a 16 bit e dato che sono disponibili simulatori ed emulatori, può essere ancora interessante lo studio della programmazione a 16 bit, riferita al modello x86-16, se non si devono affrontare problematiche relative alla protezione della memoria e a gestioni sofisticate della stessa.

Questo e gli altri capitoli dedicati alla programmazione con i microprocessori x86-16 e l'architettura dell'elaboratore IBM PC dei primi anni 1980, si limitano ad affrontare le questioni che consentono di lavorare con i registri di segmento posti tutti allo stesso valore (salva la possibilità di travasare dei dati da una parte della memoria all'altra). Molte questioni importanti non vengono affrontate e si rimanda ai riferimenti posti alla fine dei capitoli, per gli approfondimenti eventuali, oltre che al capitolo 64, in cui si fa riferimento ai microprocessori x86-32.

Segmenti

Prima di considerare i registri di un microprocessore x86-16, è importante comprendere il concetto di ***segmento***, utilizzato in questo contesto.

Dal momento che i registri sono a 16 bit, con questi si possono rappresentare valori senza segno da zero a 65535; pertanto, dato che la memoria è organizzata in byte, con un registro si può scandire soltanto un intervallo di 64 Kibyte. Per poter scandire lo spazio di

1024 Kibyte, occorrono due registri, in modo da comporre assieme un indirizzo da 20 bit.

Per indirizzare la memoria, a qualunque titolo, nei microprocessori x86-16 è necessario un ***registro di segmento*** e un altro valore che esprima lo scostamento dall'inizio del segmento a cui il contesto si riferisce.

I segmenti possono collocarsi in memoria con una certa libertà, pertanto possono sovrapporsi, parzialmente o completamente. In pratica la memoria viene suddivisa idealmente in ***paragrafi*** (o *click*) da 16 byte ciascuno e i segmenti possono iniziare soltanto all'inizio di un paragrafo. Per garantire che ciò avvenga in questo modo, i registri che sono dedicati a rappresentare l'inizio di un segmento, riportano il numero del paragrafo, ovvero l'indirizzo assoluto di memoria diviso per 16. Questa divisione si ottiene con un semplice scorrimento a destra di quattro bit; pertanto, per ritrovare il valore originale è sufficiente fare lo scorrimento opposto, verso sinistra. Per esempio, il valore $159D_{16}$ contenuto in un registro di segmento, individua in realtà l'indirizzo $159D0_{16}$, pari a 88528_{10}.

Come accennato, per individuare una certa posizione in memoria si usa sempre un registro di segmento e un altro valore che rappresenta lo scostamento a partire dall'inizio del segmento a cui si riferisce il contesto. Per esempio, se il registro di segmento contiene il valore $159D_{16}$ e si specifica lo scostamento $FFFE_{16}$, si sta in pratica facendo riferimento alla posizione di memoria $159D0_{16}$+$FFFE_{16}$, pari a $259CE_{16}$, ovvero 154062_{10}.

Stante questa organizzazione, per indicare in un documento un certo indirizzo di memoria, si può usare la definizione di «indirizzo efficace» e si può scrivere un solo numero, come per esempio $159DF_{16}$.

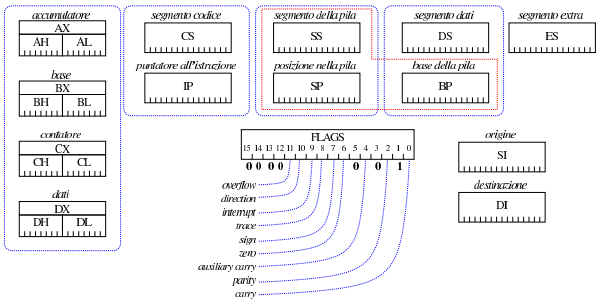
Riquadro u139.1. Valori affiancati e divisi dal simbolo '**:**'.

Nella programmazione a 16 bit, con i microprocessori della famiglia x86, per affiancare due valori a 16 bit si usa normalmente il segno di due punti, come per esempio ***DX*:*AX*** o $159D_{16}$:$FFFE_{16}$.
In generale, questa rappresentazione indica soltanto che si vuole fare riferimento a un numero a 32 bit, formato dall'unione delle due parti indicate, ma il significato che questo numero deve avere va interpretato in base al contesto. Per esempio, ***DX*:*AX*** potrebbe essere il risultato di una moltiplicazione, da prendere numericamente tale e quale, nel senso che il registro ***DX*** rappresenta i 16 bit più significativi; ma in un altro contesto, ***DS*:*SI*** può fare riferimento a un indirizzo che si interpreta come ***DS***·16+***SI***. Nello stesso modo, il numero rappresentato come 1000_{16}:$59DF_{16}$, potrebbe indicare precisamente il valore $100059DF_{16}$, oppure, se si tratta di un indirizzo, composto da segmento e scostamento, andrebbe inteso come $159DF_{16}$.

Registri

I registri dei microprocessori x86-16 sono schematizzati dalla figura successiva. I registri per uso generale, denominati ***AX***, ***BX***, ***CX*** e ***DX***, possono essere utilizzati nella loro interezza o divisi in byte; per esempio si può intervenire nel byte meno significativo di ***AX*** con il nome ***AL*** (*low*) e si può accedere al byte più significativo con il nome ***AH*** (*high*).

Figura u139.2. I registri dei microprocessori x86-16.



I registri di segmento sono: ***CS***, ***DS***, ***SS*** e ***ES***. Il segmento individuato dal registro ***CS*** (*code segment*) è quello in cui si svolge il codice in corso di esecuzione, e il puntatore all'istruzione da eseguire, nell'ambito del segmento codice, è contenuta nel registro ***IP*** (*instruction pointer*, ma noto anche come *program counter* e indicato a volte con la sigla «PC»). Il segmento individuato dal registro ***SS*** (***stack segment***) è quello in cui si trova la pila dei dati, ovvero quella struttura che consente il trasferimento delle variabili alle funzioni o la creazione di variabili locali. L'indice della pila è costituito dal registro ***SP*** (*stack pointer*) e l'indirizzo della base della pila, nell'ambito della funzione in corso di esecuzione, viene annotato convenzionalmente nel registro ***BP*** (*base pointer*). Il segmento individuato dal registro ***DS*** (***data segment***) è quello in cui si trovano i dati correnti, mentre il segmento del registro ***ES*** (***extra segment***) riguarda un'area dati alternativa, utile soprattutto quando si vogliono fare dei trasferimenti di dati tra segmenti differenti.

Convenzionalmente è stato adottato il registro ***BP*** per annotare il riferimento all'inizio della pila di una funzione, per poter accedere agli argomenti attuali o alle variabili locali con un riferimento relativo a tale puntatore. Tuttavia, va osservato che il segmento a cui si riferisce il registro ***BP*** è quello dei dati, ovvero ***DS***, per cui, quando si utilizza ***BP*** per accedere al contenuto della pila, è indispensabile che ***DS*** sia uguale a ***SS***.

Il registro ***FLAGS*** raccoglie gli indicatori disponibili, come descritto nella tabella successiva. In alcuni documenti, tale registro è chiamato *program status word* e abbreviato come «PSW».

Tabella u139.3. Gli indicatori principali contenuti nel registro ***FLAGS***.

Indicatore (*flag*)	Bit	Descrizione
C *carry*	0	È l'indicatore del riporto per le operazioni con valori senza segno. In particolare si attiva dopo una somma che genera un riporto e dopo una sottrazione che richiede il prestito di una cifra (in tal caso si chiama anche *borrow*).
1	1	Riservato.
P *parity*	2	Si attiva quando l'ultima operazione produce un risultato i cui otto bit meno significativi contengono una quantità pari di cifre a uno.
0	3	Riservato.
A *auxiliary carry*	4	È un tipo di riporto ausiliario.
0	5	Riservato.
Z *zero*	6	Viene impostato dopo un'operazione che dà come risultato il valore zero.
S *sign*	7	Riproduce il bit più significativo di un valore, dopo un'operazione. Se il valore è da intendersi con segno, l'indicatore serve a riprodurre il segno stesso.
T *trace*	8	Se è attivo, fa in modo che il microprocessore possa funzionare un passo alla volta.
I *interrupt*	9	Se è attivo, le interruzioni hardware sono abilitate, diversamente risultano bloccate.
D *direction*	10	Si usa per automatizzare le operazioni relative alle stringhe. Se è a zero, indica che la scansione della memoria deve procedere incrementando gli indici; se invece è pari a uno, la scansione deve proseguire decrementando gli indici.
O *overflow*	11	È l'indicatore di traboccamento per le operazioni che riguardano valori con segno.

Indicatore (*flag*)	Bit	Descrizione
0	12	Riservato.
0	13	Riservato.
0	14	Riservato.
0	15	Riservato.

I registri che sono definiti «per usi generali», hanno comunque un ruolo predominante. Tra questi si includono anche ***SI*** e ***DI***:

Registro	Definizione	Scopo prevalente
AX	accumulatore	Usato soprattutto nei calcoli e per l'input e output. Nelle convenzioni di chiamata comuni, si usa ***AX*** per restituire un valore attraverso una funzione.
BX	base	Viene usato particolarmente come indice da sommare ad altri, per individuare una posizione in memoria.
CX	contatore	Usato come contatore nei cicli.
DX	dati	Si affianca a ***AX***, soprattutto nelle divisioni e moltiplicazioni.
SI	*source index*	Usato prevalentemente come indice dell'origine, nell'ambito di un segmento dati (***DS*** o ***ES***).
DI	*destination index*	Usato prevalentemente come indice della destinazione, nell'ambito di un segmento dati (***DS*** o ***ES***).

Trasferimento di dati tra due segmenti differenti

Il trasferimento di dati tra segmenti di memoria differenti richiede l'uso di istruzioni apposite, con cui il registro ***DS*** individua il segmento di origine e ***ES*** quello di destinazione. Viene mostrato un esempio, con una porzione di codice, che ha lo scopo di copiare un intero segmento, dall'indirizzo efficace 10000_{16}, a $1FFFF_{16}$ incluso, a partire dall'indirizzo 30000_{16}, fino a $3FFFF_{16}$. La notazione è quella «Intel».

```
cld             ; Azzera l'indicatore di direzione.
mov   ax, 3000h ; Assegna a ES il segmento di destinazione,
mov   es, ax    ; attraverso AX.
mov   ax, 1000h ; Assegna a DS il segmento di origine,
mov   ds, ax    ; attraverso AX.
mov   cx, 8000h ; Imposta il contatore a 32768.
mov   si, 0h    ; Indice iniziale nel segmento di origine.
mov   di, 0h    ; Indice iniziale nel segmento di
                ; destinazione.
rep             ; Ripete l'istruzione successiva finché
                ; CX != 0; riducendo CX di una unità a ogni
                ; ciclo.
movsw           ; Copia 16 bit da DS:SI a ES:DI,
                ; incrementando di due unità sia SI, sia DI
                ; (in base all'indicatore di direzione).
```

Va osservato che ***CX*** riceve inizialmente un valore pari a metà della dimensione di un segmento, perché la copia avviene a coppie di byte, ovvero a interi di 16 bit. Si può notare anche che i registri di segmento coinvolti ricevono il valore attraverso la mediazione di ***AX***, perché non gli si può assegnare direttamente un valore immediato.

Riferimenti a indirizzi di memoria con i registri

Per indicare un indirizzo di memoria, generalmente si può utilizzare una costante numerica pura e semplice, ovvero un valore immediato, ma spesso è possibile combinare il valore di uno o più registri. Nella notazione Intel, per specificare che il risultato di un'espressione rappresenta un indirizzo di memoria, la si racchiude tra parentesi quadre. Per esempio, '`-2[DX+SI]`' fa riferimento all'indirizzo di memoria efficace che si ottiene come ***DS***·16+***DX***+***SI***−2 (***DS*** partecipa in quanto si fa riferimento a un segmento e può trattarsi solo di quello dei dati). Le combinazioni ammissibili sono rappresentate dal modello seguente, tenendo conto che qui le parentesi quadre indicano un blocco opzionale:

[costante]+[BX|BP]+[SI|DI]

Lo specchietto successivo riepiloga tutte le combinazioni ammissibili, dove la sigla ***imm*** rappresenta un valore immediato (una costante numerica letterale) che può essere sia positivo, sia negativo:

Notazione	Indirizzo efficace corrispondente	Notazione	Indirizzo efficace corrispondente
`[SI]`	***DS***·16+***SI***	***imm***`[SI]`	***DS***·16+***SI***+***imm***
`[DI]`	***DS***·16+***DI***	***imm***`[DI]`	***DS***·16+***DI***+***imm***
`[BP]`	***DS***·16+***BP***	***imm***`[BP]`	***DS***·16+***BP***+***imm***
`[BX]`	***DS***·16+***BX***	***imm***`[BX]`	***DS***·16+***BX***+***imm***
`[BX+SI]`	***DS***·16+***BX***+***SI***	***imm***`[BX+SI]`	***DS***·16+***BX***+***SI***+***imm***
`[BX+DI]`	***DS***·16+***BX***+***DI***	***imm***`[BX+DI]`	***DS***·16+***BX***+***DI***+***imm***
`[BP+SI]`	***DS***·16+***BP***+***SI***	***imm***`[BP+SI]`	***DS***·16+***BP***+***SI***+***imm***
`[BP+DI]`	***DS***·16+***BP***+***DI***	***imm***`[BP+DI]`	***DS***·16+***BP***+***DI***+***imm***

Si osservi che la costante letterale che precede il gruppo tra parentesi quadre può essere sostituita da un nome simbolico, con il quale si indica una variabile in memoria (preferibilmente un array). In tal modo, la notazione richiama quella degli array, come si fa con il linguaggio C. Per esempio, '`x[SI]`', individua così il byte ***SI***-esimo a partire dall'indirizzo a cui si riferisce ***x***.

Convenzioni di chiamata

Le convenzioni di chiamata adottate per i microprocessori x86-16 sono le stesse di quelle usate per x86-32:

- si inseriscono nella pila dei dati gli argomenti della chiamata, in ordine inverso, in modo che l'ultimo inserimento sia quello del primo parametro della funzione;

  ```
  push …
  ```

- si esegue la chiamata;

  ```
  call …
  ```

- all'interno della funzione si salva il valore di ***BP*** nella pila, si assegna a ***BP*** l'indice attuale della pila (in modo da poter usare ***BP*** come riferimento per raggiungere nella pila gli argomenti della chiamata e le variabili locali) e si allocano nella stessa le variabili locali (variabili automatiche);

  ```
  enter …
  ```

- si salvano nella pila i registri che la funzione va a modificare, quindi si procede con il lavoro della funzione;

  ```
  pusha
  ```

- il primo argomento della chiamata si raggiunge con '`+4[BP]`', il secondo con '`+6[BP]`',... la prima variabile locale si raggiunge con '`-2[BP]`', la seconda con '`-4[BP]`',...

Al termine della funzione si fa in modo di ripristinare la situazione precedente alla chiamata, restituendo eventualmente un valore attraverso il registro ***AX*** o eventualmente la coppia ***DX***:***AX***;

- vengono ripristinati i registri salvati all'inizio della funzione;

  ```
  popa
  ```

- se la funzione deve restituire un valore viene, questo viene assegnato a ***AX***, oppure ***DX***:***AX*** (se questo valore è da 32 bit);

  ```
  mov ax, -m[bp]
  mov dx, -n[bp]
  ```

- viene ridotta la pila riportandone l'indice al valore di ***BP*** e recuperando il valore precedente di ***BP***;

  ```
  leave
  ```

• si ritorna all'indirizzo successivo alla chiamata;

```
ret
```

• si espellono gli argomenti della chiamata.

```
pop …
```

Sintesi delle istruzioni x86-16

Nelle tabelle successive vengono annotate le istruzioni che possono essere utilizzate con i microprocessori x86-16, raggruppate secondo il contesto a cui appartengono. Sono però escluse le istruzioni '**AA***x*' e '**DA***x*', relative alla gestione dei numeri in formato BCD (*Binary coded decimal*).

L'ordine in cui sono specificati gli operandi è quello «Intel», ovvero appare prima la destinazione e poi l'origine. Le sigle usate per definire i tipi di operandi sono: ***reg*** per «registro»; ***mem*** per «memoria»; ***imm*** per «immediato» (costante numerica).

Quando appare la pseudocodifica che deve spiegare l'effetto di un'istruzione, i riferimenti agli indirizzi in memoria vengono fatti in modo inusuale. Per esempio, ***(DS·16+SI)*** indica un indirizzo in memoria, individuato dal registro ***SI*** che si riferisce al segmento annotato in ***DS***. In modo analogo, ****(DS·16+SI)*** individua il contenuto della memoria al tale indirizzo, mentre ***&nome*** rappresenta l'indirizzo in memoria del simbolo ***nome***.

Nella colonna degli indicatori appare: il simbolo «#» per annotare che l'indicatore relativo può essere modificato dall'istruzione; il simbolo «t» per annotare che lo stato precedente dell'indicatore viene considerato dall'istruzione; zero o uno se l'indicatore viene impostato in un certo modo; il simbolo «?» se l'effetto dell'istruzione sull'indicatore è indefinito.

Tabella u139.16. Assegnamenti, scambi, conversioni e istruzione nulla.

Nome	Operandi: ***dst***, ***org1***, ***org2***	Descrizione	Indicatori
`NOP`		*not operate* Istruzione nulla.	`cpazstido` `.........`
`MOV`	***reg***, ***reg*** ***reg***, ***mem*** ***reg***, ***imm*** ***mem***, ***reg*** ***mem***, ***imm***	*move* Copia il valore dell'origine nella destinazione. Consente la copia da e verso i registri di segmento, ma per assegnare un valore a un registro di segmento occorre eseguire un passaggio intermedio attraverso un registro per usi generali. Origine e destinazione devono avere la stessa quantità di bit. ***dst*** `:=` ***org***	`cpazstido` `.........`
`LEA`	***reg***, ***mem***	*load effective address* Mette nel registro l'indirizzo della memoria, inteso come scostamento dall'inizio del segmento dati. ***dst*** `:=` ***&org***	`cpazstido` `.........`
`LDS`	***reg***, ***mem***	*load pointer using DS* Carica dalla memoria un valore a 32 bit, diviso in due blocchi da 16 bit, mettendo il primo blocco nel registro indicato e mettendo il secondo nel registro ***DS*** (segmento dati). ***DS*** `:` ***dst*** `:=` ***org***	`cpazstido` `.........`

Nome	Operandi: ***dst***, ***org1***, ***org2***	Descrizione	Indicatori
`LES`	***reg***, ***mem***	*load pointer using ES* Carica dalla memoria un valore a 32 bit, diviso in due blocchi da 16 bit, mettendo il primo blocco nel registro indicato e mettendo il secondo nel registro ***ES***. ***ES*** `:` ***dst*** `:=` ***org***	`cpazstido` `.........`
`XCHG`	***reg***, ***reg*** ***reg***, ***mem*** ***mem***, ***reg***	*exchange data* Scambia i valori. ***dst*** `:==:` ***org***	`cpazstido` `.........`
`CBW`		*convert byte to word* Converte un intero con segno, della dimensione di 8 bit, contenuto in ***AL***, in modo da occupare tutto ***AX*** (da 8 bit a 16 bit). L'espansione tiene conto del segno. ***AX*** `:=` ***AL***	`cpazstido` `.........`

`CWD`		*convert word to double word*
		Converte un intero con segno, della dimensione di 16 bit, contenuto in ***AX***, in modo da estendersi anche in ***DX***, tenendo conto del segno.
		`IF` ***AX*** `>= 0`
		`THEN`
		`  ` ***DX*** `:= 0`
		`ELSE`
		`  ` ***DX*** `:=` $\texttt{FFFF}_{16}$
			`cpazstido`
			`.........`

`LAHF`		*load flags into AH* Carica i primi otto indicatori in ***AH***, escludendo quelli riservati. AH_{bit0}`:=`***c*** AH_{bit1}`:=1` AH_{bit2}`:=`***p*** AH_{bit3}`:=0` AH_{bit4}`:=`***a*** AH_{bit5}`:=0` AH_{bit6}`:=`***z*** AH_{bit7}`:=`***s***	`cpazstido` `#########`
`SAHF`		*store AH into flags* Modifica il valore dei primi otto indicatori, esclusi i bit 1, 3 e 5 (il secondo, il quarto e il sesto, che sono riservati e a loro non si attribuisce un significato particolare), scrivendoci sopra il contenuto di ***AH***. ***c*** `:=`AH_{bit0} ***p*** `:=`AH_{bit2} ***a*** `:=`AH_{bit4} ***z*** `:=`AH_{bit6} ***s*** `:=`AH_{bit7}	`cpazstido` `#########`

Tabella u139.17. Movimento di dati.

Nome	Operandi: ***dst***, ***org1***, ***org2***	Descrizione	Indicatori

LODSB		*load string byte*
		Dall'indirizzo a cui punta la coppia ***DS:SI*** (***DS·16+SI***), viene letto un byte e copiato in ***AL***. Se l'indicatore di direzione è pari a zero, ***SI*** viene incrementato di una unità, altrimenti viene decrementato di una unità.
		AL `:=` ****(DS·16+SI)***
		`IF` ***d***`==0`
		`THEN`
		`    `***SI***`++`
		`ELSE`
		`    `***SI***`--`
			`cpazstido`
			`.......t.`
			`.........`

LODSW		*load string word*
		Dall'indirizzo a cui punta la coppia ***DS:SI*** (***DS·16+SI***), viene letto un blocco da 16 bit e copiato in ***AX***. Se l'indicatore di direzione è pari a zero, ***SI*** viene incrementato di due unità, altrimenti viene decrementato di due unità.
		AL `:=` ****(DS·16+SI)***
		`IF` ***d***`==0`
		`THEN`
		`    `***SI***`:+=2`
		`ELSE`
		`    `***SI***`:-=2`
			`cpazstido`
			`.......t.`
			`.........`

STOSB		*store string byte*
		All'indirizzo a cui punta la coppia ***ES:DI*** (***ES·16+DI***), viene scritto il valore contenuto in ***AL***, aggiornando ***DI*** in base al contenuto dell'indicatore di direzione.
		****(ES·16+DI)*** `:=` ***AL***
		`IF` ***d***`==0`
		`THEN`
		`    `***DI***`++`
		`ELSE`
		`    `***DI***`--`
			`cpazstido`
			`.......t.`
			`.........`

STOSW		*store string word*
		All'indirizzo a cui punta la coppia ***ES:DI*** (***ES·16+DI***), viene scritto il valore contenuto in ***AX***, aggiornando ***DI*** in base al contenuto dell'indicatore di direzione.
		****(ES·16+DI)*** `:=` ***AX***
		`IF` ***d***`==0`
		`THEN`
		`    `***DI***`:+=2`
		`ELSE`
		`    `***DI***`:-=2`
			`cpazstido`
			`.......t.`
			`.........`

Nome	Operandi: ***dst***, ***org1***, ***org2***	Descrizione	Indicatori

MOVSB		*move string byte*
		Copia un byte, dall'indirizzo a cui punta la coppia ***DS:SI*** (***DS·16+SI***), all'indirizzo a cui punta la coppia ***ES:DI*** (***ES·16+DI***), aggiornando ***SI*** e ***DI*** in base al valore dell'indicatore di direzione.
		****(ES·16+DI)*** `:=` ****(DS·16+SI)***
		`IF` ***d***`==0`
		`THEN`
		`    `***SI***`++`
		`    `***DI***`++`
		`ELSE`
		`    `***SI***`--`
		`    `***DI***`--`
			`cpazstido`
			`.......t.`
			`.........`

MOVSW		*move string word*
		Copia un blocco di 16 bit, dall'indirizzo a cui punta la coppia ***DS:SI*** (***DS·16+SI***), all'indirizzo a cui punta la coppia ***ES:DI*** (***ES·16+DI***), aggiornando ***SI*** e ***DI*** in base al valore dell'indicatore di direzione.
		****(ES·16+DI)*** `:=` ****(DS·16+SI)***
		`IF` ***d***`==0`
		`THEN`
		`    `***SI***`:+=2`
		`    `***DI***`:+=2`
		`ELSE`
		`    `***SI***`:-=2`
		`    `***DI***`:-=2`
			`cpazstido`
			`.......t.`
			`.........`

REP		*repeat*
		Ripete l'istruzione successiva (che può essere una tra: '**`LODSB`**', '**`LODSW`**', '**`STOSB`**', '**`STOSW`**', '**`MOVSB`**', '**`MOVSW`**'), per ***CX*** volte.
		`IF` ***CX***`!=0`
		`THEN`
		`    istruzione successiva`
		`    `***CX***`--`
		`ELSE`
		`    break`
			`cpazstido`
			`.........`

XLATB		*translate table to byte* Assegna a ***AL*** il valore che si può raggiungere all'indirizzo composto da ***DS***:***BX***+***AL*** (***DS·16+BX+AL***), dove ***AL*** va inteso come valore senza segno. ***AL*** `:=` ****(DS·16+BX+AL)***	`cpazstido` `.........`

Tabella u139.18. Confronti con la memoria.

Nome	Operandi: ***dst***, ***org1***, ***org2***	Descrizione	Indicatori

`SCASB`		*compare string byte*
		Confronta il contenuto di ***AL*** con il valore a cui punta la coppia ***ES:DI*** (***ES·16+DI***), aggiornando di conseguenza gli indicatori e anche il registro ***DI*** in base all'indicatore di direzione.
		****(ES·16+DI) - AL***
		`IF d==0`
		`THEN`
		`  DI++`
		`ELSE`
		`  DI--`
			`cpazstido`
			`.......t.`
			`#####···#`

`SCASW`		*compare string word*
		Confronta il contenuto di ***AX*** con il valore a cui punta la coppia ***ES:DI*** (***ES·16+DI***), aggiornando di conseguenza gli indicatori e anche il registro ***DI*** in base all'indicatore di direzione.
		****(ES·16+DI) - AX***
		`IF d==0`
		`THEN`
		`  DI:+=2`
		`ELSE`
		`  DI:-=2`
			`cpazstido`
			`.......t.`
			`#####···#`

`CMPSB`		*compare string byte in memory*
		Confronta il byte a cui punta la coppia ***ES:DI*** (***ES·16+DI***), con quello a cui punta la coppia ***DS:SI*** (***DS·16+SI***), aggiornando di conseguenza gli indicatori e anche i registri ***DI*** e ***SI*** in base all'indicatore di direzione.
		****(DS·16+SI) - *(ES·16+DI)***
		`IF d==0`
		`THEN`
		`  SI++`
		`  DI++`
		`ELSE`
		`  SI++`
		`  DI--`
			`cpazstido`
			`.......t.`
			`#####···#`

Nome	Operandi: ***dst***, ***org1***, ***org2***	Descrizione	Indicatori

`CMPSW`		*compare string word in memory*
		Confronta il blocco da 16 bit a cui punta la coppia ***ES:DI*** (***ES·16+DI***), con quello a cui punta la coppia ***DS:SI*** (***DS·16+SI***), aggiornando di conseguenza gli indicatori e anche i registri ***DI*** e ***SI*** in base all'indicatore di direzione.
		****(DS·16+SI) - *(ES·16+DI)***
		`IF d==0`
		`THEN`
		`  SI++`
		`  DI++`
		`ELSE`
		`  SI++`
		`  DI--`
			`cpazstido`
			`.......t.`
			`#####···#`

`REPE`
`REPZ`
		repeat while equal
		repeat while zero
		Ripete l'istruzione successiva (che può essere una tra: '**`SCASB`**', '**`SCASW`**', '**`CMPSB`**', '**`CMPSW`**'), fino a che l'indicatore ***z*** è pari a uno (rappresentante l'uguaglianza di una comparazione, ovvero che la sottrazione dà zero), fino a un massimo di ***CX*** volte.
		`IF CX!=0`
		`THEN`
		`  istruzione successiva`
		`  CX--`
		`  IF z==1`
		`  THEN`
		`  continue`
		`  ELSE`
		`  break`
		`ELSE`
		`  break`
			`cpazstido`
			`...#.....`

Nome	Operandi: ***dst***, ***org1***, ***org2***	Descrizione	Indicatori

`REPNE`
`REPNZ`
		repeat while not equal
		repeat while not zero
		Ripete l'istruzione successiva (che può essere una tra: '**`SCASB`**', '**`SCASW`**', '**`CMPSB`**', '**`CMPSW`**'), fino a che l'indicatore ***z*** è pari a zero (rappresentante la disuguaglianza della comparazione, ovvero che la sottrazione non dà zero), fino a un massimo di ***CX*** volte.
		`IF CX!=0`
		`THEN`
		`  istruzione successiva`
		`  CX--`
		`  IF z==0`
		`  THEN`
		`  continue`
		`  ELSE`
		`  break`
		`ELSE`
		`  break`
			`cpazstido`
			`...#.....`

Tabella u139.19. Operazioni aritmetiche.

Nome	Operandi: ***dst***, ***org1***, ***org2***	Descrizione	Indicatori
`NEG`	***reg*** ***mem***	*negation* Inverte il segno di un numero, attraverso il complemento a due. ***operand*** `:=` ***−operand***	`cpazstido` `##·##···#`
`ADD`	***reg***, ***reg*** ***reg***, ***mem*** ***reg***, ***imm*** ***mem***, ***reg*** ***mem***, ***imm***	*addition* Somma di interi, con o senza segno, ignorando il riporto precedente. Se i valori si intendono con segno, è importante l'esito dell'indicatore di traboccamento (*overflow*), se invece i valori sono da intendersi senza segno, è importante l'esito dell'indicatore di riporto (*carry*). ***dst*** `:=` ***org*** `+` ***dst***	`cpazstido` `##·##···#`
`SUB`	***reg***, ***reg*** ***reg***, ***mem*** ***reg***, ***imm*** ***mem***, ***reg*** ***mem***, ***imm***	*subtraction* Sottrazione di interi con o senza segno, ignorando il riporto precedente. ***dst*** `:=` ***org*** `−` ***dst***	`cpazstido` `##·##···#`

`ADC`	***reg***, ***reg***
	reg, ***mem***
	reg, ***imm***
	mem, ***reg***
	mem, ***imm***
		addition with carry
		Somma di interi, con o senza segno, aggiungendo anche il riporto precedente (l'indicatore *carry*).
		dst `:=` ***org*** `+` ***dst*** `+` ***c***
			`cpazstido`
			`t·····`
			`##·##···#`

`SBB`	***reg***, ***reg***
	reg, ***mem***
	reg, ***imm***
	mem, ***reg***
	mem, ***imm***
		subtraction with borrow
		Sottrazione di interi, con o senza segno, tenendo conto del «prestito» precedente (l'indicatore *carry*).
		dst `:=` ***org*** `+` ***dst*** `−` ***c***
			`cpazstido`
			`t·····`
			`##·##···#`

Nome	Operandi: ***dst***, ***org1***, ***org2***	Descrizione	Indicatori
`INC`	***reg*** ***mem***	*increment* Incrementa di una unità un intero. ***operand***`++`	`cpazstido` `·#·##···#`
`DEC`	***reg*** ***mem***	*decrement* Decrementa di una unità un valore intero. ***operand***`−−`	`cpazstido` `·#·##···#`
`MUL`	***reg*** ***mem***	*multiply* Moltiplicazione intera senza segno. L'operando è il moltiplicatore, mentre il moltiplicando è costituito da registri prestabiliti. ***AX*** `:=` ***AL*** `*` ***operand*** ***DX:AX*** `:=` ***AX*** `*` ***operand***	`cpazstido` `#?·??···#`
`DIV`	***reg*** ***mem***	*division* Divisione intera senza segno. L'operando è il divisore, mentre il dividendo è costituito da registri prestabiliti. ***AL*** `:=` ***AX*** `/` ***operand*** ***AH*** `:=` ***AX%operand*** ***AX*** `:=` ***(DX:AX)*** `/` ***operand*** ***DX*** `:=` ***(DX:AX)%operand***	`cpazstido` `??·??···?`
`IMUL`	***reg*** ***mem***	*signed multiply* Moltiplicazione intera con segno. In questo caso l'operando è il moltiplicatore, mentre il moltiplicando è costituito da registri prestabiliti. ***AX*** `:=` ***AL*** `*` ***operand*** ***(DX:AX)*** `:=` ***AX*** `*` ***operand***	`cpazstido` `#?·??···#`
`IDIV`	***reg*** ***mem***	*signed division* Divisione intera con segno. L'operando è il divisore, mentre il dividendo è costituito da registri prestabiliti. ***AL*** `:=` ***AX*** `/` ***operand*** ***AH*** `:=` ***AX%operand*** ***AX*** `:=` ***(DX:AX)*** `/` ***operand*** ***DX*** `:=` ***(DX:AX)%operand***	`cpazstido` `??·??···?`

Tabella u139.20. Operazioni logiche.

Nome	Operandi: ***dst***, ***org1***, ***org2***	Descrizione	Indicatori
`NOT`	***reg*** ***mem***	NOT di tutti i bit dell'operando. ***dst*** `:= NOT` ***dst***	`cpazstido` `·····`

`AND`
`OR`
`XOR`
	reg, ***reg***
	reg, ***mem***
	reg, ***imm***
	mem, ***reg***
	mem, ***imm***
		AND, OR, o XOR, tra tutti i bit dei due operandi.
		dst `:=` ***org*** `AND` ***dst***
		dst `:=` ***org*** `OR` ***dst***
		dst `:=` ***org*** `XOR` ***dst***
			`cpazstido`
			`0#·##···0`

Tabella u139.21. Scorrimenti e rotazioni.

Nome	Operandi: ***dst***, ***org1***, ***org2***	Descrizione	Indicatori
SHL SHR	***reg***, 1 ***mem***, 1 ***reg*** ***mem***	*shift left* *shift right* Fa scorrere i bit, rispettivamente verso sinistra o verso destra (l'ultima cifra perduta finisce nell'indicatore del riporto). Se appare un solo operando, la rotazione viene eseguita ***CL*** volte. Se il valore immediato è maggiore di uno, è il compilatore che ripete l'istruzione per più volte.	cpazstido #·····#
SAL SAR	***reg***, 1 ***mem***, 1 ***reg*** ***mem***	*shift arithmetically left* *shift arithmetically right* Fa scorrere i bit, rispettivamente verso sinistra o verso destra (l'ultima cifra perduta finisce nell'indicatore del riporto), mantenendo il segno originale (logicamente '**SAL**' è identico a '**SHL**'). Se appare un solo operando, la rotazione viene eseguita ***CL*** volte. Se il valore immediato è maggiore di uno, è il compilatore che ripete l'istruzione per più volte.	cpazstido #·····
RCL RCR	***reg***, 1 ***mem***, 1 ***reg*** ***mem***	*rotate left with carry* *rotate right with carry* Ruota i bit, rispettivamente verso sinistra o verso destra, utilizzando anche l'indicatore di riporto (*carry*). Se appare un solo operando, la rotazione viene eseguita ***CL*** volte. Se il valore immediato è maggiore di uno, è il compilatore che ripete l'istruzione per più volte.	cpazstido t····· #·····#
ROL ROR	***reg***, 1 ***mem***, 1 ***reg*** ***mem***	*rotate left* *rotate right* Ruota i bit, rispettivamente verso sinistra o verso destra. Se appare un solo operando, la rotazione viene eseguita ***CL*** volte. Se il valore immediato è maggiore di uno, è il compilatore che ripete l'istruzione per più volte.	cpazstido #·····#

Tabella u139.22. Chiamate e gestione della pila.

Nome	Operandi: ***dst***, ***org1***, ***org2***	Descrizione	Indicatori
CALL	***reg*** ***mem*** ***imm***	Inserisce nella pila l'indirizzo dell'istruzione successiva e salta all'indirizzo indicato, che si riferisce allo scostamento a partire dall'inizio del segmento codice (***CS***). Pertanto, l'indirizzo a cui ci si riferisce è a 16 bit. push ***indirizzo_successivo*** ***IP*** := ***operand***	cpazstido ·····

Nome	Operandi: ***dst***, ***org1***, ***org2***	Descrizione	Indicatori
CALL FAR	***imm***:***imm***	*call procedure* Inserisce nella pila il valore di ***CS*** e poi l'indirizzo dell'istruzione successiva (***IP*** dell'istruzione successiva) e salta all'indirizzo indicato. L'indirizzo deve essere di quattro byte (32 bit), in quanto deve specificare anche il segmento codice da raggiungere. push ***CS*** push ***indirizzo_successivo*** ***CS***:***IP*** := ***operand***	cpazstido ·····
RET		*return from call* Estrae dalla pila l'indirizzo dell'istruzione da raggiungere (***IP***) e salta a quella (serve a concludere una chiamata eseguita con '**CALL**'). pop ***IP***	cpazstido ·····
RETF RET FAR		*return from far call* Estrae dalla pila il valore di ***CS*** e quindi l'indirizzo dell'istruzione da raggiungere (***IP***) e salta a quella (serve a concludere una chiamata eseguita con '**CALL FAR**'). pop ***IP*** pop ***CS***	cpazstido ·····
PUSH	***reg*** ***mem***	*push data onto stack* Inserisce nella pila il valore (della dimensione di un registro comune). ***SP***:-=2 *(***SS***·16+***SP***):=***operand***	cpazstido ·····
POP	***reg*** ***mem***	*pop data from stack* Estrae dalla pila l'ultimo valore inserito (della dimensione di un registro comune). ***operand***:=*(***SS***·16+***SP***) ***SP***:+=2	cpazstido ·····
PUSHF		*push flags onto stack* Inserisce nella pila l'insieme del registro degli indicatori (***FLAGS***). push ***FLAGS***	cpazstido ·····
POPF		*pop flags from stack* Estrae dalla pila l'insieme del registro degli indicatori (***FLAGS***), aggiornando di conseguenza il registro stesso. pop ***FLAGS***	cpazstido ?????????

Nome	Operandi: ***dst***, ***org1***, ***org2***	Descrizione	Indicatori
ENTER	***imm8***, 0	*enter stack frame* Questa funzione esiste a partire dai microprocessori i186. Inserisce nella pila il valore di ***BP***, poi assegna a ***BP*** il valore di ***SP*** e infine decrementa ***SP*** del valore fornito come immediato. Serve a predisporre ***BP*** e ***SP*** all'inizio di una funzione, specificando lo spazio necessario per le variabili locali nella pila. `push` ***BP*** ***BP***`:=`***SP*** ***SP***`:-=2*`***dst***	`cpazstido` `.........`
PUSHA		*push all registers onto stack* Questa funzione esiste a partire dai microprocessori i186. Inserisce nella pila i registri principali: ***AX***, ***CX***, ***DX***, ***BX***, ***SP***, ***BP***, ***SI***, ***DI***. `push` ***AX*** `push` ***CX*** `push` ***DX*** `push` ***BX*** `push` ***SP*** `push` ***BP*** `push` ***SI*** `push` ***DI***	`cpazstido` `.........`
POPA		*pop all registers from stack* Questa funzione esiste a partire dai microprocessori i186. Ripristina i registri principali, estraendo i contenuti dalla pila: ***DI***, ***SI***, ***BP***, ***SP*** viene eliminato senza aggiornare il registro, ***BX***, ***DX***, ***CX***, ***AX***. Come si vede, anche se '**`PUSHA`**' salva l'indice della pila, in pratica questo indice non viene ripristinato. `pop` ***DI*** `pop` ***SI*** `pop` ***BP*** ***SP***`:+=2` `pop` ***BX*** `pop` ***DX*** `pop` ***CX*** `pop` ***AX***	`cpazstido` `.........`
LEAVE		*leave stack frame* Questa funzione esiste a partire dai microprocessori i186. Ripristina i valori di ***BP*** e di ***SP***, allo stato che avevano prima dell'uso dell'istruzione '**`ENTER`**'. ***SP***`:=`***BP*** `pop` ***BP***	`cpazstido` `.........`

Tabella u139.23. Interruzioni.

Nome	Operandi: ***dst***, ***org1***, ***org2***	Descrizione	Indicatori

INT	***imm8***	*call to interrupt*
		Esegue una chiamata attraverso un'interruzione. Prima di saltare alla codice relativo all'interruzione selezionata, inserisce nella pila ***FLAGS***, ***CS*** e ***IP***. Azzera anche l'indicatore ***IF*** (*interrupt flag*), mentre gli altri indicatori rimangono inalterati.
		`pusf`
		`push` ***CS***
		`push` ***IP***
		i`:=0`
		`jmp far 0:(`***operand***`·4)`
			`cpazstido`
			`.........`

IRET		*return from interrupt* Conclude l'esecuzione del codice relativo a un'interruzione recuperando dalla pila i valori inseriti alla chiamata con '**`INT`**': ***IP***, ***CS*** e ***FLAGS***. Pertanto, il valore degli indicatori viene ripristinato allo stato precedente alla chiamata. `pop` ***IP*** `pop` ***CS*** `popf`	`cpazstido` `##·##···#`
CLI		*clear interrupt flag* Azzera l'indicatore di abilitazione delle interruzioni (*interrupt flag*), disabilitando di conseguenza le interruzioni hardware. ***i***`:=0`	`cpazstido` `......0..`
STI		*set interrupt flag* Attiva l'indicatore di abilitazione delle interruzioni (*interrupt flag*), abilitando di conseguenza le interruzioni hardware. ***i*** `:= 1`	`cpazstido` `......1..`
HLT		*enter halt state* Ferma il sistema, fino a quando viene ricevuta un'interruzione hardware.	`cpazstido` `.........`

INTO		*interrupt if overflow*
		Se l'indicatore di straripamento è attivo, esegue la chiamata dell'interruzione numero 4 (la quale dovrebbe gestore il problema).
			`cpazstido`
			`........t`
			`.........`

Tabella u139.24. Indicatori e confronti tra registri.

Nome	Operandi: ***dst***, ***org1***, ***org2***	Descrizione	Indicatori
CLC		*clear carry flag* Azzera l'indicatore del riporto (*carry*), senza intervenire negli altri indicatori. ***c***`:=0`	`cpazstido` `0.......`

Nome	Operandi: ***dst***, ***org1***, ***org2***	Descrizione	Indicatori
`CLD`		*clear direction flag* Azzera l'indicatore di direzione (*direction*), senza intervenire negli altri indicatori. ***d***`:=0`	`cpazstido` `·····0·`
`STC`		*set carry flag* Attiva l'indicatore di riporto (*carry*). ***c***`:=1`	`cpazstido` `1·····`
`STD`		*set direction flag* Attiva l'indicatore di direzione (*direction*). ***d***`:=1`	`cpazstido` `·····1·`
`CMC`		*complement carry flag* Inverte il valore dell'indicatore del riporto (*carry*).	`cpazstido` `#·····`
`CMP`	***reg***, ***reg*** ***reg***, ***mem*** ***reg***, ***imm*** ***mem***, ***reg*** ***mem***, ***imm***	*compare operands* Confronta due valori interi. La comparazione avviene simulando la sottrazione dell'origine dalla destinazione, senza però modificare gli operandi, ma aggiornando gli indicatori, come se fosse avvenuta una sottrazione vera e propria. ***dst*** `-` ***org***	`cpazstido` `##·##···#`
`TEST`	***reg***, ***reg*** ***reg***, ***imm*** ***mem***, ***reg*** ***mem***, ***imm***	*logical compare* AND dei due valori senza conservare il risultato. Serve solo a ottenere l'aggiornamento degli indicatori. ***dst*** `AND` ***org***	`cpazstido` `0#·##···0`

Tabella u139.25. Salti.

Nome	Operandi: ***dst***, ***org1***, ***org2***	Descrizione	Indicatori
`JMP`	***reg*** ***mem*** ***imm***	*jump* Salto incondizionato all'indirizzo indicato, che si intende relativo al segmento codice (***CS***). ***IP*****`:=`*****operand***	`cpazstido` `·····`
`JMP FAR`	***imm***`:`***imm***	*far jump* Salto incondizionato all'indirizzo indicato, costituito sia dal segmento codice, sia dall'indirizzo relativo, all'interno di questo. ***CS*****`:`*****IP*****`:=`*****operand***	`cpazstido` `·····`
`JA` `JNBE`	***imm***	*conditional jump* Dopo un confronto di valori senza segno, salta se la destinazione era maggiore dell'origine. Il salto riguarda solo l'ambito del segmento codice attuale. `CMP` ***dst***`,` ***org*** `IF` ***dst*** `>` ***org*** `THEN` `go to` ***imm***	`cpazstido` `t··t····`

Nome	Operandi: ***dst***, ***org1***, ***org2***	Descrizione	Indicatori
`JAE` `JNB`	***imm***	*conditional jump* Dopo un confronto di valori senza segno, salta se la destinazione era maggiore o uguale all'origine. Il salto riguarda solo l'ambito del segmento codice attuale. `CMP` ***dst***`,` ***org*** `IF` ***dst*** `>=` ***org*** `THEN` `go to` ***imm***	`cpazstido` `t··t····`
`JB` `JNAE`	***imm***	*conditional jump* Dopo un confronto di valori senza segno, salta se la destinazione era minore dell'origine. Il salto riguarda solo l'ambito del segmento codice attuale. `CMP` ***dst***`,` ***org*** `IF` ***dst*** `<` ***org*** `THEN` `go to` ***imm***	`cpazstido` `t··t····`
`JBE` `JNA`	***imm***	*conditional jump* Dopo un confronto di valori senza segno, salta se la destinazione era minore o uguale all'origine. Il salto riguarda solo l'ambito del segmento codice attuale. `CMP` ***dst***`,` ***org*** `IF` ***dst*** `<=` ***org*** `THEN` `go to` ***imm***	`cpazstido` `t··t····`
`JE`	***imm***	*conditional jump* Dopo un confronto, indipendentemente dal segno, salta se la destinazione era uguale all'origine. Il salto riguarda solo l'ambito del segmento codice attuale. `CMP` ***dst***`,` ***org*** `IF` ***dst*** `==` ***org*** `THEN` `go to` ***imm***	`cpazstido` `···t····`
`JNE`	***imm***	*conditional jump* Dopo un confronto, indipendentemente dal segno, salta se la destinazione era diversa dall'origine. Il salto riguarda solo l'ambito del segmento codice attuale. `CMP` ***dst***`,` ***org*** `IF` ***dst*** `!=` ***org*** `THEN` `go to` ***imm***	`cpazstido` `···t····`
`JG` `JNLE`	***imm***	*conditional jump* Dopo un confronto con segno, salta se la destinazione era maggiore dell'origine. Il salto riguarda solo l'ambito del segmento codice attuale. `CMP` ***dst***`,` ***org*** `IF` ***dst*** `>` ***org*** `THEN` `go to` ***imm***	`cpazstido` `···tt···t`

Nome	Operandi: ***dst***, ***org1***, ***org2***	Descrizione	Indicatori
JGE JNL	***imm***	*conditional jump* Dopo un confronto con segno, salta se la destinazione era maggiore o uguale all'origine. Il salto riguarda solo l'ambito del segmento codice attuale. `CMP` ***dst***`,` ***org*** `IF` ***dst*** `>=` ***org*** `THEN` `go to` ***imm***	`cpazstido` `···tt···t`
JL JNGE	***imm***	*conditional jump* Dopo un confronto con segno, salta se la destinazione era minore dell'origine. Il salto riguarda solo l'ambito del segmento codice attuale. `CMP` ***dst***`,` ***org*** `IF` ***dst*** `<` ***org*** `THEN` `go to` ***imm***	`cpazstido` `···tt···t`
JLE JNG	***imm***	*conditional jump* Dopo un confronto con segno, salta se la destinazione era minore o uguale all'origine. Il salto riguarda solo l'ambito del segmento codice attuale. `CMP` ***dst***`,` ***org*** `IF` ***dst*** `<=` ***org*** `THEN` `go to` ***imm***	`cpazstido` `···tt···t`
JC JNC	***imm***	*conditional jump* Salta se l'indicatore del riporto (*carry*), rispettivamente, è attivo, oppure non è attivo. Il salto riguarda solo l'ambito del segmento codice attuale.	`cpazstido` `t·····`
JO JNO	***imm***	*conditional jump* Salta se l'indicatore di traboccamento (*overflow*), rispettivamente, è attivo, oppure non è attivo.	`cpazstido` `·····t`
JS JNS	***imm***	*conditional jump* Salta se l'indicatore di segno (*sign*), rispettivamente, è attivo, oppure non è attivo.	`cpazstido` `····t···`
JZ JNZ	***imm***	*conditional jump* Salta se l'indicatore di zero, rispettivamente, è attivo, oppure non è attivo.	`cpazstido` `···t····`
JP JPE	***imm***	*conditional jump* Salta se l'indicatore di parità è attivo.	`cpazstido` `·t·····`
JNP JPO	***imm***	*conditional jump* Salta se l'indicatore di parità non è attivo.	`cpazstido` `·t·····`
JCXZ	***imm***	*conditional jump* Salta se il valore contenuto nel registro ***CX*** è pari a zero.	`cpazstido` `·····`

Tabella u139.26. Iterazioni

Nome	Operandi: ***dst***, ***org1***, ***org2***	Descrizione	Indicatori
LOOP	***imm8***	*loop* Senza alterare gli indicatori, decrementa di una unità il registro '**CX**', quindi, se il registro è ancora diverso da zero, salta all'indirizzo cui fa riferimento l'operando. Tale indirizzo non può essere molto lontano dalla posizione corrente.	`cpazstido` `·····`
LOOPE LOOPZ	***imm8***	*conditional loop* Senza alterare gli indicatori, decrementa di una unità il registro '**CX**', quindi, se il registro è ancora diverso da zero e l'indicatore «zero» è attivo, salta all'indirizzo cui fa riferimento l'operando. Tale indirizzo non può essere molto lontano dalla posizione corrente.	`cpazstido` `···t·····`
LOOPNE LOOPNZ	***imm8***	*conditional loop* Senza alterare gli indicatori, decrementa di una unità il registro '**CX**', quindi, se il registro è ancora diverso da zero e l'indicatore «zero» non è attivo, salta all'indirizzo cui fa riferimento l'operando. Tale indirizzo non può essere molto lontano dalla posizione corrente.	`cpazstido` `···t·····`

Tabella u139.27. Input e output.

Nome	Operandi: ***dst***, ***org1***, ***org2***	Descrizione	Indicatori
IN	***AL***, ***imm8*** ***AX***, ***imm8***	*input* Assegna a ***AL*** o ***AX*** il valore letto dalla porta specificata; in tal caso il numero di porta non può essere superiore a 255.	`cpazstido` `·····`
IN	***AL***, ***DX*** ***AX***, ***DX***	*input* Assegna a ***AL*** o ***AX*** il valore letto dalla porta specificata da ***DX***; in tal caso il numero di porta può essere superiore a 255.	`cpazstido` `·····`
OUT	***imm8***, ***AL*** ***imm8***, ***AX***	*output* Scrive nella porta specificata il valore contenuto in ***AL*** o ***AX***; in tal caso il numero di porta non può essere superiore a 255.	`cpazstido` `·····`
OUT	***DX***, ***AL*** ***DX***, ***AX***	*output* Scrive nella porta indicata da ***DX*** il valore contenuto in ***AL*** o ***AX***; in tal caso il numero di porta può essere superiore a 255.	`cpazstido` `·····`

Sostituzione delle istruzioni per i186

Nella sezione precedente sono state menzionate delle istruzioni che non fanno parte dei microprocessori 8086/8088, ma queste possono essere ottenute facilmente attraverso altre istruzioni elementari,

tanto che l'assemblatore potrebbe provvedervi direttamente. A ogni modo viene annotato qui come possono essere sostituite.

Listato u139.28. Sostituzione per l'istruzione '**PUSHA**'.

```
push ax
push cx
push dx
push bx
push sp
push bp
push si
push di
```

Listato u139.29. Sostituzione per l'istruzione '**POPA**'. Il registro ***SP*** non viene ripristinato, di conseguenza si riduce l'indice della pila (si incrementa ***SP***) senza estrarne il valore.

```
pop di
pop si
pop bp
add sp, 2       ; non ripristina SP
pop bx
pop dx
pop cx
pop ax
```

Listato u139.30. Sostituzione per l'istruzione '**ENTER**'. La riduzione di ***SP*** dipende dalla quantità di variabili locali che si vogliono gestire. Usando interi da 16 bit, si tratta di moltiplicare la quantità di variabili locali per due. Va ricordato che il segmento a cui si riferisce ***BP*** è ***DS***, per cui è indispensabile che ***DS*** sia uguale a ***SS***, essendo usato in questo modo come riferimento alla pila.

```
push bp
mov  bp, sp
sub  sp, 2      ; 0, 2, 4, 6,...
```

Listato u139.31. Sostituzione per l'istruzione '**LEAVE**'.

```
mov sp, bp
pop bp
```

Riferimenti

«

- Andrew S. Tanenbaum, *Operating Systems: Design and Implementation*, **prima edizione**, 1987, Prentice-Hall, ISBN 0-13-637406-9

  Appendice B: *introduction to the IBM PC*
- MAD, *Assembly tutorial*
  http://www.xs4all.nl/~smit/asm01001.htm
- Wikipedia, *x86 instruction listings*
  http://en.wikipedia.org/wiki/X86_instruction_listings
- *The x86 Interrupt List, aka "Ralf Brown's Interrupt List", "RBIL"*
  http://www.cs.cmu.edu/~ralf/files.html
- *Computer interrupt*
  http://wayback.archive.org/web/20040101000000/http://calab.kaist.ac.kr/~hyoon/courses/cs310_2001fa01ll/micro17.ppt*
  http://www.ece.msstate.edu/~reese/EE3724/lectures/interrupt/interrupt.pdf
- BiosCentral, *BIOS data area*
  http://www.bioscentral.com/misc/bda.htm
- Robert de Bath, *Linux 8086 development environment*
  http://homepage.ntlworld.com/robert.debath/
  http://homepage.ntlworld.com/robert.debath/dev86/