

# Programmare a 16 bit



Microprocessori x86-16 .....	2939
Segmenti .....	2941
Registri .....	2943
Trasferimento di dati tra due segmenti differenti .....	2947
Riferimenti a indirizzi di memoria con i registri .....	2948
Convenzioni di chiamata .....	2950
Sintesi delle istruzioni x86-16 .....	2951
Sostituzione delle istruzioni per i186 .....	2998
Riferimenti .....	3000
Architettura IBM PC .....	3003
IVT: «interrupt vector table» .....	3004
BIOS data area .....	3007
Altre aree di memoria .....	3008
Interruzioni principali .....	3008
Riferimenti .....	3014
Strumenti di sviluppo e di utilizzo .....	3017
Preparazione .....	3017
Bcc .....	3019
As86 .....	3021
Ld86 .....	3022
Bootblocks .....	3025
Riferimenti .....	3027



# Microprocessori x86-16



Segmenti .....	2941
Registri .....	2943
Trasferimento di dati tra due segmenti differenti .....	2947
Riferimenti a indirizzi di memoria con i registri .....	2948
Convenzioni di chiamata .....	2950
Sintesi delle istruzioni x86-16 .....	2951
Sostituzione delle istruzioni per i186 .....	2998
Riferimenti .....	3000

ADC 2970 ADD 2970 AH 2943 AL 2943 AND 2973 AX 2943 BH  
2943 BL 2943 BP 2943 BX 2943 CALL 2977 CALL FAR 2977  
CBW 2952 CH 2943 CL 2943 CLC 2986 CLD 2986 CLI 2983 CMC  
2986 CMP 2986 CMPSB 2964 CMPSW 2964 CWD 2952 CX 2943  
DEC 2970 DH 2943 DI 2943 DIV 2970 DL 2943 DX 2943 ENTER  
2977 FLAGS 2943 HLT 2983 IDIV 2970 IMUL 2970 IN 2997  
2997 INC 2970 INT 2983 INTO 2983 IP 2943 IRET 2983 JA  
2988 JAE 2988 JB 2988 JBE 2988 JC 2988 JCXZ 2988 JE 2988  
JG 2988 JGE 2988 JL 2988 JLE 2988 JMP 2988 JMP FAR 2988  
JNA 2988 JNAE 2988 JNB 2988 JNBE 2988 JNC 2988 JNE 2988  
JNG 2988 JNGE 2988 JNL 2988 JNO 2988 JNP 2988 JNS 2988  
JNZ 2988 JO 2988 JP 2988 JPE 2988 JPO 2988 JS 2988 JZ  
2988 LAHF 2952 LDS 2952 LEA 2952 LEAVE 2977 LES 2952  
LODSB 2957 LODSW 2957 LOOP 2995 LOOPE 2995 LOOPNE  
2995 LOOPNZ 2995 LOOPZ 2995 MOV 2952 MOVSB 2957 MOVSW  
2957 MUL 2970 NEG 2970 NOP 2952 NOT 2973 OR 2973 OUT 2997

2997 POP 2977 POPA 2977 POPF 2977 PUSH 2977 PUSHA 2977  
PUSHF 2977 RCL 2974 RCR 2974 REP 2957 REPE 2964 REPNE  
2964 REPNZ 2964 REPZ 2964 RET 2977 RETF 2977 RET FAR  
2977 ROL 2974 ROR 2974 SAHF 2952 SAL 2974 SAR 2974 SBB  
2970 SCASB 2964 SCASW 2964 SHL 2974 SHR 2974 SI 2943 SP  
2943 STC 2986 2986 STI 2983 STOSB 2957 STOSW 2957 SUB  
2970 TEST 2986 XCHG 2952 XLATB 2957 XOR 2973

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 ':':

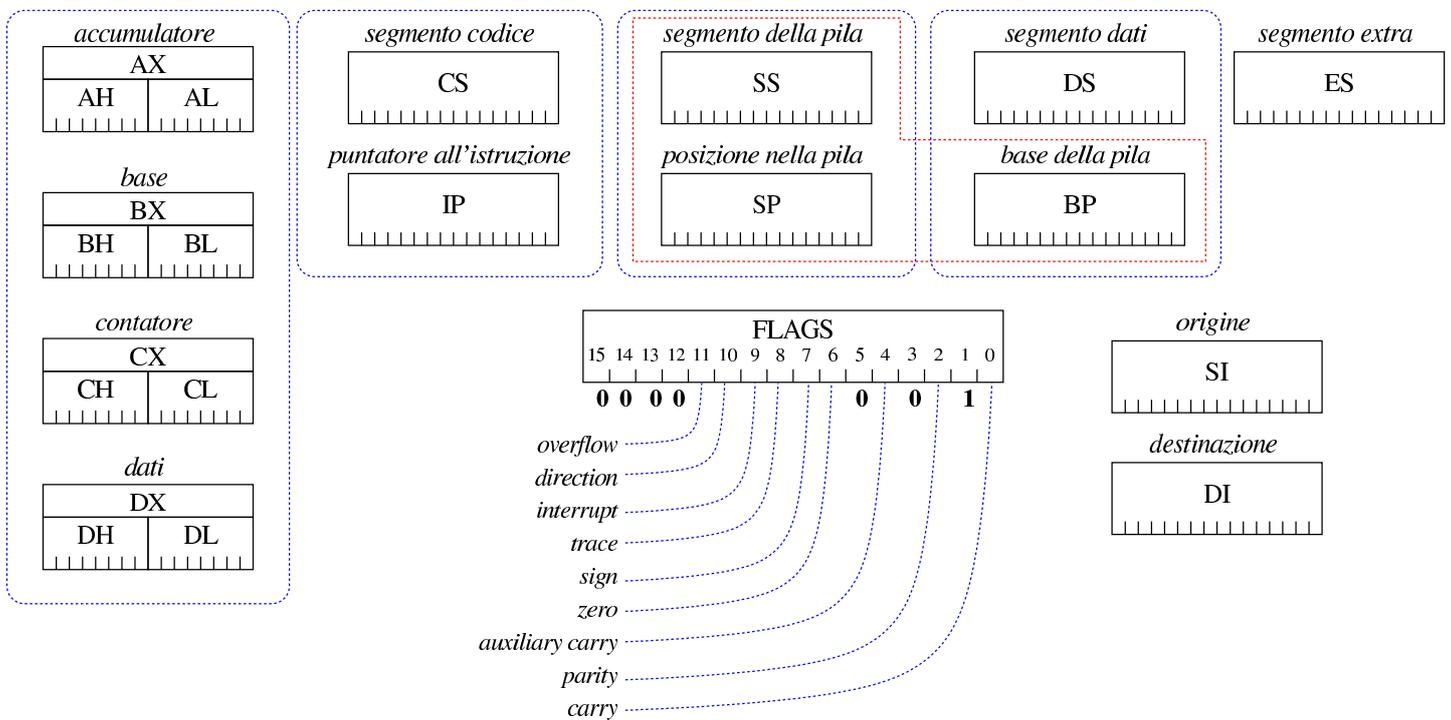
Nella programmazione a 16 bit, con i microprocessori della famiglia x86, per affiancare due valori a 16 bit si usa normalmente il segno di due punti, come per esempio  $DX:AX$  o  $159D_{16}:FFFE_{16}$ .

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  $DS \cdot 16 + SI$ . 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 ( <i>flag</i> )	Bit	Descrizione
<b>C</b> <i>carry</i>	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 <i>borrow</i> ).
1	1	Riservato.
<b>P</b> <i>parity</i>	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.
<b>A</b> <i>auxiliary carry</i>	4	È un tipo di riporto ausiliario.
0	5	Riservato.
<b>Z</b> <i>zero</i>	6	Viene impostato dopo un'operazione che dà come risultato il valore zero.
<b>S</b> <i>sign</i>	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.
<b>T</b> <i>trace</i>	8	Se è attivo, fa in modo che il microprocessore possa funzionare un passo alla volta.
<b>I</b> <i>interrupt</i>	9	Se è attivo, le interruzioni hardware sono abilitate, diversamente risultano bloccate.

Indicatore ( <i>flag</i> )	Bit	Descrizione
<b>D</b> <i>direction</i>	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.
<b>O</b> <i>overflow</i>	11	È l'indicatore di traboccamento per le operazioni che riguardano valori con segno.
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
<b>AX</b>	accumulatore	Usato soprattutto nei calcoli e per l'input e output. Nelle convenzioni di chiamata comuni, si usa <b>AX</b> per restituire un valore attraverso una funzione.
<b>BX</b>	base	Viene usato particolarmente come indice da sommare ad altri, per individuare una posizione in memoria.
<b>CX</b>	contatore	Usato come contatore nei cicli.

Registro	Definizione	Scopo prevalente
<b><i>DX</i></b>	dati	Si affianca a <b><i>AX</i></b> , soprattutto nelle divisioni e moltiplicazioni.
<b><i>SI</i></b>	<i>source index</i>	Usato prevalentemente come indice dell'origine, nell'ambito di un segmento dati ( <b><i>DS</i></b> o <b><i>ES</i></b> ).
<b><i>DI</i></b>	<i>destination index</i>	Usato prevalentemente come indice della destinazione, nell'ambito di un segmento dati ( <b><i>DS</i></b> o <b><i>ES</i></b> ).

## 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  $DS \cdot 16 + DX + SI - 2$  (**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:

$$[\textit{costante}] + [\textit{BX} | \textit{BP}] + [\textit{SI} | \textit{DI}]$$

Lo specchio 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]	$DS \cdot 16 + SI$	<i>imm</i> [SI]	$DS \cdot 16 + SI + imm$
[DI]	$DS \cdot 16 + DI$	<i>imm</i> [DI]	$DS \cdot 16 + DI + imm$
[BP]	$DS \cdot 16 + BP$	<i>imm</i> [BP]	$DS \cdot 16 + BP + imm$
[BX]	$DS \cdot 16 + BX$	<i>imm</i> [BX]	$DS \cdot 16 + BX + imm$
[BX+SI]	$DS \cdot 16 + BX + SI$	<i>imm</i> [BX+SI]	$DS \cdot 16 + BX + SI + imm$
[BX+DI]	$DS \cdot 16 + BX + DI$	<i>imm</i> [BX+DI]	$DS \cdot 16 + BX + DI + imm$
[BP+SI]	$DS \cdot 16 + BP + SI$	<i>imm</i> [BP+SI]	$DS \cdot 16 + BP + SI + imm$
[BP+DI]	$DS \cdot 16 + BP + DI$	<i>imm</i> [BP+DI]	$DS \cdot 16 + BP + DI + imm$

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 '**AAx**' e '**DAx**', 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 defi-

nire 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,  $(DS \cdot 16 + SI)$  indica un indirizzo in memoria, individuato dal registro *SI* che si riferisce al segmento annotato in *DS*. In modo analogo,  $*(DS \cdot 16 + SI)$  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: <i>dst, org1, org2</i>	Descrizione	Indicatori
NOP		<i>not operate</i> Istruzione nulla.	cpazstido .....

Nome	Operandi: <i>dst, org1, org2</i>	Descrizione	Indicatori
MOV	<i>reg, reg</i> <i>reg, mem</i> <i>reg, imm</i> <i>mem, reg</i> <i>mem, imm</i>	<i>move</i> 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. <i>dst := org</i>	cpazstido .....
LEA	<i>reg, mem</i>	<i>load effective address</i> Mette nel registro l'indirizzo della memoria, inteso come scostamento dall'inizio del segmento dati. <i>dst := &amp;org</i>	cpazstido .....

Nome	Operandi: <i>dst, org1, org2</i>	Descrizione	Indicatori
LDS	<b><i>reg, mem</i></b>	<p><i>load pointer using DS</i> 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 <b><i>DS</i></b> (segmento dati).</p> <p><b><i>DS:dst := org</i></b></p>	<p>cpazstido .....</p>
LES	<b><i>reg, mem</i></b>	<p><i>load pointer using ES</i> 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 <b><i>ES</i></b>.</p> <p><b><i>ES:dst := org</i></b></p>	<p>cpazstido .....</p>
XCHG	<b><i>reg, reg</i></b> <b><i>reg, mem</i></b> <b><i>mem, reg</i></b>	<p><i>exchange data</i> Scambia i valori.</p> <p><b><i>dst ::=: org</i></b></p>	<p>cpazstido .....</p>

Nome	Operandi: <i>dst, org1, org2</i>	Descrizione	Indicatori
CBW		<p><i>convert byte to word</i></p> <p>Converte un intero con segno, della dimensione di 8 bit, contenuto in <b>AL</b>, in modo da occupare tutto <b>AX</b> (da 8 bit a 16 bit). L'espansione tiene conto del segno.</p> <p><b>AX</b> := <b>AL</b></p>	<p>cpazstido .....</p>
CWD		<p><i>convert word to double word</i></p> <p>Converte un intero con segno, della dimensione di 16 bit, contenuto in <b>AX</b>, in modo da estendersi anche in <b>DX</b>, tenendo conto del segno.</p> <p>IF <b>AX</b> &gt;= 0</p> <p>THEN</p> <p style="padding-left: 40px;"><b>DX</b> := 0</p> <p>ELSE</p> <p style="padding-left: 40px;"><b>DX</b> := FFFF<sub>16</sub></p>	<p>cpazstido .....</p>

Nome	Operandi: <i>dst, org1, org2</i>	Descrizione	Indicatori
LAHF		<p><i>load flags into AH</i></p> <p>Carica i primi otto indicatori in <i>AH</i>, escludendo quelli riservati.</p> <p><math>AH_{bit0} := c</math></p> <p><math>AH_{bit1} := 1</math></p> <p><math>AH_{bit2} := p</math></p> <p><math>AH_{bit3} := 0</math></p> <p><math>AH_{bit4} := a</math></p> <p><math>AH_{bit5} := 0</math></p> <p><math>AH_{bit6} := z</math></p> <p><math>AH_{bit7} := s</math></p>	<p>cpazstido</p> <p>#####</p>
SAHF		<p><i>store AH into flags</i></p> <p>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 <i>AH</i>.</p> <p><math>c := AH_{bit0}</math></p> <p><math>p := AH_{bit2}</math></p> <p><math>a := AH_{bit4}</math></p> <p><math>z := AH_{bit6}</math></p> <p><math>s := AH_{bit7}</math></p>	<p>cpazstido</p> <p>#####</p>

Tabella u139.17. Movimento di dati.

Nome	Operandi: <i>dst, org1, org2</i>	Descrizione	Indicatori
LODSB		<p><i>load string byte</i></p> <p>Dall'indirizzo a cui punta la coppia <b><i>DS:SI</i></b> (<math>DS \cdot 16 + SI</math>), viene letto un byte e copiato in <b><i>AL</i></b>. Se l'indicatore di direzione è pari a zero, <b><i>SI</i></b> viene incrementato di una unità, altrimenti viene decrementato di una unità.</p> <p><b><i>AL := *(DS · 16 + SI)</i></b></p> <p>IF <b><i>d == 0</i></b>  THEN            <b><i>SI++</i></b>  ELSE            <b><i>SI--</i></b></p>	<p>cpazstido  .....t.  .....</p>

Nome	Operandi: <i>dst, org1, org2</i>	Descrizione	Indicatori
LODSW		<p><i>load string word</i></p> <p>Dall'indirizzo a cui punta la coppia <b><i>DS:SI</i></b> (<b><i>DS·16+SI</i></b>), viene letto un blocco da 16 bit e copiato in <b><i>AX</i></b>. Se l'indicatore di direzione è pari a zero, <b><i>SI</i></b> viene incrementato di due unità, altrimenti viene decrementato di due unità.</p> <p><b><i>AL :=*(DS·16+SI)</i></b></p> <p>IF <b><i>d==0</i></b>  THEN            <b><i>SI :=+2</i></b>  ELSE            <b><i>SI :=-2</i></b></p>	<p>cpazstido  .....t.  .....</p>

Nome	Operandi: <i>dst, org1, org2</i>	Descrizione	Indicatori
STOSB		<p><i>store string byte</i></p> <p>All'indirizzo a cui punta la coppia <b>ES:DI</b> (<math>ES \cdot 16 + DI</math>), viene scritto il valore contenuto in <b>AL</b>, aggiornando <b>DI</b> in base al contenuto dell'indicatore di direzione.</p> <p><math>*(ES \cdot 16 + DI) := AL</math></p> <p>IF <math>d == 0</math>  THEN            <b>DI++</b>  ELSE            <b>DI--</b></p>	<p>cpazstido  .....t.  .....</p>

Nome	Operandi: <i>dst, org1, org2</i>	Descrizione	Indicatori
STOSW		<p><i>store string word</i></p> <p>All'indirizzo a cui punta la coppia <b>ES:DI</b> (<math>ES \cdot 16 + DI</math>), viene scritto il valore contenuto in <b>AX</b>, aggiornando <b>DI</b> in base al contenuto dell'indicatore di direzione.</p> <p><math>*(ES \cdot 16 + DI) := AX</math></p> <p>IF <math>d == 0</math>  THEN            <b>DI</b> : += 2  ELSE            <b>DI</b> : -= 2</p>	<p>cpazstido  .....t.  .....</p>

Nome	Operandi: <i>dst, org1, org2</i>	Descrizione	Indicatori
MOVSB		<p><i>move string byte</i></p> <p>Copia un byte, dall'indirizzo a cui punta la coppia <i>DS:SI</i> (<math>DS \cdot 16 + SI</math>), all'indirizzo a cui punta la coppia <i>ES:DI</i> (<math>ES \cdot 16 + DI</math>), aggiornando <i>SI</i> e <i>DI</i> in base al valore dell'indicatore di direzione.</p> <p><math>*(ES \cdot 16 + DI) := *(DS \cdot 16 + SI)</math></p> <p>IF <math>d == 0</math>  THEN            <i>SI</i>++            <i>DI</i>++  ELSE            <i>SI</i>--            <i>DI</i>--</p>	<p>cpazstido  .....t.  .....</p>

Nome	Operandi: <i>dst, org1, org2</i>	Descrizione	Indicatori
MOVSW		<p><i>move string word</i></p> <p>Copia un blocco di 16 bit, dall'indirizzo a cui punta la coppia <b><i>DS:SI</i></b> (<math>DS \cdot 16 + SI</math>), all'indirizzo a cui punta la coppia <b><i>ES:DI</i></b> (<math>ES \cdot 16 + DI</math>), aggiornando <i>SI</i> e <i>DI</i> in base al valore dell'indicatore di direzione.</p> <p><b><i>*(ES·16+DI) := *(DS·16+SI)</i></b></p> <p>IF <i>d</i>==0 THEN     <b><i>SI</i></b> : +=2     <b><i>DI</i></b> : +=2 ELSE     <b><i>SI</i></b> : -=2     <b><i>DI</i></b> : -=2</p>	<p>cpazstido .....t. .....</p>

Nome	Operandi: <i>dst, org1, org2</i>	Descrizione	Indicatori
REP		<p><i>repeat</i></p> <p>Ripete l'istruzione successiva (che può essere una tra: '<b>LODSB</b>', '<b>LODSW</b>', '<b>STOSB</b>', '<b>STOSW</b>', '<b>MOVSB</b>', '<b>MOVSW</b>'), per <b>CX</b> volte.</p> <p>IF <b>CX</b>!=0</p> <p>THEN</p> <p style="padding-left: 40px;">istruzione successiva</p> <p style="padding-left: 40px;"><b>CX</b>--</p> <p>ELSE</p> <p style="padding-left: 40px;">break</p>	<p>cpazstido</p> <p>.....</p>
XLATB		<p><i>translate table to byte</i></p> <p>Assegna a <b>AL</b> il valore che si può raggiungere all'indirizzo composto da <b>DS:BX+AL</b> (<b>DS·16+BX+AL</b>), dove <b>AL</b> va inteso come valore senza segno.</p> <p><b>AL :=*(DS·16+BX+AL)</b></p>	<p>cpazstido</p> <p>.....</p>

Tabella u139.18. Confronti con la memoria.

Nome	Operandi: <i>dst, org1, org2</i>	Descrizione	Indicatori
SCASB		<p><i>compare string byte</i></p> <p>Confronta il contenuto di <b>AL</b> con il valore a cui punta la coppia <b>ES:DI</b> (<math>ES \cdot 16 + DI</math>), aggiornando di conseguenza gli indicatori e anche il registro <b>DI</b> in base all'indicatore di direzione.</p> <p><math>*(ES \cdot 16 + DI) - AL</math></p> <p>IF <math>d == 0</math></p> <p>THEN</p> <p style="padding-left: 40px;"><b>DI</b>++</p> <p>ELSE</p> <p style="padding-left: 40px;"><b>DI</b>--</p>	<p>cpazstido</p> <p>.....t.</p> <p>#####...</p>

Nome	Operandi: <i>dst, org1, org2</i>	Descrizione	Indicatori
SCASW		<p><i>compare string word</i></p> <p>Confronta il contenuto di <b>AX</b> con il valore a cui punta la coppia <b>ES:DI</b> (<b>ES·16+DI</b>), aggiornando di conseguenza gli indicatori e anche il registro <b>DI</b> in base all'indicatore di direzione.</p> <p><b>*(ES·16+DI)-AX</b></p> <p>IF <b>d==0</b></p> <p>THEN</p> <p style="padding-left: 40px;"><b>DI : +=2</b></p> <p>ELSE</p> <p style="padding-left: 40px;"><b>DI : -=2</b></p>	<p>cpazstido</p> <p>.....t.</p> <p>#####...</p>

Nome	Operandi: <i>dst, org1, org2</i>	Descrizione	Indicatori
CMP SB		<p><i>compare string byte in memory</i></p> <p>Confronta il byte a cui punta la coppia <b>ES:DI</b> (<math>ES \cdot 16 + DI</math>), con quello a cui punta la coppia <b>DS:SI</b> (<math>DS \cdot 16 + SI</math>), aggiornando di conseguenza gli indicatori e anche i registri <b>DI</b> e <b>SI</b> in base all'indicatore di direzione.</p> <p><math>*(DS \cdot 16 + SI) - *(ES \cdot 16 + DI)</math></p> <p>IF <math>d == 0</math></p> <p>THEN</p> <p style="padding-left: 40px;"><b>SI</b>++</p> <p style="padding-left: 40px;"><b>DI</b>++</p> <p>ELSE</p> <p style="padding-left: 40px;"><b>SI</b>++</p> <p style="padding-left: 40px;"><b>DI</b>--</p>	<p>cpazstido</p> <p>.....t.</p> <p>#####...</p>

Nome	Operandi: <i>dst, org1, org2</i>	Descrizione	Indicatori
CMP SW		<p><i>compare string word in memory</i></p> <p>Confronta il blocco da 16 bit a cui punta la coppia <b>ES:DI</b> (<math>ES \cdot 16 + DI</math>), con quello a cui punta la coppia <b>DS:SI</b> (<math>DS \cdot 16 + SI</math>), aggiornando di conseguenza gli indicatori e anche i registri <b>DI</b> e <b>SI</b> in base all'indicatore di direzione.</p> <p><math>*(DS \cdot 16 + SI) - *(ES \cdot 16 + DI)</math></p> <p>IF <math>d == 0</math>  THEN            <b>SI</b>++            <b>DI</b>++  ELSE            <b>SI</b>++            <b>DI</b>--</p>	<p>cpazstido  .....t.  #####...</p>

Nome	Operandi: <i>dst, org1, org2</i>	Descrizione	Indicatori
REPE  REPZ		<p><i>repeat while equal</i>  <i>repeat while zero</i></p> <p>Ripete l'istruzione successiva (che può essere una tra: 'SCASB', 'SCASW', 'CMPSB', 'CMPSW'), fino a che l'indicatore <i>z</i> è pari a uno (rappresentante l'uguaglianza di una comparazione, ovvero che la sottrazione dà zero), fino a un massimo di <b>CX</b> volte.</p> <pre> IF <b>CX</b> != 0 THEN     istruzione successiva     <b>CX</b>--     IF <i>z</i> == 1     THEN         continue     ELSE         break ELSE     break </pre>	cpazstido ...#.....

Nome	Operandi: <i>dst, org1, org2</i>	Descrizione	Indicatori
REPNE  REPNZ		<p><i>repeat while not equal</i> <i>repeat while not zero</i></p> <p>Ripete l'istruzione successiva (che può essere una tra: 'SCASB', 'SCASW', 'CMPSB', 'CMPSW'), fino a che l'indicatore <i>z</i> è pari a zero (rappresentante la disuguaglianza della comparazione, ovvero che la sottrazione non dà zero), fino a un massimo di <b>CX</b> volte.</p> <pre> IF <b>CX</b> != 0 THEN     istruzione successiva     <b>CX</b>--     IF <i>z</i> == 0     THEN         continue     ELSE         break ELSE     break </pre>	<p>cpazstido ...#.....</p>

Tabella u139.19. Operazioni aritmetiche.

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

Nome	Operandi: <i>dst, org1, org2</i>	Descrizione	Indicatori
ADC	<i>reg, reg</i> <i>reg, mem</i> <i>reg, imm</i> <i>mem, reg</i> <i>mem, imm</i>	<i>addition with carry</i> Somma di interi, con o senza segno, aggiungendo anche il riporto precedente (l'indicatore <i>carry</i> ).  <i>dst := org + dst + c</i>	cpazstido t..... ##.##...#
SBB	<i>reg, reg</i> <i>reg, mem</i> <i>reg, imm</i> <i>mem, reg</i> <i>mem, imm</i>	<i>subtraction with borrow</i> Sottrazione di interi, con o senza segno, tenendo conto del «prestito» precedente (l'indicatore <i>carry</i> ).  <i>dst := org + dst - c</i>	cpazstido t..... ##.##...#
INC	<i>reg</i> <i>mem</i>	<i>increment</i> Incrementa di una unità un intero.  <i>operand++</i>	cpazstido .#.##...#
DEC	<i>reg</i> <i>mem</i>	<i>decrement</i> Decrementa di una unità un valore intero.  <i>operand--</i>	cpazstido .#.##...#

Nome	Operandi: <i>dst, org1, org2</i>	Descrizione	Indicatori
MUL	<i>reg mem</i>	<p><i>multiply</i></p> <p>Moltiplicazione intera senza segno. L'operando è il moltiplicatore, mentre il moltiplicando è costituito da registri prestabiliti.</p> <p><i>AX := AL*operand</i></p> <p><i>DX:AX := AX*operand</i></p>	<p>cpazstido</p> <p>#?·??·...#</p>
DIV	<i>reg mem</i>	<p><i>division</i></p> <p>Divisione intera senza segno. L'operando è il divisore, mentre il dividendo è costituito da registri prestabiliti.</p> <p><i>AL := AX/operand</i></p> <p><i>AH := AX%operand</i></p> <p><i>AX := (DX:AX)/operand</i></p> <p><i>DX := (DX:AX)%operand</i></p>	<p>cpazstido</p> <p>??·??·...?</p>
IMUL	<i>reg mem</i>	<p><i>signed multiply</i></p> <p>Moltiplicazione intera con segno. In questo caso l'operando è il moltiplicatore, mentre il moltiplicando è costituito da registri prestabiliti.</p> <p><i>AX := AL*operand</i></p> <p><i>(DX:AX) := AX*operand</i></p>	<p>cpazstido</p> <p>#?·??·...#</p>

Nome	Operandi: <i>dst, org1, org2</i>	Descrizione	Indicatori
IDIV	<i>reg</i> <i>mem</i>	<p><i>signed division</i></p> <p>Divisione intera con segno. L'operando è il divisore, mentre il dividendo è costituito da registri prestabiliti.</p> <p><i>AL := AX / operand</i></p> <p><i>AH := AX % operand</i></p> <p><i>AX := (DX:AX) / operand</i></p> <p><i>DX := (DX:AX) % operand</i></p>	<p>cpazstido</p> <p>??·??·...?</p>

Tabella u139.20. Operazioni logiche.

Nome	Operandi: <i>dst, org1, org2</i>	Descrizione	Indicatori
NOT	<i>reg</i> <i>mem</i>	<p>NOT di tutti i bit dell'operando.</p> <p><i>dst := NOT dst</i></p>	<p>cpazstido</p> <p>.....</p>
AND OR XOR	<p><i>reg, reg</i></p> <p><i>reg, mem</i></p> <p><i>reg, imm</i></p> <p><i>mem, reg</i></p> <p><i>mem, imm</i></p>	<p>AND, OR, o XOR, tra tutti i bit dei due operandi.</p> <p><i>dst := org AND dst</i></p> <p><i>dst := org OR dst</i></p> <p><i>dst := org XOR dst</i></p>	<p>cpazstido</p> <p>0#·##·...0</p>

Tabella u139.21. Scorrimenti e rotazioni.

Nome	Operandi: <i>dst, org1, org2</i>	Descrizione	Indicatori
SHL  SHR	<i>reg, 1</i> <i>mem, 1</i> <i>reg</i> <i>mem</i>	<i>shift left</i> <i>shift right</i> Fa scorrere i bit, rispettivamente verso sinistra o verso destra (l'ultima cifra perduta finisce nell'indicatore del rapporto). Se appare un solo operando, la rotazione viene eseguita <b>CL</b> volte. Se il valore immediato è maggiore di uno, è il compilatore che ripete l'istruzione per più volte.	cpazstido #.....#

Nome	Operandi: <i>dst, org1, org2</i>	Descrizione	Indicatori
SAL  SAR	<i>reg, 1</i> <i>mem, 1</i>  <i>reg</i> <i>mem</i>	<i>shift arithmetically left</i> <i>shift arithmetically right</i> 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 <b>CL</b> volte. Se il valore immediato è maggiore di uno, è il compilatore che ripete l'istruzione per più volte.	cpazstido #.....

Nome	Operandi: <i>dst, org1, org2</i>	Descrizione	Indicatori
RCL RCR	<i>reg, 1</i> <i>mem, 1</i> <i>reg</i> <i>mem</i>	<i>rotate left with carry</i> <i>rotate right with carry</i> Ruota i bit, rispettivamente verso sinistra o verso destra, utilizzando anche l'indicatore di riporto ( <i>carry</i> ). Se appare un solo operando, la rotazione viene eseguita <b>CL</b> volte. Se il valore immediato è maggiore di uno, è il compilatore che ripete l'istruzione per più volte.	cpazstido t..... #.....#
ROL ROR	<i>reg, 1</i> <i>mem, 1</i> <i>reg</i> <i>mem</i>	<i>rotate left</i> <i>rotate right</i> Ruota i bit, rispettivamente verso sinistra o verso destra. Se appare un solo operando, la rotazione viene eseguita <b>CL</b> 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: <i>dst, org1, org2</i>	Descrizione	Indicatori
CALL	<i>reg</i> <i>mem</i> <i>imm</i>	<p>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 (<i>CS</i>). Pertanto, l'indirizzo a cui ci si riferisce è a 16 bit.</p> <p>push <i>indirizzo_successivo</i></p> <p><i>IP := operand</i></p>	<p>cpazstido .....</p>
CALL FAR	<i>imm:imm</i>	<p><i>call procedure</i></p> <p>Inserisce nella pila il valore di <i>CS</i> e poi l'indirizzo dell'istruzione successiva (<i>IP</i> 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.</p> <p>push <i>CS</i></p> <p>push <i>indirizzo_successivo</i></p> <p><i>CS:IP := operand</i></p>	<p>cpazstido .....</p>

Nome	Operandi: <i>dst, org1, org2</i>	Descrizione	Indicatori
RET		<p><i>return from call</i></p> <p>Estrae dalla pila l'indirizzo dell'istruzione da raggiungere (<i>IP</i>) e salta a quella (serve a concludere una chiamata eseguita con '<b>CALL</b>').</p> <p>pop <i>IP</i></p>	<p>cpazstido</p> <p>.....</p>
RETF  RET FAR		<p><i>return from far call</i></p> <p>Estrae dalla pila il valore di <i>CS</i> e quindi l'indirizzo dell'istruzione da raggiungere (<i>IP</i>) e salta a quella (serve a concludere una chiamata eseguita con '<b>CALL FAR</b>').</p> <p>pop <i>IP</i></p> <p>pop <i>CS</i></p>	<p>cpazstido</p> <p>.....</p>
PUSH	<i>reg</i> <i>mem</i>	<p><i>push data onto stack</i></p> <p>Inserisce nella pila il valore (della dimensione di un registro comune).</p> <p><i>SP</i> := -2</p> <p>* (<i>SS</i> · 16 + <i>SP</i>) := <i>operand</i></p>	<p>cpazstido</p> <p>.....</p>

Nome	Operandi: <i>dst, org1, org2</i>	Descrizione	Indicatori
POP	<i>reg mem</i>	<p><i>pop data from stack</i>  Estrae dalla pila l'ultimo valore inserito (della dimensione di un registro comune).</p> <p><i>operand := * (SS · 16 + SP)</i></p> <p><i>SP : +=2</i></p>	cpazstido .....
PUSHF		<p><i>push flags onto stack</i>  Inserisce nella pila l'insieme del registro degli indicatori (<i>FLAGS</i>).</p> <p>push <i>FLAGS</i></p>	cpazstido .....
POPF		<p><i>pop flags from stack</i>  Estrae dalla pila l'insieme del registro degli indicatori (<i>FLAGS</i>), aggiornando di conseguenza il registro stesso.</p> <p>pop <i>FLAGS</i></p>	cpazstido ?????????

Nome	Operandi: <i>dst, org1, org2</i>	Descrizione	Indicatori
ENTER	<i>imm8, 0</i>	<p><i>enter stack frame</i></p> <p>Questa funzione esiste a partire dai microprocessori i186.</p> <p>Inserisce nella pila il valore di <b>BP</b>, poi assegna a <b>BP</b> il valore di <b>SP</b> e infine decrementa <b>SP</b> del valore fornito come immediato. Serve a predisporre <b>BP</b> e <b>SP</b> all'inizio di una funzione, specificando lo spazio necessario per le variabili locali nella pila.</p> <p>push <b>BP</b></p> <p><b>BP</b> := <b>SP</b></p> <p><b>SP</b> := -2 * <i>dst</i></p>	<p>cpazstido</p> <p>.....</p>

Nome	Operandi: <i>dst, org1, org2</i>	Descrizione	Indicatori
PUSHA		<p><i>push all registers onto stack</i></p> <p>Questa funzione esiste a partire dai microprocessori i186.</p> <p>Inserisce nella pila i registri principali: <b>AX</b>, <b>CX</b>, <b>DX</b>, <b>BX</b>, <b>SP</b>, <b>BP</b>, <b>SI</b>, <b>DI</b>.</p> <p>push <b>AX</b></p> <p>push <b>CX</b></p> <p>push <b>DX</b></p> <p>push <b>BX</b></p> <p>push <b>SP</b></p> <p>push <b>BP</b></p> <p>push <b>SI</b></p> <p>push <b>DI</b></p>	<p>cpazstido</p> <p>.....</p>

Nome	Operandi: <i>dst, org1, org2</i>	Descrizione	Indicatori
POPA		<p><i>pop all registers from stack</i></p> <p>Questa funzione esiste a partire dai microprocessori i186.</p> <p>Ripristina i registri principali, estraendo i contenuti dalla pila: <b>DI</b>, <b>SI</b>, <b>BP</b>, <b>SP</b> viene eliminato senza aggiornare il registro, <b>BX</b>, <b>DX</b>, <b>CX</b>, <b>AX</b>. Come si vede, anche se '<b>PUSHA</b>' salva l'indice della pila, in pratica questo indice non viene ripristinato.</p> <p><code>pop <b>DI</b></code></p> <p><code>pop <b>SI</b></code></p> <p><code>pop <b>BP</b></code></p> <p><code><b>SP</b> += 2</code></p> <p><code>pop <b>BX</b></code></p> <p><code>pop <b>DX</b></code></p> <p><code>pop <b>CX</b></code></p> <p><code>pop <b>AX</b></code></p>	<p>cpazstido .....</p>

Nome	Operandi: <i>dst, org1, org2</i>	Descrizione	Indicatori
LEAVE		<p><i>leave stack frame</i></p> <p>Questa funzione esiste a partire dai microprocessori i186.</p> <p>Ripristina i valori di <b>BP</b> e di <b>SP</b>, allo stato che avevano prima dell'uso dell'istruzione '<b>ENTER</b>'.</p> <p><b>SP := BP</b></p> <p>pop <b>BP</b></p>	<p>cpazstido</p> <p>.....</p>

Tabella u139.23. Interruzioni.

Nome	Operandi: <i>dst, org1, org2</i>	Descrizione	Indicatori
INT	<i>imm8</i>	<p><i>call to interrupt</i></p> <p>Esegue una chiamata attraverso un'interruzione. Prima di saltare alla codice relativo all'interruzione selezionata, inserisce nella pila <b>FLAGS</b>, <b>CS</b> e <b>IP</b>. Azzera anche l'indicatore <b>IF</b> (<i>interrupt flag</i>), mentre gli altri indicatori rimangono inalterati.</p> <pre> pusf push CS push IP i:=0 jmp far 0:(operand·4) </pre>	<p>cpazstido .....</p>

Nome	Operandi: <i>dst, org1, org2</i>	Descrizione	Indicatori
IRET		<p><i>return from interrupt</i></p> <p>Conclude l'esecuzione del codice relativo a un'interruzione recuperando dalla pila i valori inseriti alla chiamata con '<b>INT</b>': <i>IP</i>, <i>CS</i> e <i>FLAGS</i>. Pertanto, il valore degli indicatori viene ripristinato allo stato precedente alla chiamata.</p> <p>pop <i>IP</i></p> <p>pop <i>CS</i></p> <p>popf</p>	<p>cpazstido</p> <p>##.##. . . #</p>
CLI		<p><i>clear interrupt flag</i></p> <p>Azzera l'indicatore di abilitazione delle interruzioni (<i>interrupt flag</i>), disabilitando di conseguenza le interruzioni hardware.</p> <p><i>i := 0</i></p>	<p>cpazstido</p> <p>. . . . . 0 . .</p>

Nome	Operandi: <i>dst, org1, org2</i>	Descrizione	Indicatori
STI		<p><i>set interrupt flag</i></p> <p>Attiva l'indicatore di abilitazione delle interruzioni (<i>interrupt flag</i>), abilitando di conseguenza le interruzioni hardware.</p> <p><math>i := 1</math></p>	<p>cpazstido</p> <p>.....1..</p>
HLT		<p><i>enter halt state</i></p> <p>Ferma il sistema, fino a quando viene ricevuta un'interruzione hardware.</p>	<p>cpazstido</p> <p>.....</p>
INTO		<p><i>interrupt if overflow</i></p> <p>Se l'indicatore di straripamento è attivo, esegue la chiamata dell'interruzione numero 4 (la quale dovrebbe gestire il problema).</p>	<p>cpazstido</p> <p>.....t</p> <p>.....</p>

Tabella u139.24. Indicatori e confronti tra registri.

Nome	Operandi: <i>dst, org1, org2</i>	Descrizione	Indicatori
CLC		<i>clear carry flag</i> Azzera l'indicatore del ri- porto ( <i>carry</i> ), senza inter- venire negli altri indicato- ri. <i>c := 0</i>	cpazstido 0.....
CLD		<i>clear direction flag</i> Azzera l'indicatore di di- rezione ( <i>direction</i> ), sen- za intervenire negli altri indicatori. <i>d := 0</i>	cpazstido .....0.
STC		<i>set carry flag</i> Attiva l'indicatore di ri- porto ( <i>carry</i> ). <i>c := 1</i>	cpazstido 1.....
STD		<i>set direction flag</i> Attiva l'indicatore di di- rezione ( <i>direction</i> ). <i>d := 1</i>	cpazstido .....1.
CMC		<i>complement carry flag</i> Inverte il valore dell'indi- catore del riporto ( <i>carry</i> ).	cpazstido #.....

Nome	Operandi: <i>dst, org1, org2</i>	Descrizione	Indicatori
CMP	<i>reg, reg</i> <i>reg, mem</i> <i>reg, imm</i> <i>mem, reg</i> <i>mem, imm</i>	<i>compare operands</i> 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. <i>dst - org</i>	cpazstido ##.##...#
TEST	<i>reg, reg</i> <i>reg, imm</i> <i>mem, reg</i> <i>mem, imm</i>	<i>logical compare</i> AND dei due valori senza conservare il risultato. Serve solo a ottenere l'aggiornamento degli indicatori. <i>dst AND org</i>	cpazstido 0#.##...0

Tabella u139.25. Salti.

Nome	Operandi: <i>dst, org1, org2</i>	Descrizione	Indicatori
JMP	<i>reg</i> <i>mem</i> <i>imm</i>	<i>jump</i> Salto incondizionato all'indirizzo indicato, che si intende relativo al segmento codice ( <i>CS</i> ).  <i>IP := operand</i>	cpazstido .....
JMP FAR	<i>imm:imm</i>	<i>far jump</i> Salto incondizionato all'indirizzo indicato, costituito sia dal segmento codice, sia dall'indirizzo relativo, all'interno di questo.  <i>CS:IP := operand</i>	cpazstido .....
JA JNBE	<i>imm</i>	<i>conditional jump</i> 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 <i>dst, org</i>  IF <i>dst &gt; org</i>  THEN  go to <i>imm</i>	cpazstido t..t.....

Nome	Operandi: <i>dst, org1, org2</i>	Descrizione	Indicatori
JAE  JNB	<i>imm</i>	<p style="text-align: center;"><i>conditional jump</i></p> <p>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.</p> <p>CMP <i>dst, org</i></p> <p>IF <i>dst</i> &gt;= <i>org</i></p> <p>THEN</p> <p style="text-align: center;">go to <i>imm</i></p>	cpazstido t..t.....
JB  JNAE	<i>imm</i>	<p style="text-align: center;"><i>conditional jump</i></p> <p>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.</p> <p>CMP <i>dst, org</i></p> <p>IF <i>dst</i> &lt; <i>org</i></p> <p>THEN</p> <p style="text-align: center;">go to <i>imm</i></p>	cpazstido t..t.....

Nome	Operandi: <i>dst, org1, org2</i>	Descrizione	Indicatori
<p>JBE</p> <p>JNA</p>	<p><i>imm</i></p>	<p><i>conditional jump</i></p> <p>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.</p> <p>CMP <i>dst, org</i></p> <p>IF <i>dst</i> &lt;= <i>org</i></p> <p>THEN</p> <p style="padding-left: 40px;">go to <i>imm</i></p>	<p>cpazstido</p> <p>t..t.....</p>
<p>JE</p>	<p><i>imm</i></p>	<p><i>conditional jump</i></p> <p>Dopo un confronto, indipendentemente dal segno, salta se la destinazione era uguale all'origine. Il salto riguarda solo l'ambito del segmento codice attuale.</p> <p>CMP <i>dst, org</i></p> <p>IF <i>dst</i> == <i>org</i></p> <p>THEN</p> <p style="padding-left: 40px;">go to <i>imm</i></p>	<p>cpazstido</p> <p>...t.....</p>

Nome	Operandi: <i>dst, org1, org2</i>	Descrizione	Indicatori
JNE	<i>imm</i>	<p><i>conditional jump</i></p> <p>Dopo un confronto, indipendentemente dal segno, salta se la destinazione era diversa dall'origine. Il salto riguarda solo l'ambito del segmento codice attuale.</p> <p>CMP <i>dst, org</i></p> <p>IF <i>dst</i> != <i>org</i></p> <p>THEN</p> <p>    go to <i>imm</i></p>	<p>cpazstido</p> <p>...t.....</p>
JG JNLE	<i>imm</i>	<p><i>conditional jump</i></p> <p>Dopo un confronto con segno, salta se la destinazione era maggiore dell'origine. Il salto riguarda solo l'ambito del segmento codice attuale.</p> <p>CMP <i>dst, org</i></p> <p>IF <i>dst</i> &gt; <i>org</i></p> <p>THEN</p> <p>    go to <i>imm</i></p>	<p>cpazstido</p> <p>...tt...t</p>

Nome	Operandi: <i>dst, org1, org2</i>	Descrizione	Indicatori
JGE  JNL	<i>imm</i>	<p><i>conditional jump</i></p> <p>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.</p> <p>CMP <i>dst, org</i></p> <p>IF <i>dst</i> &gt;= <i>org</i></p> <p>THEN</p> <p>    go to <i>imm</i></p>	<p>cpazstido</p> <p>...tt...t</p>
JL  JNGE	<i>imm</i>	<p><i>conditional jump</i></p> <p>Dopo un confronto con segno, salta se la destinazione era minore dell'origine. Il salto riguarda solo l'ambito del segmento codice attuale.</p> <p>CMP <i>dst, org</i></p> <p>IF <i>dst</i> &lt; <i>org</i></p> <p>THEN</p> <p>    go to <i>imm</i></p>	<p>cpazstido</p> <p>...tt...t</p>

Nome	Operandi: <i>dst, org1, org2</i>	Descrizione	Indicatori
JLE JNG	<i>imm</i>	<p><i>conditional jump</i></p> <p>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.</p> <p>CMP <i>dst, org</i></p> <p>IF <i>dst</i> &lt;= <i>org</i></p> <p>THEN</p> <p style="padding-left: 40px;">go to <i>imm</i></p>	cpazstido ...tt...t
JC JNC	<i>imm</i>	<p><i>conditional jump</i></p> <p>Salta se l'indicatore del riporto (<i>carry</i>), rispettivamente, è attivo, oppure non è attivo. Il salto riguarda solo l'ambito del segmento codice attuale.</p>	cpazstido t.....
JO JNO	<i>imm</i>	<p><i>conditional jump</i></p> <p>Salta se l'indicatore di traboccamento (<i>overflow</i>), rispettivamente, è attivo, oppure non è attivo.</p>	cpazstido .....t
JS JNS	<i>imm</i>	<p><i>conditional jump</i></p> <p>Salta se l'indicatore di segno (<i>sign</i>), rispettivamente, è attivo, oppure non è attivo.</p>	cpazstido ...t....

Nome	Operandi: <i>dst, org1, org2</i>	Descrizione	Indicatori
JZ JNZ	<i>imm</i>	<i>conditional jump</i> Salta se l'indicatore di zero, rispettivamente, è attivo, oppure non è attivo.	cpazstido ...t.....
JP JPE	<i>imm</i>	<i>conditional jump</i> Salta se l'indicatore di parità è attivo.	cpazstido ·t.....
JNP JPO	<i>imm</i>	<i>conditional jump</i> Salta se l'indicatore di parità non è attivo.	cpazstido ·t.....
JCXZ	<i>imm</i>	<i>conditional jump</i> Salta se il valore contenuto nel registro <b>CX</b> è pari a zero.	cpazstido .....

Tabella u139.26. Iterazioni

Nome	Operandi: <i>dst, org1, org2</i>	Descrizione	Indicatori
LOOP	<i>imm8</i>	<p><i>loop</i></p> <p>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.</p>	<p>cpazstido .....</p>
LOOPE  LOOPZ	<i>imm8</i>	<p><i>conditional loop</i></p> <p>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.</p>	<p>cpazstido ...t.....</p>

Nome	Operandi: <i>dst, org1, org2</i>	Descrizione	Indicatori
LOOPNE  LOOPNZ	<i>imm8</i>	<i>conditional loop</i> 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: <i>dst, org1, org2</i>	Descrizione	Indicatori
IN	<i>AL, imm8</i> <i>AX, imm8</i>	<i>input</i> Assegna a <i>AL</i> o <i>AX</i> il valore letto dalla porta specificata; in tal caso il numero di porta non può essere superiore a 255.	cpazstido .....
IN	<i>AL, DX</i> <i>AX, DX</i>	<i>input</i> Assegna a <i>AL</i> o <i>AX</i> il valore letto dalla porta specificata da <i>DX</i> ; in tal caso il numero di porta può essere superiore a 255.	cpazstido .....

Nome	Operandi: <i>dst, org1, org2</i>	Descrizione	Indicatori
OUT	<i>imm8, AL</i> <i>imm8, AX</i>	<i>output</i> Scrive nella porta specificata il valore contenuto in <i>AL</i> o <i>AX</i> ; in tal caso il numero di porta non può essere superiore a 255.	cpazstido .....
OUT	<i>DX, AL</i> <i>DX, AX</i>	<i>output</i> Scrive nella porta indicata da <i>DX</i> il valore contenuto in <i>AL</i> o <i>AX</i> ; 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](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://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/>



# Architettura IBM PC

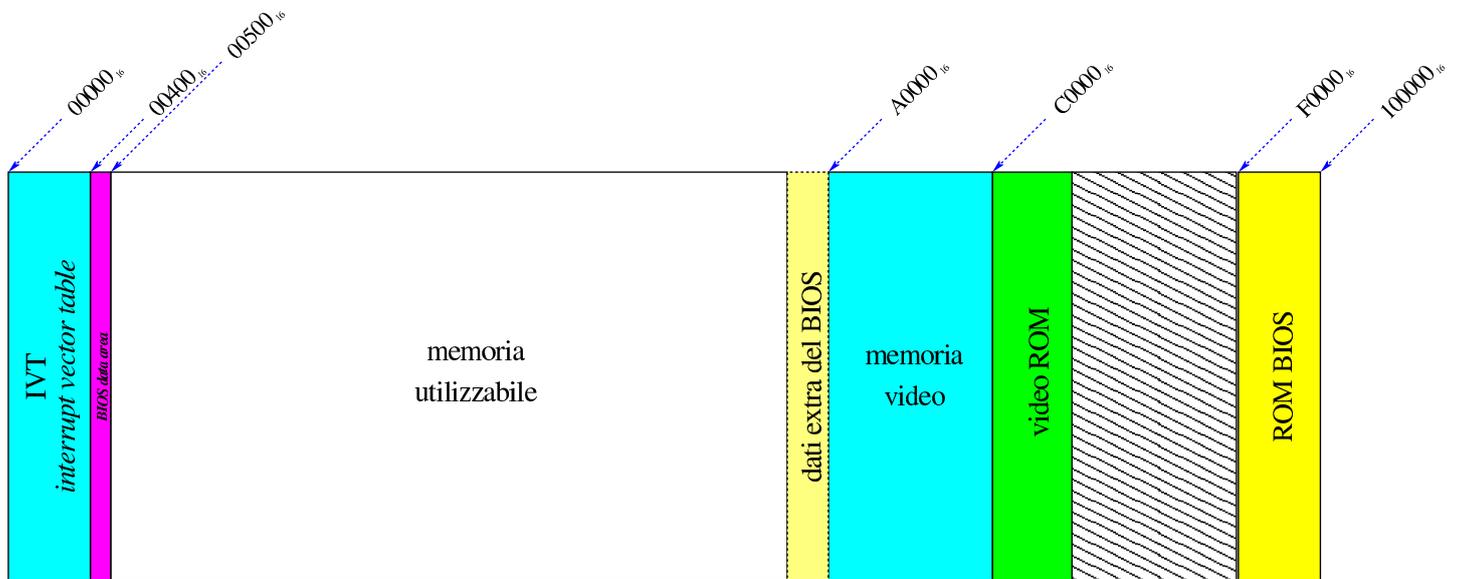


IVT: «interrupt vector table» .....	3004
BIOS data area .....	3007
Altre aree di memoria .....	3008
Interruzioni principali .....	3008
INT 10 <sub>16</sub> /AH=00 <sub>16</sub> «set video mode» .....	3009
INT 10 <sub>16</sub> /AH=05 <sub>16</sub> «select active display page» .....	3009
INT 10 <sub>16</sub> /AH=0E <sub>16</sub> «teletype output» .....	3009
INT 12 <sub>16</sub> «get memory size» .....	3010
INT 13 <sub>16</sub> /AH=00 <sub>16</sub> «reset disk system» .....	3010
INT 13 <sub>16</sub> /AH=02 <sub>16</sub> «read disk sectors into memory» .....	3011
INT 13 <sub>16</sub> /AH=03 <sub>16</sub> «write disk sectors» .....	3012
INT 16 <sub>16</sub> /AH=00 <sub>16</sub> «get keystroke from keyboard» .....	3013
INT 16 <sub>16</sub> /AH=01 <sub>16</sub> «check for keystroke in the keyboard buffer» .....	3013
Riferimenti .....	3014

L'architettura del vecchio IBM PC prevede 1 Mibyte di memoria, in cui alcune fasce hanno degli scopi particolari e non possono essere utilizzate diversamente. Quando si programma a 32 bit, di norma si dispone, in proporzione, di una quantità enorme di memoria, per cui di solito lo spazio inferiore a 1 Mibyte viene semplicemente ignorato e si considera solo lo spazio successivo; ma se la programmazione avviene a 16 bit, si deve operare in quello spazio ristretto.

Buona parte della memoria di un elaboratore conforme all'architettura del IBM PC è impegnata per il codice del BIOS, in cui sono contenute diverse routine attivate da interruzioni, hardware o software. Nella limitazione dell'architettura originale, il sistema offerto di gestione delle interruzioni consentirebbe un utilizzo uniforme dell'hardware.

Figura u140.1. Mappa generale della memoria.



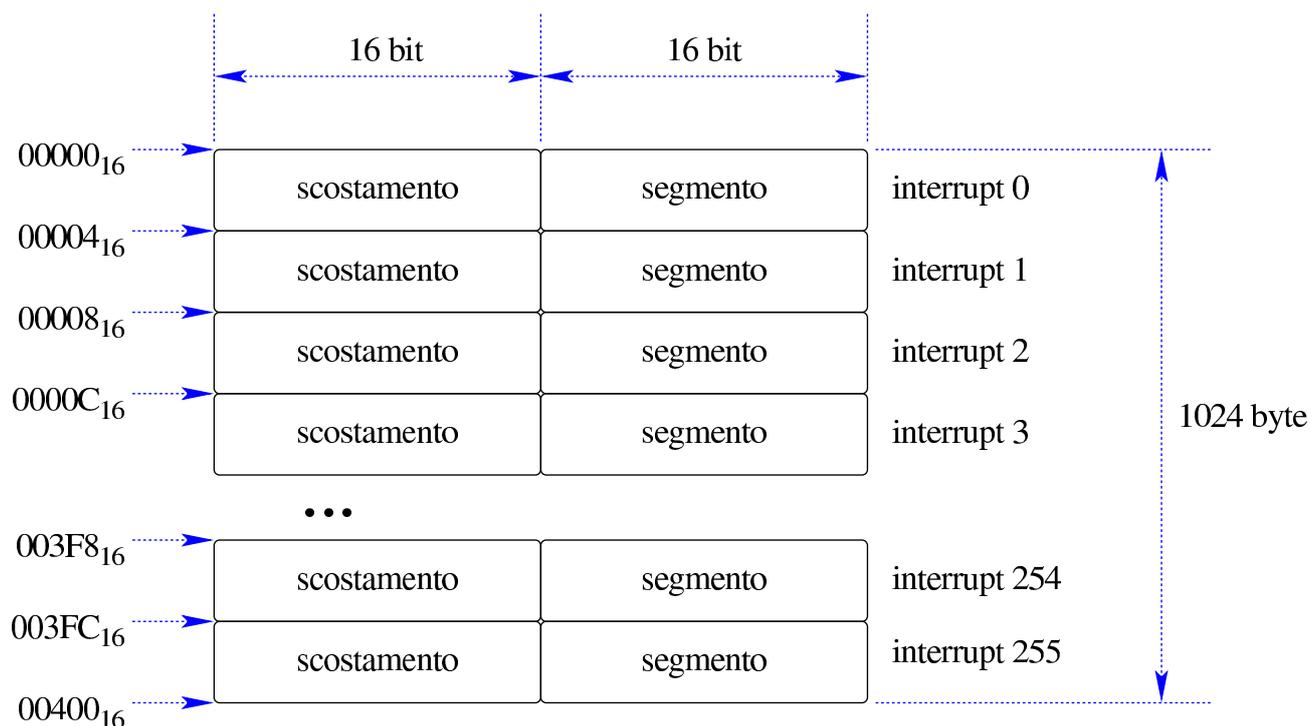
IVT: «interrupt vector table»

«

I microprocessori x86-16 utilizzano una tabella delle interruzioni collocata nella prima parte della memoria, a partire dall'indirizzo 0. La tabella è nota con la sigla IVT, ovvero *Interrupt vector table*.

Le voci della tabella IVT sono costituite semplicemente da due informazioni: l'indirizzo di segmento e lo scostamento in cui l'interruzione rispettiva viene gestita, attraverso del codice appropriato. Va osservato però che appare prima lo scostamento e poi il segmento.

Figura u140.2. Tabella IVT in memoria.



Supponendo sia  $n$  il numero di un'interruzione, si ottiene l'indirizzo della routine di interruzione con un calcolo molto semplice: lo scostamento è disponibile all'indirizzo  $n \times 4$ ; il segmento è disponibile all'indirizzo  $n \times 4 + 2$ .

Solo una porzione delle voci della tabella è utilizzata in modo predefinito, mentre le altre sono disponibili per interruzioni aggiuntive a uso del sistema operativo. In ogni caso, tutta la tabella può essere modificata, in modo da dirigere le interruzioni a codice alternativo.

Tabella u140.3. Configurazione iniziale della tabella IVT, secondo l'architettura IBM PC.

Interruzione	Descrizione
00 <sub>16</sub>	Eccezione del microprocessore: <i>divide error</i> .
01 <sub>16</sub>	Eccezione del microprocessore: <i>debug</i> .
02 <sub>16</sub>	Eccezione del microprocessore: <i>non maskable interrupt</i> .
03 <sub>16</sub>	Eccezione del microprocessore: <i>breakpoint</i> .

Interruzione	Descrizione
04 <sub>16</sub>	Eccezione del microprocessore: <i>arithmetic overflow</i> .
05 <sub>16</sub>	BIOS: il tasto [ <i>Stampa</i> ] o [ <i>Print_Screen</i> ] è stato premuto.
06 <sub>16</sub>	Eccezione del microprocessore: <i>invalid opcode</i> .
07 <sub>16</sub>	Eccezione del microprocessore: <i>no coprocessor</i> .
08 <sub>16</sub>	Interruzione hardware: IRQ 0, temporizzatore.
09 <sub>16</sub>	Interruzione hardware: IRQ 1, tastiera.
0A <sub>16</sub>	Interruzione hardware: IRQ 2.
0B <sub>16</sub>	Interruzione hardware: IRQ 3.
0C <sub>16</sub>	Interruzione hardware: IRQ 4.
0D <sub>16</sub>	Interruzione hardware: IRQ 5.
0E <sub>16</sub>	Interruzione hardware: IRQ 6.
0F <sub>16</sub>	Interruzione hardware: IRQ 7.
10 <sub>16</sub>	BIOS: video.
11 <sub>16</sub>	BIOS: controllo dell'apparecchiatura.
12 <sub>16</sub>	BIOS: memoria disponibile.
13 <sub>16</sub>	BIOS: unità a disco.
14 <sub>16</sub>	BIOS: porte seriali.
15 <sub>16</sub>	BIOS: funzioni varie.
16 <sub>16</sub>	BIOS: tastiera.
17 <sub>16</sub>	BIOS: porte parallele.
18 <sub>16</sub>	BIOS: interprete BASIC in ROM.
19 <sub>16</sub>	BIOS: riavvio.
1A <sub>16</sub>	BIOS: orologio.
1B <sub>16</sub>	BIOS: tasto [ <i>Interr</i> ] o [ <i>Break</i> ].
1C <sub>16</sub>	BIOS: temporizzatore.
1D <sub>16</sub>	BIOS: inizializzazione video.
1E <sub>16</sub>	BIOS: inizializzazione gestione dischi.
1F <sub>16</sub>	BIOS: caratteri del video.

Quando si costruisce una procedura da associare a una voce della tabella delle interruzioni, occorre considerare che prima che la procedura stessa sia raggiunta, il microprocessore inserisce nella pila delle informazioni. Nello specchio successivo si mettono a confronto le istruzioni relative alle chiamate di interruzioni e di procedure comuni:

int	call far	call
pushf push cs push ip	push cs push ip	push ip
iret	retf	ret
pop ip pop cs popf	pop ip pop cs	pop ip

## BIOS data area

Dopo la tabella IVT, a partire dall'indirizzo  $00400_{16}$  e fino a  $004FF_{16}$  incluso, si trova un'area di memoria utilizzata dal BIOS, per annodarvi delle informazioni. Alla fine del capitolo sono riportati i riferimenti alla documentazione che consente di interpretare il contenuto di questa area, ma quello che conta sapere è che non ci si deve scrivere, a meno di impedire alle funzioni del BIOS di operare correttamente.

## Altre aree di memoria

«

A partire dall'indirizzo  $A0000_{16}$ , fino a  $BFFFF_{16}$  incluso, si trova la memoria usata per rappresentare i dati sullo schermo. Successivamente ci sono altre aree di memoria in sola lettura (un'area precedente all'indirizzo  $F0000_{16}$  potrebbe essere priva di qualunque cosa) in particolare il blocco da  $F0000_{16}$  a  $FFFFFF_{16}$  che contiene le procedure del BIOS.

In pratica, a parte la possibilità di scrivere direttamente nella memoria video, per ottenere la rappresentazione del testo sullo schermo, la memoria da  $A0000_{16}$  fino alla fine, non può essere utilizzata, ma rimane incerta una porzione di memoria antecedente l'indirizzo  $A0000_{16}$  che potrebbe essere utilizzata anch'essa dalle procedure che compongono il BIOS.

Per conoscere l'ammontare di memoria libera si può leggere il valore contenuto all'indirizzo  $00413_{16}$ , nell'ambito della *BIOS data area*, tenendo conto che si tratta di un numero a 16 bit. Quel valore indica la quantità di memoria utile, espressa in multipli di 1024 byte, ma occorre considerare che si può utilizzare solo a partire dall'indirizzo  $00500_{16}$ , ovvero dalla fine della BDA. In alternativa, si può chiamare l'interruzione  $12_{16}$ , ottenendo dal registro *AX* tale valore.

## Interruzioni principali

«

Il BIOS di un elaboratore IBM PC offre una serie di funzionalità, attraverso delle interruzioni, le quali possono essere utilizzate in mancanza di funzioni più sofisticate del sistema operativo. Il testo di riferimento per le interruzioni, del BIOS e dei sistemi operativi che le estendono, è quello di Ralf Brown, annotato alla fine del capitolo.

Nelle sezioni successive vengono descritte solo alcune interruzioni offerte esclusivamente da BIOS standard.

### INT 10<sub>16</sub>/AH=00<sub>16</sub> «set video mode»

Definisce la modalità di funzionamento del video. A seconda della modalità scelta, si possono usare una o più «pagine» distinte.

Parametro	Descrizione
<b>AH</b>	00 <sub>16</sub>
<b>AL</b>	Modalità video: 00 <sub>16</sub> testo, 40×25 caratteri, 16 colori, 8 pagine; 03 <sub>16</sub> testo, 80×25 caratteri, 16 colori, 4 pagine.

### INT 10<sub>16</sub>/AH=05<sub>16</sub> «select active display page»

Seleziona la pagina video attiva. La numerazione parte da zero e la quantità di pagine disponibili dipende dalla modalità scelta.

Parametro	Descrizione
<b>AH</b>	05 <sub>16</sub>
<b>BH</b>	Pagina video da selezionare.

### INT 10<sub>16</sub>/AH=0E<sub>16</sub> «teletype output»

Mostra un carattere sullo schermo, alla pagina specificata, facendo avanzare il cursore e facendo scorrere il testo precedente se necessario. In questa modalità di visualizzazione, i caratteri con funzioni speciali vengono interpretati secondo la tradizione, tenendo conto che <CR> riporta il cursore all'inizio della stessa riga e che <LF> fa avanzare alla riga successiva, ma senza riportare il cursore all'inizio.

Parametro	Descrizione
<b>AH</b>	0E <sub>16</sub>
<b>AL</b>	Carattere da rappresentare.
<b>BL</b>	Numero della pagina video.

### INT 12<sub>16</sub> «get memory size»

«

Restituisce la dimensione della memoria utilizzabile, partendo dall'indirizzo 00000<sub>16</sub>, espressa in multipli di 1024 byte. La memoria utilizzabile effettivamente inizia solo a partire dall'indirizzo 00500<sub>16</sub>. Eventualmente lo stesso valore sarebbe accessibile all'indirizzo 00413<sub>16</sub>, leggendo un numero da 16 bit.

Valore restituito	Descrizione
<b>AX</b>	Dimensione della memoria disponibile.

### INT 13<sub>16</sub>/AH=00<sub>16</sub> «reset disk system»

«

Azzerare il sistema di gestione dei dischi, per l'unità indicata. L'unità è un numero da zero in su per i dischetti, mentre per i dischi fissi si parte da 80<sub>16</sub> in su.

Parametro	Descrizione
<b>AH</b>	00 <sub>16</sub>
<b>DL</b>	Numero dell'unità da azzerare.

Valore restituito	Descrizione
<b>AH</b>	Stato: zero indica un risultato soddisfacente, altrimenti si tratta di un errore.

Valore restituito	Descrizione
<i>c</i> ( <i>carry</i> )	Zero se <i>AH</i> è pari a zero; altrimenti, in presenza di un errore, l'indicatore viene attivato.

INT 13<sub>16</sub>/AH=02<sub>16</sub> «read disk sectors into memory»

Legge uno o più settori dal disco alla memoria. Il numero del cilindro in cui si trova il settore iniziale viene indicato utilizzando il registro *CH* per gli otto bit meno significativi, mentre si aggiungono altri due bit, più significativi, dal registro *CL*.

L'unità è un numero da zero in su per i dischetti, mentre per i dischi fissi si parte da 80<sub>16</sub> in su.

Parametro	Descrizione
<i>AH</i>	02 <sub>16</sub>
<i>DL</i>	Numero dell'unità da cui leggere.
<i>AL</i>	Quantità di settori da leggere (deve essere maggiore di zero).
<i>CH</i> , <i>CL</i> <sub>bit6</sub> , <i>CL</i> <sub>bit7</sub>	Numero del cilindro del primo settore da leggere, costituito dai bit 7 e 6 del registro <i>DL</i> e dal registro <i>CL</i> (da 0 a 1023).
<i>CL</i> <sub>bit0..</sub> <i>CL</i> <sub>bit5</sub>	Numero del settore, relativo alla traccia, intesa come combinazione di cilindro e testina (da 1 a 63).
<i>DH</i>	Numero della testina (da 0 a 63, perché i due bit più significativi potrebbero essere attribuiti ad altre funzioni).
<i>ES:BX</i> ( <i>ES*16+BX</i> )	Puntatore all'area di memoria che deve ricevere i settori letti.

Valore restituito	Descrizione
<b>AH</b>	Stato: zero indica un risultato soddisfacente, altrimenti si tratta di un errore.
<b>c (carry)</b>	Zero se <b>AH</b> è pari a zero; altrimenti, in presenza di un errore, l'indicatore viene attivato.
<b>AL</b>	In presenza di un errore, riporta la quantità di settori letti correttamente.

INT 13<sub>16</sub>/AH=03<sub>16</sub> «write disk sectors»

«

Scrive uno o più settori dalla memoria nel disco. Il numero del cilindro in cui si trova il settore iniziale viene indicato utilizzando il registro **CH** per gli otto bit meno significativi, mentre si aggiungono altri due bit, più significativi, dal registro **CL**.

L'unità è un numero da zero in su per i dischetti, mentre per i dischi fissi si parte da 80<sub>16</sub> in su.

Parametro	Descrizione
<b>AH</b>	03 <sub>16</sub>
<b>DL</b>	Numero dell'unità in cui scrivere.
<b>AL</b>	Quantità di settori da scrivere (deve essere maggiore di zero).
<b>CH</b> , <b>CL</b> <sub>bit6</sub> , <b>CL</b> <sub>bit7</sub>	Numero del cilindro del primo settore da scrivere, costituito dai bit 7 e 6 del registro <b>DL</b> e dal registro <b>CL</b> (da 0 a 1023).
<b>CL</b> <sub>bit0..CL</sub> <sub>bit5</sub>	Numero del settore, relativo alla traccia, intesa come combinazione di cilindro e testina (da 1 a 63).
<b>DH</b>	Numero della testina (da 0 a 63, perché i due bit più significativi potrebbero essere attribuiti ad altre funzioni).
<b>ES:BX</b> ( <b>ES*16+BX</b> )	Puntatore all'area di memoria da cui trarre i dati per la scrittura dei settori.

Valore restituito	Descrizione
<b>AH</b>	Stato: zero indica un risultato soddisfacente, altrimenti si tratta di un errore.
<b>c (carry)</b>	Zero se <b>AH</b> è pari a zero; altrimenti, in presenza di un errore, l'indicatore viene attivato.
<b>AL</b>	In presenza di un errore, riporta la quantità di settori scritti correttamente.

INT 16<sub>16</sub>/AH=00<sub>16</sub> «get keystroke from keyboard»

Legge un valore inserito dalla tastiera, eliminandolo dalla memoria tampone associata. <<

Parametro	Descrizione
<b>AH</b>	00 <sub>16</sub>

Valore restituito	Descrizione
<b>AH</b>	Il valore letto, secondo il codice usato dal BIOS.
<b>AL</b>	Il valore letto, tradotto in un carattere ASCII.

INT 16<sub>16</sub>/AH=01<sub>16</sub> «check for keystroke in the keyboard buffer»

Legge un valore inserito dalla tastiera, ma senza eliminarlo dalla memoria tampone associata. <<

Parametro	Descrizione
<b>AH</b>	01 <sub>16</sub>

Valore restituito	Descrizione
<i>z</i> ( <i>zero</i> )	Zero se la lettura è avvenuta con successo; altrimenti, se la memoria tampone è vuota, l'indicatore risulta attivato.
<i>AH</i>	Il valore letto, secondo il codice usato dal BIOS.
<i>AL</i>	Il valore letto, tradotto in un carattere ASCII.

## 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](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\\_2001fa0111/micro17.ppt](http://wayback.archive.org/web/20040101000000*/http://calab.kaist.ac.kr/~hyoon/courses/cs310_2001fa0111/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/>



# Strumenti di sviluppo e di utilizzo

Preparazione .....	3017
Bcc .....	3019
As86 .....	3021
Ld86 .....	3022
Bootblocks .....	3025
Riferimenti .....	3027

Per chi si avvale di un sistema operativo GNU, gli strumenti per sviluppare codice per x86-16 sono costituiti dalla raccolta nota con il nome Dev86, la quale mette assieme il compilatore C Bcc,<sup>1</sup> l'assemblatore As86 e il «collegatore» Ld86, oltre a una libreria C adatta per produrre applicazioni per ELKS (*Embeddable Linux kernel subset*).

Considerato che strumenti del genere sono utili, presumibilmente, per realizzare un programma autonomo (*stand alone*) o il kernel di un sistema operativo, un programma di avvio facilita molto il lavoro e consente di concentrare l'attenzione su ciò che si vuole realizzare veramente. Per questo motivo, nel capitolo viene anche preso in considerazione Bootblocks per l'avvio di un sistema operativo da dischetti con file system Minix 1.

## Preparazione

In una distribuzione GNU/Linux Debian sono disponibili i pacchetti **'bcc'**, **'bin86'** e **'elks-libc'** che forniscono il necessario per la

compilazione, ma in un altro sistema GNU può essere necessario procurarsi il pacchetto sorgente Dev86, dal quale si ottiene ciò che serve.

Se si è costretti a partire dai sorgenti di Dev86, una volta scaricato il pacchetto, questo può essere espanso in una directory qualunque nell'elaboratore GNU, come mostrato dall'esempio seguente, dove però, successivamente, si possano acquisire i privilegi dell'utente 'root':

```
$ tar xzvf Dev86src-0.16.17.tar.gz [Invio]
```

Si ottiene la directory 'dev86-0.16.17/' che si articola ulteriormente. Terminata l'installazione occorre compilare questi sorgenti e installarli. In questo caso si prevede di installare Dev86 a partire da '/opt/dev86/':

```
$ cd dev86-0.16.17 [Invio]
```

```
$ make PREFIX=/opt/dev86/ [Invio]
```

Viene richiesto di intervenire su alcuni indicatori (*flag*); in generale dovrebbe andare bene ciò che viene proposto in modo predefinito:

- 1) (ON) Library of bcc helper functions
- 2) (ON) Minimal syscalls for BIOS level
- 3) (ON) Unix error functions
- 4) (ON) Management for /etc/passwd /etc/group /etc/utmp
- 5) (OFF) Linux-i386 system call routines GCC
- 6) (ON) GNU termcap routines
- 7) (ON) Bcc 386 floating point
- 8) (ON) Linux-i386 system call routines
- 9) (ON) Example kernel include files and syscall.dat
- 10) (ON) Malloc routines
- 11) (ON) Various unix lib functions

- 12) (ON) Msdos system calls
- 13) (ON) Regular expression lib
- 14) (ON) Stdio package
- 15) (ON) String and memory manipulation
- 16) (ON) Linux-8086 system call routines
- 17) (ON) Termios functions
- 18) (ON) Unix time manipulation functions.

Select config option to flip [or quit] > **quit** [Invio]

Al termine della compilazione si passa all'installazione, cominciando dalla creazione della directory `‘/opt/dev86/’`. Per fare questo occorrono i privilegi dell'utente `‘root’`:

```
$ su [Invio]
```

...

```
# mkdir -p /opt/dev86 [Invio]
```

```
# make install [Invio]
```

## Bcc

Bcc<sup>2</sup> è un compilatore C tradizionale, ovvero fatto per la vecchia sintassi, nota con la sigla K&R. Tuttavia, con l'ausilio di un programma esterno (di norma si tratta di Unproto di Wietse Venema, incluso nella distribuzione Dev86), può compilare sorgenti scritti nella forma di un C standard, pur non potendo disporre di tutte le funzionalità di un compilatore aggiornato.

```
bcc [opzioni] file_c...
```

Tabella u141.2. Alcune opzioni per l'uso di Bcc.

Opzione	Descrizione
-ansi	Si avvale di un programma esterno per poter accettare un sorgente scritto secondo le convenzioni attuali del linguaggio C, pur nei limiti di quanto Bcc può poi elaborare.
-O	Produce un codice adatto per CPU 8086/8088.
-S	Produce un file in linguaggio assembler, da usare poi con l'assemblatore As86. In mancanza dell'opzione '-S' o '-c', si ottiene direttamente un file eseguibile, con l'intervento automatico di As86 e di Ld86.
-c	Produce un file oggetto, da utilizzare poi con il collegatore Ld86. In mancanza dell'opzione '-S' o '-c', si ottiene direttamente un file eseguibile, con l'intervento automatico di As86 e di Ld86.
-o <i>nome</i>	Produce un file con il nome specificato.
-I	Non utilizza i percorsi predefiniti per l'inclusione dei file di intestazione.
-I <i>percorso</i>	L'opzione '-I', a cui si attacca un percorso, aggiunge quel percorso a quelli usati per l'inclusione dei file di intestazione. Possono essere specificati più percorsi ripetendo l'uso dell'opzione.

L'esempio seguente mostra la compilazione del file 'mio.c', per produrre il file 'mio.s', contenente il codice in linguaggio assembler. Per la compilazione, i file di intestazione vengono cercati esclusivamente in percorsi stabiliti: './include' e './..

/include’.

```
$ bcc -ansi -O -S -o mio.s ←  
↪ -I -I../include -I../../include mio.c [Invio]
```

L’esempio successivo è simile, ma si produce il file oggetto ‘mio.o’:

```
$ bcc -ansi -O -c -o mio.s ←  
↪ -I -I../include -I../../include mio.c [Invio]
```

I nomi delle variabili e delle funzioni scritte in linguaggio C, si traducono nel linguaggio assembler in nomi preceduti dal trattino basso. Per esempio, la funzione *main()*, diventa il simbolo ‘**\_main**’. Per questa ragione, quando si scrivono porzioni di codice in linguaggio assembler da esportare, occorre ricordare di aggiungere un trattino basso all’inizio.

A meno di voler produrre programmi per il sistema operativo ELKS, il compilatore Bcc va utilizzato con l’opzione ‘**-c**’, oppure ‘**-S**’, per poter controllare i passaggi successivi, in particolare la fase di collegamento dei vari componenti.

## As86

As86 è un assembler in linguaggio x86, adatto alla compilazione di quanto prodotto da Bcc. La sintassi usata da As86 è fondamentalmente quella Intel. «

```
as86 [opzioni] file_s
```

Tabella u141.3. Alcune opzioni per l'uso di As86.

Opzione	Descrizione
-0	Compila nella modalità a 16 bit e avvisa quando incontra istruzioni che non sono adatte a CPU 8086/8088.
-o <i>nome</i>	Produce un file oggetto con il nome specificato.
-s <i>nome</i>	Produce un file di testo contenente l'elenco dei simboli individuati. Questa opzione può essere usata assieme a '-o'.
-u	Fa in modo che i simboli privi di una dichiarazione siano importati dall'esterno senza specificare il segmento.

L'esempio seguente rappresenta una situazione di utilizzo comune, in cui si produce il file oggetto 'mio.o', a partire dal sorgente 'mio.s':

```
$ as86 -u -0 -o mio.o mio.s [Invio]
```

Come precisato a proposito di Bcc, se si devono importare dei simboli dal codice C, occorre aggiungere un trattino basso all'inizio dei nomi.

## Ld86

«

Ld86 è il «collegatore» (*linker*) associato a As86. La caratteristica di Ld86 è quella di poter produrre un eseguibile «impuro» (come viene definito nella sua pagina di manuale), per il quale il segmento usato dal codice è lo stesso usato per i dati ( $CS==DS==SS$ ), oppure può tenere separati il codice e i dati in segmenti distinti (ma in tal caso vale ancora l'uguaglianza  $DS==SS$ ).

Nella pagina di manuale di Ld86 si parla di «I&D», ovvero di istruzioni e dati, che possono essere separati o meno.

```
ld86 [opzioni] file_o...
```

Tabella u141.4. Alcune opzioni per l'uso di Ld86.

Opzione	Descrizione
-d	Elimina l'intestazione dal file che va a essere generato. L'utilizzo di questa opzione implica l'uso di '-s'.
-s	Elimina i simboli.
-o <i>nome</i>	Produce un file eseguibile con il nome specificato.
-i	Tiene separati il segmento usato dal codice rispetto a quello dei dati. In mancanza di questa opzione, il segmento è lo stesso.

L'esempio seguente mostra la creazione di un programma, privo di intestazione e di simboli, dove tutto viene così definito attraverso il codice in modo esplicito. In particolare, si presume che il file 'crt0.o' sia realizzato in modo da definire esattamente la forma della prima parte del file eseguibile.

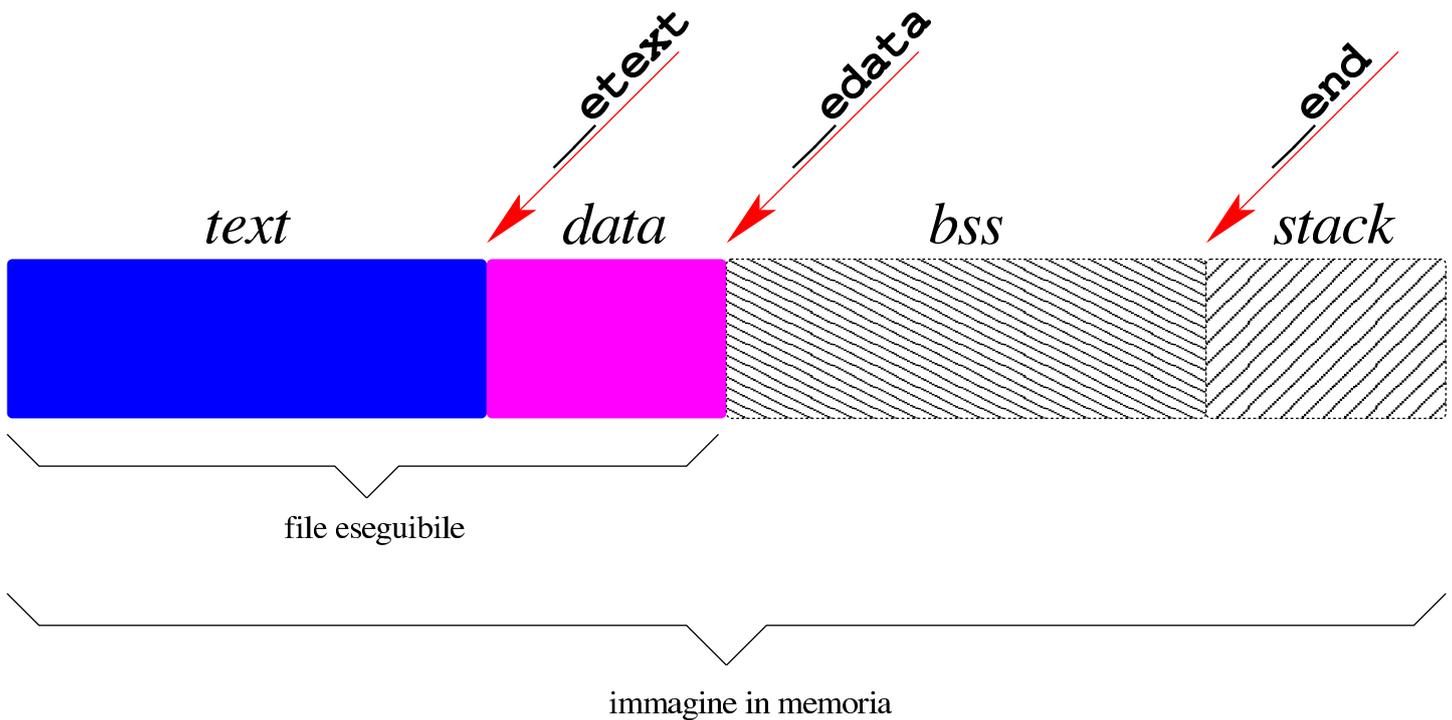
```
§ ld86 -d -s -o kimage crt0.o body.o main.o [Invio]
```

Ld86 definisce implicitamente dei simboli, raggiungibili dal codice che si scrive. Tra questi, sono molto importanti quelli seguenti, con cui è possibile determinare la collocazione in memoria del programma, distinguendo tra codice e dati:

Simbolo per As86	Nome per Bcc	Descrizione
<code>__etext</code>	<code>_etext</code>	Variabile a 16 bit contenente l'indirizzo conclusivo dell'area usata dal codice, nell'ambito del segmento in cui si colloca.
<code>__edata</code>	<code>_edata</code>	Variabile a 16 bit contenente l'indirizzo conclusivo dell'area usata dai dati inizializzati, nell'ambito del segmento in cui si colloca.
<code>__end</code>	<code>_end</code>	Variabile a 16 bit contenente l'indirizzo conclusivo dell'area usata dai dati non inizializzati (BSS), esclusa la pila, nell'ambito del segmento in cui si colloca.
<code>__segoff</code>	<code>_segoff</code>	Variabile a 16 bit contenente la distanza tra l'inizio del segmento codice e l'inizio di quello usato per i dati, espressa in multipli di 16 bit. In presenza di eseguibili in cui il segmento è lo stesso, questo valore è zero.

In generale, la struttura di un file eseguibile prodotto da Ld86 è composta inizialmente dal codice, quindi continua con i dati inizializzati. Lo spazio dei dati non inizializzati non fa parte del file e deve essere previsto quando si carica il file in memoria, per metterlo in esecuzione; inoltre, lo stesso va fatto per la pila (*stack*) dei dati, la quale deve collocarsi dopo tale area.

Figura u141.6. Contenuto di un programma privo di intestazione e di simboli.



## Bootblocks

Il pacchetto Bootblocks<sup>3</sup> consente di avviare un sistema per elaboratori x86-16, contenuto in un dischetto o in una partizione del disco fisso, con il kernel inserito nello stesso file system. Il pacchetto viene distribuito assieme agli strumenti di sviluppo Dev86, ma non viene compilato automaticamente assieme a quelli. Si trova precisamente nella sottodirectory 'bootblocks/' dei sorgenti di Dev86. Si compila in modo molto semplice con il comando 'make':

```
# cd sorgenti_dev86/bootblocks [Invio]
```

```
# make [Invio]
```

Dalla compilazione si ottengono diversi file e sono utili in particolare:

File	Descrizione
<code>'makeboot'</code>	programma per l'installazione del settore di avvio, da usare attraverso un sistema GNU/Linux comune;
<code>'makeboot.com'</code>	programma analogo a <code>'makeboot'</code> , da usare con un sistema Dos.

Per avviare un programma autonomo o un kernel vero e proprio, in un dischetto con file system Minix 1, è sufficiente copiare tale file in modo che si trovi nella directory principale con il nome `'boot'`, oppure si crea la directory `'/boot/'` e vi si colloca il file con il nome che si preferisce.

Supponendo di utilizzare un sistema GNU/Linux, supponendo di avere preparato il dischetto Minix (con i nomi al massimo di 14 byte) contenente tutto quello che serve, soprattutto con la directory `'/boot/'` o con il file `'boot'`, se questo dischetto risulta inserito nell'unità corrispondente al file di dispositivo `'/dev/fd0'`, **senza essere stato innestato**, si può eseguire il comando seguente, tenendo conto che il programma `'makeboot'` si presume collocato in una directory prevista tra i vari percorsi della variabile di ambiente **'PATH'**:

```
# makeboot minix /dev/fd0 [Invio]
```

```
Wrote sector 0
```

```
Wrote sector 1
```

Se il programma si accorge che il settore di avvio del dischetto contiene già qualcosa, si rifiuta di procedere, a meno di usare l'opzione **'-f'**:

```
# makeboot -f minix /dev/fd0 [Invio]
```

```
Boot block isn't empty, zap it first
Wrote sector 0
Wrote sector 1
```

È importante sapere che questo programma di avvio colloca in memoria il programma o il kernel da avviare a partire dall'indirizzo efficace  $10000_{16}$ . Pertanto, dopo l'avvio effettivo, rimane inutilizzato lo spazio di memoria da  $00500_{16}$   $0FFFF_{16}$ .

## 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](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://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/>

<sup>1</sup> **Bcc, As86, Ld86** GNU GPL

<sup>2</sup> Va specificato che si tratta del compilatore Bcc di Bruce Evans, perché con questo nome o con questa sigla si trovano più facilmente riferimenti a compilatori C diversi, per esempio quello di Borland.

<sup>3</sup> **Bootblocks** GNU GPL