

Microprocessori x86-16	1285
Segmenti	1285
Registri	1286
Trasferimento di dati tra due segmenti differenti	1288
Riferimenti a indirizzi di memoria con i registri	1288
Convenzioni di chiamata	1289
Sintesi delle istruzioni x86-16	1290
Sostituzione delle istruzioni per i186	1305
Riferimenti	1306
Architettura IBM PC	1307
IVT: «interrupt vector table»	1307
BIOS data area	1308
Altre aree di memoria	1309
Interruzioni principali	1309
Riferimenti	1311
Strumenti di sviluppo e di utilizzo	1313
Preparazione	1313
Bcc	1314
As86	1315
Ld86	1315
Bootblocks	1316
Riferimenti	1317

Segmenti	1285
Registri	1286
Trasferimento di dati tra due segmenti differenti	1288
Riferimenti a indirizzi di memoria con i registri	1288
Convenzioni di chiamata	1289
Sintesi delle istruzioni x86-16	1290
Sostituzione delle istruzioni per il 86	1305
Riferimenti	1306

ADC 1296 ADD 1296 AH 1286 AL 1286 AND 1297 AX 1286 BH 1286 BL 1286 BP 1286 BX 1286 CALL 1298 CALL FAR 1298 CBW 1290 CH 1286 CL 1286 CLC 1301 CLD 1301 CLI 1301 CMC 1301 CMP 1301 CMPSB 1293 CMPSW 1293 CWD 1290 CX 1286 DEC 1296 DH 1286 DI 1286 DIV 1296 DL 1286 DX 1286 ENTER 1298 FLAGS 1286 HLT 1301 IDIV 1296 IMUL 1296 IN 1305 1305 INC 1296 INT 1301 INTO 1301 IP 1286 IRET 1301 JA 1302 JAE 1302 JB 1302 JBE 1302 JC 1302 JCXZ 1302 JE 1302 JG 1302 JGE 1302 JL 1302 JLE 1302 JMP 1302 JMP FAR 1302 JNA 1302 JNAE 1302 JNB 1302 JNBE 1302 JNC 1302 JNE 1302 JNG 1302 JNGE 1302 JNL 1302 JNO 1302 JNP 1302 JNS 1302 JNZ 1302 JO 1302 JP 1302 JPE 1302 JPO 1302 JS 1302 JZ 1302 LAHF 1290 LDS 1290 LEA 1290 LEAVE 1298 LES 1290 LODSB 1292 LODSW 1292 LOOP 1305 LOOPE 1305 LOOPNE 1305 LOOPNZ 1305 LOOPZ 1305 MOV 1290 MOVSB 1292 MOVSW 1292 MUL 1296 NEG 1296 NOP 1290 NOT 1297 OR 1297 OUT 1305 1305 POP 1298 POPA 1298 POPF 1298 PUSH 1298 PUSHA 1298 PUSHF 1298 RCL 1297 RCR 1297 REP 1292 REPE 1293 REPNE 1293 REPNZ 1293 REPZ 1293 RET 1298 RETF 1298 RET FAR 1298 ROL 1297 ROR 1297 SAHF 1290 SAL 1297 SAR 1297 SBB 1296 SCASB 1293 SCASW 1293 SHL 1297 SHR 1297 SI 1286 SP 1286 STC 1301 1301 STI 1301 STOSB 1292 STOSW 1292 SUB 1296 TEST 1301 XCHG 1290 XLATB 1292 XOR 1297

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

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          ; Azzerare 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:

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

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]	$DS \cdot 16 + SI + \text{imm}$
[DI]	$DS \cdot 16 + DI$	<i>imm</i> [DI]	$DS \cdot 16 + DI + \text{imm}$
[BP]	$DS \cdot 16 + BP$	<i>imm</i> [BP]	$DS \cdot 16 + BP + \text{imm}$
[BX]	$DS \cdot 16 + BX$	<i>imm</i> [BX]	$DS \cdot 16 + BX + \text{imm}$
[BX+SI]	$DS \cdot 16 + BX + SI$	<i>imm</i> [BX+SI]	$DS \cdot 16 + BX + SI + \text{imm}$
[BX+DI]	$DS \cdot 16 + BX + DI$	<i>imm</i> [BX+DI]	$DS \cdot 16 + BX + DI + \text{imm}$
[BP+SI]	$DS \cdot 16 + BP + SI$	<i>imm</i> [BP+SI]	$DS \cdot 16 + BP + SI + \text{imm}$
[BP+DI]	$DS \cdot 16 + BP + DI$	<i>imm</i> [BP+DI]	$DS \cdot 16 + BP + DI + \text{imm}$

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 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 «!» 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
NOF		<i>not operate</i> Istruzione nulla.	cpazstido
MOV	reg, reg reg, mem reg, imm mem, reg mem, imm	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	reg, mem	<i>load effective address</i> Mette nel registro l'indirizzo della memoria, inteso come scostamento dall'inizio del segmento dati. <i>dst := &org</i>	cpazstido
LDS	reg, mem	<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 DS (segmento dati). <i>DS:dst := org</i>	cpazstido

Nome	Operandi: <i>dst, org1, org2</i>	Descrizione	Indicatori
LES	reg, mem	<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 ES . <i>ES:dst := org</i>	cpazstido
XCHG	reg, reg reg, mem mem, reg	<i>exchange data</i> Scambia i valori. <i>dst := org</i>	cpazstido
CBW		<i>convert byte to word</i> 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. <i>AX := AL</i>	cpazstido
CWD		<i>convert word to double word</i> 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 := FFFF₁₆	cpazstido
LAHF		<i>load flags into AH</i> Carica i primi otto indicatori in AH , escludendo quelli riservati. <i>AH_{bit0} := c</i> <i>AH_{bit1} := 1</i> <i>AH_{bit2} := p</i> <i>AH_{bit3} := 0</i> <i>AH_{bit4} := a</i> <i>AH_{bit5} := 0</i> <i>AH_{bit6} := z</i> <i>AH_{bit7} := s</i>	cpazstido #####
SAHF		<i>store AH into flags</i> 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 . <i>c := AH_{bit0}</i> <i>p := AH_{bit2}</i> <i>a := AH_{bit4}</i> <i>z := AH_{bit6}</i> <i>s := AH_{bit7}</i>	cpazstido #####

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 DS:SI ($DS \cdot 16 + SI$), 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.</p> <p>$AL := *(DS \cdot 16 + SI)$</p> <p>IF $d == 0$ THEN SI++ ELSE SI--</p>	cpazstidot.
LODSW		<p><i>load string word</i></p> <p>Dall'indirizzo a cui punta la coppia DS:SI ($DS \cdot 16 + SI$), 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.</p> <p>$AL := *(DS \cdot 16 + SI)$</p> <p>IF $d == 0$ THEN SI += 2 ELSE SI -= 2</p>	cpazstidot.
STOSB		<p><i>store string byte</i></p> <p>All'indirizzo a cui punta la coppia ES:DI ($ES \cdot 16 + DI$), viene scritto il valore contenuto in AL, aggiornando DI in base al contenuto dell'indicatore di direzione.</p> <p>$*(ES \cdot 16 + DI) := AL$</p> <p>IF $d == 0$ THEN DI++ ELSE DI--</p>	cpazstidot.
STOSW		<p><i>store string word</i></p> <p>All'indirizzo a cui punta la coppia ES:DI ($ES \cdot 16 + DI$), viene scritto il valore contenuto in AX, aggiornando DI in base al contenuto dell'indicatore di direzione.</p> <p>$*(ES \cdot 16 + DI) := AX$</p> <p>IF $d == 0$ THEN DI += 2 ELSE DI -= 2</p>	cpazstidot.

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 DS:SI ($DS \cdot 16 + SI$), all'indirizzo a cui punta la coppia ES:DI ($ES \cdot 16 + DI$), aggiornando SI e DI in base al valore dell'indicatore di direzione.</p> <p>$*(ES \cdot 16 + DI) := *(DS \cdot 16 + SI)$</p> <p>IF $d == 0$ THEN SI++ DI++ ELSE SI-- DI--</p>	cpazstidot.
MOVSW		<p><i>move string word</i></p> <p>Copia un blocco di 16 bit, dall'indirizzo a cui punta la coppia DS:SI ($DS \cdot 16 + SI$), all'indirizzo a cui punta la coppia ES:DI ($ES \cdot 16 + DI$), aggiornando SI e DI in base al valore dell'indicatore di direzione.</p> <p>$*(ES \cdot 16 + DI) := *(DS \cdot 16 + SI)$</p> <p>IF $d == 0$ THEN SI += 2 DI += 2 ELSE SI -= 2 DI -= 2</p>	cpazstidot.
REP		<p><i>repeat</i></p> <p>Ripete l'istruzione successiva (che può essere una tra: 'LODSB', 'LODSW', 'STOSB', 'STOSW', 'MOVSB', 'MOVSW'), per CX volte.</p> <p>IF $CX != 0$ THEN istruzione successiva CX-- ELSE break</p>	cpazstido
XLATB		<p><i>translate table to byte</i></p> <p>Assegna a AL il valore che si può raggiungere all'indirizzo composto da DS:BX+AL ($DS \cdot 16 + BX + AL$), dove AL va inteso come valore senza segno.</p> <p>$AL := *(DS \cdot 16 + BX + AL)$</p>	cpazstido

Tabella u139.18. Confronti con la memoria.

Nome	Operandi: <i>dst, org1, org2</i>	Descrizione	Indicatori
SCASB		<p><i>compare string byte</i> 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</p> <pre> IF d==0 THEN DI++ ELSE DI-- </pre>	cpazstidot. #####...#
SCASW		<p><i>compare string word</i> 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</p> <pre> IF d==0 THEN DI:+=2 ELSE DI:-=2 </pre>	cpazstidot. #####...#
CMPSB		<p><i>compare string byte in memory</i> 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)</p> <pre> IF d==0 THEN SI++ DI++ ELSE SI++ DI-- </pre>	cpazstidot. #####...#

Nome	Operandi: <i>dst, org1, org2</i>	Descrizione	Indicatori
CMPSW		<p><i>compare string word in memory</i> 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)</p> <pre> IF d==0 THEN SI++ DI++ ELSE SI++ DI-- </pre>	cpazstidot. #####...#
REPE REPZ		<p><i>repeat while equal</i> <i>repeat while zero</i> Ripete l'istruzione successiva (che può essere una tra: 'SCASB', 'SCASW', 'CMPB', 'CMPW'), 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.</p> <pre> IF CX!=0 THEN istruzione successiva CX-- IF z=1 THEN continue ELSE break ELSE break </pre>	cpazstido ...#.....

Nome	Operandi: <i>dst, org1, org2</i>	Descrizione	Indicatori
REPNE REPZ		<p><i>repeat while not equal</i> <i>repeat while not zero</i> 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 <i>CX</i> volte.</p> <p>IF <i>CX</i>!=0 THEN istruzione successiva <i>CX</i>-- IF <i>z</i>==0 THEN continue ELSE break ELSE break</p>	cpazstido ...#.....

Tabella u139.19. Operazioni aritmetiche.

Nome	Operandi: <i>dst, org1, org2</i>	Descrizione	Indicatori
NEG	<i>reg mem</i>	<p><i>negation</i> Inverte il segno di un numero, attraverso il complemento a due.</p> <p><i>operand := -operand</i></p>	cpazstido ##.##...#
ADD	<i>reg, reg reg, mem reg, imm mem, reg mem, imm</i>	<p><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>).</p> <p><i>dst := org + dst</i></p>	cpazstido ##.##...#
SUB	<i>reg, reg reg, mem reg, imm mem, reg mem, imm</i>	<p><i>subtraction</i> Sottrazione di interi con o senza segno, ignorando il riporto precedente.</p> <p><i>dst := org - dst</i></p>	cpazstido ##.##...#
ADC	<i>reg, reg reg, mem reg, imm mem, reg mem, imm</i>	<p><i>addition with carry</i> Somma di interi, con o senza segno, aggiungendo anche il riporto precedente (l'indicatore <i>carry</i>).</p> <p><i>dst := org + dst + c</i></p>	cpazstido t..... ##.##...#
SBB	<i>reg, reg reg, mem reg, imm mem, reg mem, imm</i>	<p><i>subtraction with borrow</i> Sottrazione di interi, con o senza segno, tenendo conto del «prestito» precedente (l'indicatore <i>carry</i>).</p> <p><i>dst := org + dst - c</i></p>	cpazstido t..... ##.##...#

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

Tabella u139.20. Operazioni logiche.

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

Tabella u139.21. Scorrimenti e rotazioni.

Nome	Operandi: <i>dst, org1, org2</i>	Descrizione	Indicatori
SHL SHR	reg, 1 mem, 1 reg mem	<i>shift left shift right</i> 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	<i>shift arithmetically left 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 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	<i>rotate left with carry 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 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	<i>rotate left rotate right</i> 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: <i>dst, org1, org2</i>	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. <i>push indirizzo_successivo</i> IP := operand	cpazstido

Nome	Operandi: <i>dst, org1, org2</i>	Descrizione	Indicatori
CALL FAR	imm:imm	<i>call procedure</i> 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. <i>push CS</i> <i>push indirizzo_successivo</i> CS:IP := operand	cpazstido
RET		<i>return from call</i> Estrae dalla pila l'indirizzo dell'istruzione da raggiungere (IP) e salta a quella (serve a concludere una chiamata eseguita con 'CALL'). <i>pop IP</i>	cpazstido
RETF RET FAR		<i>return from far call</i> 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'). <i>pop IP</i> <i>pop CS</i>	cpazstido
PUSH	reg mem	<i>push data onto stack</i> Inserisce nella pila il valore (della dimensione di un registro comune). SP: -=2 *(SS*16+SP) := operand	cpazstido
POP	reg mem	<i>pop data from stack</i> Estrae dalla pila l'ultimo valore inserito (della dimensione di un registro comune). operand := *(SS*16+SP) SP: +=2	cpazstido
PUSHF		<i>push flags onto stack</i> Inserisce nella pila l'insieme del registro degli indicatori (FLAGS). <i>push FLAGS</i>	cpazstido
POPF		<i>pop flags from stack</i> Estrae dalla pila l'insieme del registro degli indicatori (FLAGS), aggiornando di conseguenza il registro stesso. <i>pop FLAGS</i>	cpazstido ????????

Nome	Operandi: <i>dst, org1, org2</i>	Descrizione	Indicatori
ENTER	<i>imm8, 0</i>	<i>enter stack frame</i> 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. <i>push BP</i> <i>BP:=SP</i> <i>SP:=-2*dst</i>	<i>cpazstido</i>
PUSHA		<i>push all registers onto stack</i> Questa funzione esiste a partire dai microprocessori i186. Inserisce nella pila i registri principali: AX, CX, DX, BX, SP, BP, SI, DI . <i>push AX</i> <i>push CX</i> <i>push DX</i> <i>push BX</i> <i>push SP</i> <i>push BP</i> <i>push SI</i> <i>push DI</i>	<i>cpazstido</i>
POPA		<i>pop all registers from stack</i> 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. <i>pop DI</i> <i>pop SI</i> <i>pop BP</i> <i>SP:+=2</i> <i>pop BX</i> <i>pop DX</i> <i>pop CX</i> <i>pop AX</i>	<i>cpazstido</i>
LEAVE		<i>leave stack frame</i> 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' . <i>SP:=BP</i> <i>pop BP</i>	<i>cpazstido</i>

Tabella u139.23. Interruzioni.

Nome	Operandi: <i>dst, org1, org2</i>	Descrizione	Indicatori
INT	<i>imm8</i>	<i>call to interrupt</i> Esegue una chiamata attraverso un'interruzione. Prima di saltare alla codice relativo all'interruzione selezionata, inserisce nella pila FLAGS, CS e IP . Azzerata anche l'indicatore IF (interrupt flag) , mentre gli altri indicatori rimangono inalterati. <i>pushf</i> <i>push CS</i> <i>push IP</i> <i>i:=0</i> <i>jmp far 0:(operand*4)</i>	<i>cpazstido</i>
IRET		<i>return from interrupt</i> 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. <i>pop IP</i> <i>pop CS</i> <i>popf</i>	<i>cpazstido</i> ##.##.##
CLI		<i>clear interrupt flag</i> Azzerata l'indicatore di abilitazione delle interruzioni (interrupt flag), disabilitando di conseguenza le interruzioni hardware. <i>i:=0</i>	<i>cpazstido</i>0..
STI		<i>set interrupt flag</i> Attiva l'indicatore di abilitazione delle interruzioni (interrupt flag), abilitando di conseguenza le interruzioni hardware. <i>i := 1</i>	<i>cpazstido</i>1..
HLT		<i>enter halt state</i> Ferma il sistema, fino a quando viene ricevuta un'interruzione hardware.	<i>cpazstido</i>
INTO		<i>interrupt if overflow</i> Se l'indicatore di straripamento è attivo, esegue la chiamata dell'interruzione numero 4 (la quale dovrebbe gestire il problema).	<i>cpazstido</i>t

Tabella u139.24. Indicatori e confronti tra registri.

Nome	Operandi: <i>dst, org1, org2</i>	Descrizione	Indicatori
CLC		<i>clear carry flag</i> Azzerata l'indicatore del riporto (carry), senza intervenire negli altri indicatori. <i>c:=0</i>	<i>cpazstido</i> 0.....

Nome	Operandi: <i>dst, org1, org2</i>	Descrizione	Indicatori
CLD		<i>clear direction flag</i> Azzerà l'indicatore di direzione (<i>direction</i>), senza intervenire negli altri indicatori. <i>d:=0</i>	cpazstido0.
STC		<i>set carry flag</i> Attiva l'indicatore di riporto (<i>carry</i>). <i>c:=1</i>	cpazstido 1.....
STD		<i>set direction flag</i> Attiva l'indicatore di direzione (<i>direction</i>). <i>d:=1</i>	cpazstido1.
CMC		<i>complement carry flag</i> Inverte il valore dell'indicatore del riporto (<i>carry</i>). <i>#.....</i>	cpazstido #.....
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 (CS). <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 JNB	<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 > org</i> THEN <i>go to imm</i>	cpazstido t..t.....

Nome	Operandi: <i>dst, org1, org2</i>	Descrizione	Indicatori
JAE JNB	<i>imm</i>	<i>conditional jump</i> 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 <i>dst, org</i> IF <i>dst >= org</i> THEN <i>go to imm</i>	cpazstido t..t.....
JB JNAE	<i>imm</i>	<i>conditional jump</i> 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 <i>dst, org</i> IF <i>dst < org</i> THEN <i>go to imm</i>	cpazstido t..t.....
JBE JNA	<i>imm</i>	<i>conditional jump</i> 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 <i>dst, org</i> IF <i>dst <= org</i> THEN <i>go to imm</i>	cpazstido t..t.....
JE	<i>imm</i>	<i>conditional jump</i> 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 <i>dst, org</i> IF <i>dst == org</i> THEN <i>go to imm</i>	cpazstido ...t.....
JNE	<i>imm</i>	<i>conditional jump</i> 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 <i>dst, org</i> IF <i>dst != org</i> THEN <i>go to imm</i>	cpazstido ...t.....
JG JNLE	<i>imm</i>	<i>conditional jump</i> Dopo un confronto con 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 > org</i> THEN <i>go to imm</i>	cpazstido ...tt...t

Nome	Operandi: <i>dst, org1, org2</i>	Descrizione	Indicatori
JGE JNL	<i>imm</i>	<i>conditional jump</i> 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 <i>dst, org</i> IF <i>dst >= org</i> THEN go to <i>imm</i>	cpazstido ...tt...t
JL JNGE	<i>imm</i>	<i>conditional jump</i> Dopo un confronto con segno, salta se la destinazione era minore dell'origine. Il salto riguarda solo l'ambito del segmento codice attuale. CMP <i>dst, org</i> IF <i>dst < org</i> THEN go to <i>imm</i>	cpazstido ...tt...t
JLE JNG	<i>imm</i>	<i>conditional jump</i> 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 <i>dst, org</i> IF <i>dst <= org</i> THEN go to <i>imm</i>	cpazstido ...tt...t
JC JNC	<i>imm</i>	<i>conditional jump</i> 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.	cpazstido t.....
JO JNO	<i>imm</i>	<i>conditional jump</i> Salta se l'indicatore di traboccamento (<i>overflow</i>), rispettivamente, è attivo, oppure non è attivo.	cpazstidot
JS JNS	<i>imm</i>	<i>conditional jump</i> Salta se l'indicatore di segno (<i>sign</i>), rispettivamente, è attivo, oppure non è attivo.	cpazstido ...t...
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 <i>CX</i> è pari a zero.	cpazstido

Tabella u139.26. Iterazioni

Nome	Operandi: <i>dst, org1, org2</i>	Descrizione	Indicatori
LOOP	<i>imm8</i>	<i>loop</i> Senza alterare gli indicatori, decrementa di una unità il registro ' <i>CX</i> ', 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	<i>imm8</i>	<i>conditional loop</i> Senza alterare gli indicatori, decrementa di una unità il registro ' <i>CX</i> ', 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	<i>imm8</i>	<i>conditional loop</i> Senza alterare gli indicatori, decrementa di una unità il registro ' <i>CX</i> ', 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
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
- *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_2001fa011/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.debatch/>
<http://homepage.ntlworld.com/robert.debatch/dev86/>

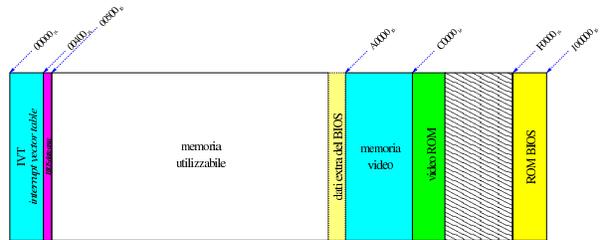
Architettura IBM PC

- IVT: «interrupt vector table» 1307
- BIOS data area 1308
- Altre aree di memoria 1309
- Interruzioni principali 1309
 - INT 10₁₆/AH=00₁₆ «set video mode» 1309
 - INT 10₁₆/AH=05₁₆ «select active display page» 1309
 - INT 10₁₆/AH=0E₁₆ «teletype output» 1309
 - INT 12₁₆ «get memory size» 1310
 - INT 13₁₆/AH=00₁₆ «reset disk system» 1310
 - INT 13₁₆/AH=02₁₆ «read disk sectors into memory» 1310
 - INT 13₁₆/AH=03₁₆ «write disk sectors» 1310
 - INT 16₁₆/AH=00₁₆ «get keystroke from keyboard» 1311
 - INT 16₁₆/AH=01₁₆ «check for keystroke in the keyboard buffer» 1311
- Riferimenti 1311

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.

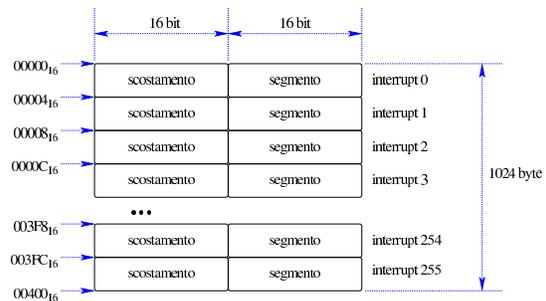


IVT: «interrupt vector table»

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 ₁₆	Eccezione del microprocessore: <i>divide error</i> .
01 ₁₆	Eccezione del microprocessore: <i>debug</i> .
02 ₁₆	Eccezione del microprocessore: <i>non maskable interrupt</i> .
03 ₁₆	Eccezione del microprocessore: <i>breakpoint</i> .
04 ₁₆	Eccezione del microprocessore: <i>arithmetic overflow</i> . BIOS: il tasto [Stampa] o [Print_Screen] è stato premuto.
05 ₁₆	Eccezione del microprocessore: <i>invalid opcode</i> .
06 ₁₆	Eccezione del microprocessore: <i>no coprocessor</i> .
07 ₁₆	Eccezione del microprocessore: <i>no coprocessor</i> .
08 ₁₆	Interruzione hardware: IRQ 0, temporizzatore.
09 ₁₆	Interruzione hardware: IRQ 1, tastiera.
0A ₁₆	Interruzione hardware: IRQ 2.
0B ₁₆	Interruzione hardware: IRQ 3.
0C ₁₆	Interruzione hardware: IRQ 4.
0D ₁₆	Interruzione hardware: IRQ 5.
0E ₁₆	Interruzione hardware: IRQ 6.
0F ₁₆	Interruzione hardware: IRQ 7.
10 ₁₆	BIOS: video.
11 ₁₆	BIOS: controllo dell'apparecchiatura.
12 ₁₆	BIOS: memoria disponibile.
13 ₁₆	BIOS: unità a disco.
14 ₁₆	BIOS: porte seriali.
15 ₁₆	BIOS: funzioni varie.
16 ₁₆	BIOS: tastiera.
17 ₁₆	BIOS: porte parallele.
18 ₁₆	BIOS: interprete BASIC in ROM.
19 ₁₆	BIOS: riavvio.
1A ₁₆	BIOS: orologio.
1B ₁₆	BIOS: tasto [Interr] o [Break].
1C ₁₆	BIOS: temporizzatore.
1D ₁₆	BIOS: inizializzazione video.
1E ₁₆	BIOS: inizializzazione gestione dischi.
1F ₁₆	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 00400₁₆ e fino a 004FF₁₆ incluso, si trova un'area di memoria utilizzata dal BIOS, per annotarvi 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 A0000₁₆, fino a BFFFF₁₆ 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 F0000₁₆ potrebbe essere priva di qualunque cosa) in particolare il blocco da F0000₁₆ a FFFFF₁₆ 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 A0000₁₆ fino alla fine, non può essere utilizzata, ma rimane incerta una porzione di memoria antecedente l'indirizzo A0000₁₆ 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 00413₁₆, 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 00500₁₆, ovvero dalla fine della BDA. In alternativa, si può chiamare l'interruzione 12₁₆, 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 10₁₆/AH=00₁₆ «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
AH	00 ₁₆
AL	Modalità video: 00 ₁₆ testo, 40x25 caratteri, 16 colori, 8 pagine; 03 ₁₆ testo, 80x25 caratteri, 16 colori, 4 pagine.

INT 10₁₆/AH=05₁₆ «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
AH	05 ₁₆
BH	Pagina video da selezionare.

INT 10₁₆/AH=0E₁₆ «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
AH	0E ₁₆
AL	Carattere da rappresentare.
BL	Numero della pagina video.

INT 12₁₆ «get memory size»

« Restituisce la dimensione della memoria utilizzabile, partendo dall'indirizzo 0000₁₆, espressa in multipli di 1024 byte. La memoria utilizzabile effettivamente inizia solo a partire dall'indirizzo 00500₁₆. Eventualmente lo stesso valore sarebbe accessibile all'indirizzo 00413₁₆, leggendo un numero da 16 bit.

Valore restituito	Descrizione
AX	Dimensione della memoria disponibile.

INT 13₁₆/AH=00₁₆ «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 80₁₆ in su.

Parametro	Descrizione
AH	00 ₁₆
DL	Numero dell'unità da azzerare.

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

INT 13₁₆/AH=02₁₆ «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 80₁₆ in su.

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

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

INT 13₁₆/AH=03₁₆ «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 80₁₆ in su.

Parametro	Descrizione
AH	03 ₁₆
DL	Numero dell'unità in cui scrivere.
AL	Quantità di settori da scrivere (deve essere maggiore di zero).

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

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

INT 16₁₆/AH=00₁₆ «get keystroke from keyboard»

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

Parametro	Descrizione
AH	00 ₁₆

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

INT 16₁₆/AH=01₁₆ «check for keystroke in the keyboard buffer»

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

Parametro	Descrizione
AH	01 ₁₆

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

Preparazione 1313
Bcc 1314
As86 1315
Ld86 1315
Bootblocks 1316
Riferimenti 1317

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,¹ 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

Bcc² 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 Unprodi 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 nome	Produce un file con il nome specificato.
-I	Non utilizza i percorsi predefiniti per l'inclusione dei file di intestazione.
-Ipercorso	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
-o	Compila nella modalità a 16 bit e avvisa quando incontra istruzioni che non sono adatte a CPU 8086/8088.
-o nome	Produce un file oggetto con il nome specificato.
-s nome	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 -o -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 nome	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.

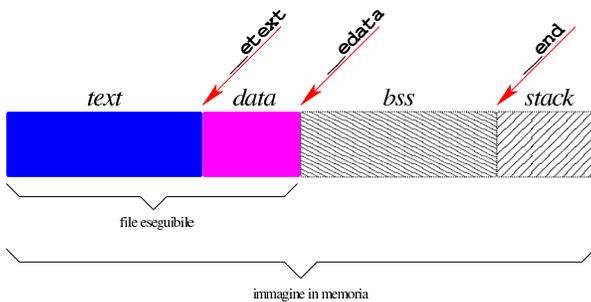
```
$ ld86 -d -s -o kimage crt0.o body.o main.o [Invio]
```

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 As86	per	Nome per Bcc	Descrizione
__etext		__etext	Variabile a 16 bit contenente l'indirizzo conclusivo dell'area usata dal codice, nell'ambito del segmento in cui si colloca.
__edata		__edata	Variabile a 16 bit contenente l'indirizzo conclusivo dell'area usata dai dati inizializzati, nell'ambito del segmento in cui si colloca.
__end		__end	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.
__segofff		__segofff	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 Bootblocks³ 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
'makeboot'	programma per l'installazione del settore di avvio, da usare attraverso un sistema GNU/Linux comune;
'makeboot.com'	programma analogo a 'makeboot', 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 10000₁₆. Pertanto, dopo l'avvio effettivo, rimane inutilizzato lo spazio di memoria da 00500₁₆ 0FFFF₁₆.

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/~ralff/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/>

¹ Bcc, As86, Ld86 GNU GPL

² 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.

³ Bootblocks GNU GPL

