La CPU Intel 8086:
Architettura
e Programmazione Assembly

Alberto BROGGI
Dipartimento di Ingegneria dell'Informazione
Università di Parma

La CPU Intel 8086: Architettura e Programmazione Assembly

Alberto BROGGI
Dipartimento di Ingegneria dell'Informazione
Università di Parma

Seconda edizione

Per eventuali aggiunte e/o correzioni:

broggi@ce.unipr.it oppure http://www.ce.unipr.it/people/broggi

Prefazione alla prima edizione

Questo volume raccoglie le trasparenze proposte durante le lezioni di Assembly nell'ambito del corso di Calcolatori Elettronici tenuto presso la Facoltà di Ingegneria dell'Università degli Studi di Parma.

In esso è affrontato lo studio del processore 8086 da un punto di vista funzionale con un breve cenno all'architettura hardware. Particolare enfasi è stata posta nella descrizione di ogni istruzione macchina, corredando ogni argomento con esempi.

La seconda parte di questo testo contiene una sostanziosa collezione di gran parte delle Prove Scritte d'Esame assegnate dal 1989 ad oggi.

Spero che questo volume possa essere un valido aiuto a quanti si avvicinano per la prima volta a questa disciplina, un buon riferimento per i più esperti, o che possa almeno occupare un posto nella biblioteca personale di ogni futuro Ingegnere.

Parma, Novembre 1992

Alberto Broggi

Sommario

I LINGUAGGI ASSEMBLY 2
Caratteristiche dei Linguaggi Assembly 3
Statements 4
Istruzioni 5
Pseudo-Istruzioni 9
Macro 10
Commenti 11
Vantaggi dei Programmi Assembly 12

ARCHITETTURA LOGICA DELLA CPU 8086 16
La CPU INTEL 8086 17
Gestione della Memoria 18
Registri e Flags 20
Composizione Manuale di Istruzioni Macchina 27

IL LINGUAGGIO ASSEMBLY 8086 29
Elementi di Base del Linguaggio 30
Istruzioni 35
Modi di Indirizzamento 70
Tempo di Esecuzione delle Istruzioni 87
Pseudo-Istruzioni 89

LE FUNZIONI MS-DOS 90
Accesso alle Funzioni DOS e BIOS 92
Accesso diretto alla memoria video 93

LA FAMIGLIA DI PROCESSORI 80X86 94
Il processore 8008 95
Il processore 8080 96
Il processore 8085 97
Il processore 8086 98
Il processore 8088 99
Il processore 80286 100
Il processore 80386 101
Il processore 80486 102
Il processore Pentium 103
La tecnologia MMX 104
Modalita‘ reale e protetta 105

CREAZIONE DI UN PROGRAMMA ASSEMBLY 106
Suddivisione in moduli differenti 107
L’Assemblatore 108
Il Linker 109
Il Debugger 110
Struttura e Documentazione di un Programma Assembly 111

APPENDICI 114
A: Programma di Esempio 114
B: Esempio di Documentazione dell’Istruzione AND 120
C: Instruction Set della CPU 8086 121

D: Funzioni DOS (INT 21h) 124
E: Funzioni BIOS 127
F: Pseudo Istruzioni del MACRO ASSEMBLER 128
G: Tracce per la Risoluzione di Alcune Prove Scritte 132

La CPU Intel 8086:
Architettura
e Programmazione Assembly

- I Linguaggi Assembly
- Architettura logica della CPU Intel 8086
- Il linguaggio Assembly 8086
- Le funzioni MS-DOS
- La famiglia di processori 80X86
- Procedura di creazione di un programma Assembly

I LINGUAGGI ASSEMBLY

- Caratteristiche dei Linguaggi Assembly
- Statements
- Istruzioni
- Pseudo-Istruzioni
- Macro
- Commenti
- Vantaggi dei Programmi Assembly

Caratteristiche dei Linguaggi Assembly

- Sono linguaggi di basso livello
- Vi è corrispondenza uno a uno con le istruzioni del linguaggio macchina
- I simboli mnemonici utilizzati sono associati a
  - istruzioni
  - sequenze di istruzioni
  - indirizzi di memoria
  - aree di memoria
  - dispositivi di I/O
- Possibilità di utilizzare al meglio la macchina hardware
- La stesura di un programma Assembly è molto complessa
- Possibilità, nei macro-assemblatori, di definire macro-istruzioni
- Possibilità di introdurre nel programma chiamate di libreria

Statements

Un programma Assembly è composto di *Statements*. Ogni statement comprende una direttiva per l'Assemblatore e corrisponde ad una riga del programma.

Se la direttiva corrisponde ad una istruzione macchina eseguibile dalla CPU, essa è detta *Istruzione*, altrimenti è una *Pseudo-Istruzione*.

Nel seguito verranno quindi analizzate:

- Istruzioni
  - Etichette
  - Codici Operativi
  - Operandi
- Pseudo-Istruzioni
- Macro
- Commenti

Istruzioni

Vengono tradotte dall'Assemblatore in istruzioni macchina. Ogni istruzione è composta in generale da:

- una *Etichetta* (o Label)
- un *Codice Operativo* (o Operation Code)
- uno o più *Operandi* (o Operands)

Esempio:

```
Label       OpCode      Operand(s)

START:      MOV         AX, BX
            CMP         AX, 12h
            JZ          EQUAL
            INT         21h
            RET
EQUAL:      ...
            ...
```

Etichette

Sono identificatori associati ad una istruzione; l'assemblatore le sostituisce con l'*indirizzo* dell'istruzione che rappresentano.

Offrono i seguenti vantaggi:

- permettono di trovare più facilmente un punto del programma
- permettono di non avere a che fare con indirizzi fisici
- facilitano la modifica del programma

Esempio:

```
                              (0111)

           010C    JLE   DIGIT1
           010E    SUB   DL
DIGIT1:    0111    MOV   CL, 2
           0113    SHL   DL, 1
```

Codici Operativi

- È lo mnemonico di un'istruzione assembly: in altri termini specifica l'operazione che deve essere eseguita dalla CPU

- È l'unico campo che non può mai mancare in un'istruzione

Esempio:

```
START:    MOV    AX, BX
          CMP    AX, 12h
          JZ     EQUAL
          INT    21h
          RET
EQUAL:    ...
          ...
```

Operandi

Contiene l'indicazione necessaria a reperire gli operandi (uno o più, a seconda dei casi) richiesti dall'istruzione.

Sulla base di quanto indicato in questo campo, la CPU provvederà, durante l'esecuzione del programma, a reperire gli operandi:

- nell'istruzione stessa
- in un registro
- in memoria
- su una porta di I/O

Esempio:

```
MOV     AX, 2
MOV     AX, BX
MOV     AX, VALORE
IN      AX, DX
```

Pseudo-Istruzioni

Sono comandi utilizzati durante il processo di assemblaggio (dall'Assemblatore o Assembler), che non vengono tradotti in istruzioni macchina eseguibili dalla CPU.

Esempio:

```
                TITLE    Programma di Prova
DSEG             SEGMENT PARA PUBLIC 'DATA'
LETTURA_SN        PROC   NEAR       ;Inizio procedura LETTURA_SN
                ENDP            ;Fine procedura LETTURA_SN
                END             ;Fine del codice da assemblare
```

Macro

Sono comandi utilizzati per semplificare la stesura di un programma complesso in cui c'è la necessità di ripetere più volte determinati segmenti di codice.

Vengono tradotti in sequenze di istruzioni macchina eseguibili dalla CPU.

Esempio:

```
SHIFT˙LEFT˙AX˙4        MACRO
                      SHL   AX, 1
                      SHL   AX, 1
                      SHL   AX, 1
                      SHL   AX, 1
SHIFT˙LEFT˙AX˙4        ENDM

LOAD˙AX˙AND˙MUL˙16      MACRO    VALUE
                      MOV   AX, VALUE
                      SHIFT˙LEFT˙AX˙4
LOAD˙AX˙AND˙MUL˙16      ENDM

                      MOV   AX, [MEM]
                      SHIFT˙LEFT˙AX˙4
                      MOV   BX, AX
                      LOAD˙AX˙AND˙MUL˙16   20
                      MOV   BX, AX
                      ...
```

Commenti

Sono parole o frasi inserite dal programmatore per rendere il programma più comprensibile; servono al programmatore stesso e a chi analizzerà in futuro il codice.

Vengono ignorati dall'assemblatore, che si limita a visualizzarli quando si richiede il listato del programma.

Tutti i caratteri compresi tra un ';' e un $< CR >$, vengono considerati commenti.

- Devono essere utili ed esplicativi; ad esempio:

```
;
;  programma mal commentato
;
   START:    MOV      AX, BX   ;Carico AX con il contenuto di BX
             CMP      AX, 24   ;Confronto AX con il valore 24 dec.
             JZ       EQUAL    ;Se AX=24 allora salta a EQUAL
             INT      21h      ;Chiama l'INTERRUPT numero 21 hex.
             RET               ;Ritorna alla procedura chiamante
   EQUAL:    ...

;
;  programma ben commentato
;
   START:    MOV      AX, BX   ;Carico in AX il numero della riga
             CMP      AX, 24   ;Se sono al termine dello schermo
             JZ       EQUAL    ; allora non scrivo nulla
             INT      21h      ;Scrivi la prossima riga del testo
             RET               ;Ritorna
   EQUAL:    ...
```

Vantaggi dei programmi Assembly

L'utilizzo del linguaggio Assembly anzichè di un linguaggio ad alto livello (tipo C o Pascal) è talvolta giustificato dalla maggiore efficienza del codice; infatti i programmi in Assembly sono tipicamente

- più veloci,
- più corti,
- ma più complessi

dei programmi scritti in linguaggi ad alto livello.

La maggior complessità è data dal fatto che anche le più comuni routines devono essere sintetizzate dal programmatore (talvolta per semplificare la programmazione e per aumentare la compatibilità del codice, si utilizzano librerie *general purpose*, ma sono ovviamente meno efficienti).

Come esempio si consideri un programma per stampare i numeri pari da 0 a 100:

Il programma BASIC è:

```
100 I=0
110 PRINT I
120 I=I+2
130 IF I¡100 GOTO 110
```

Il codice Assembly generato da un compilatore BASIC è il seguente:

```
I        DW     ?

L00100:  MOV    I, 0
L00110:  MOV    AX, I
         CALL   STAMPA
L00120:  MOV    AX, I
         ADD    AX, 2
         MOV    I, AX
L00130:  MOV    AX, I
         CMP    AX, 100
         JB     L00110
```

Si notano almeno due semplici modifiche, che ne migliorano notevolmente le prestazioni:

- L'uso di registri al posto di locazioni di memoria
- L'uso di particolari caratteristiche dell'Assembly

Il programma scritto direttamente in Assembly è quindi il seguente:

```
        MOV   AX,0        ;Inizializza il valore del contatore
CICLO:  CALL  STAMPA      ;Stampa il valore corrente di AX
        INC   AX          ;Calcola il nuovo numero pari a
        INC   AX          ;   partire dal vecchio AX
        CMP   AX, 100     ;Se AX non ha raggiunto il valore
        JB    CICLO       ;   massimo, ritorna a CICLO
```

Il programma così ottenuto presenta rispetto a quello prodotto dal compilatore BASIC due fondamentali vantaggi:

- è più veloce (perchè usa i registri e non locazioni di memoria)
- è composto da un numero minore di istruzioni e quindi occupa una minore estensione di memoria

NB: si noti che l'operazione generale di somma (in questo caso $+2$) è stata tradotta in una sequenza di operazioni elementari *ad hoc*.

Per programmi più articolati risulta più evidente la maggiore complessità di sintesi direttamente in Assembly.

Esempio di procedura Assembly: assemblato e disassemblato

```
;*********************************************************************
;*                                                                   *
;*     Procedura di attesa di risposta (S/N) dall'utente via tastiera  *
;*                                                                   *
;*********************************************************************

LETTURA_SN          PROC   NEAR          ;Inizio procedura LETTURA_SN
NUOVA_LETTURA:      MOV    AH,07h        ;Servizio DOS 'Read Keyboard
                    INT    21h           ; Without Echo'
                    OR     AL,20h        ;Converte in minuscolo
                    CMP    AL,'n'        ;Se il tasto premuto e' 'N'
                    JZ     FINE_LETTURA  ; esce dalla procedura
                    CMP    AL,'s'        ;Se non e' 'S',
                    JNZ    NUOVA_LETTURA ; ne legge un altro
FINE_LETTURA:       RET                  ;Ritorno alla proc. chiamante
LETTURA_SN          ENDP                 ;Fine della procedura
```

```
57DA:00A2  B407    MOV   AH,07
57DA:00A4  CD21    INT   21
57DA:00A6  0C20    OR    AL,20
57DA:00A8  3C6E    CMP   AL,6E
57DA:00AA  7404    JZ    00B0
57DA:00AC  3C73    CMP   AL,73
57DA:00AE  75F2    JNZ   00A2
57DA:00B0  C3      RET
```

ARCHITETTURA LOGICA DELLA CPU INTEL 8086

- La CPU INTEL 8086
- Gestione della Memoria
- Registri e Flags
- Composizione manuale di istruzioni macchina

La CPU INTEL 8086

L'8086 è un microprocessore *general purpose* a 16 bit.

Le caratteristiche principali sono:

- Capacità di indirizzamento di 1 MegaByte
- 14 registri interni da 16 bit
- 7 modi di indirizzamento
- Alimentazione a 5 volt
- 48 pin di interconnessione
- Set di istruzioni esteso (CISC)

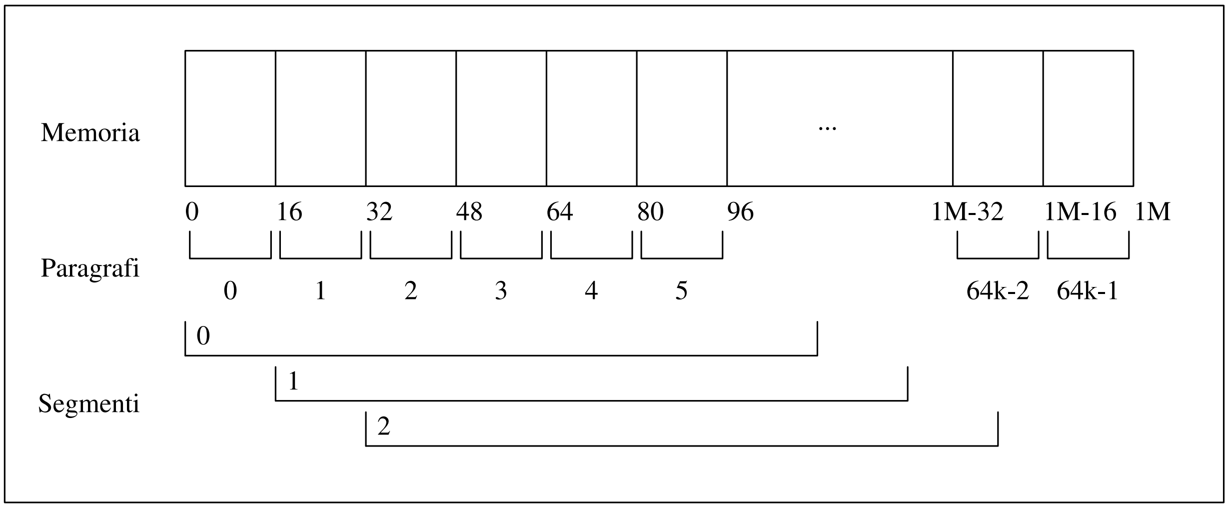
Gestione della memoria

Per comodità la memoria si può pensare divisa in

- *Paragrafi*
- *Segmenti*

I *paragrafi* sono zone di memoria costituite da 16 byte contigui. Il sistema può gestire fino a 64k paragrafi. I paragrafi sono numerati a partire dalla locazione 00000h di memoria. I paragrafi non possono sovrapporsi.

I *segmenti* sono zone di memoria costituite da 64k byte contigui. Il sistema può gestire fino a 64k segmenti; ogni segmento inizia in corrispondenza con un paragrafo, ossia ad un indirizzo multiplo di 16. I segmenti possono sovrapporsi (*Overlapping Segments*).

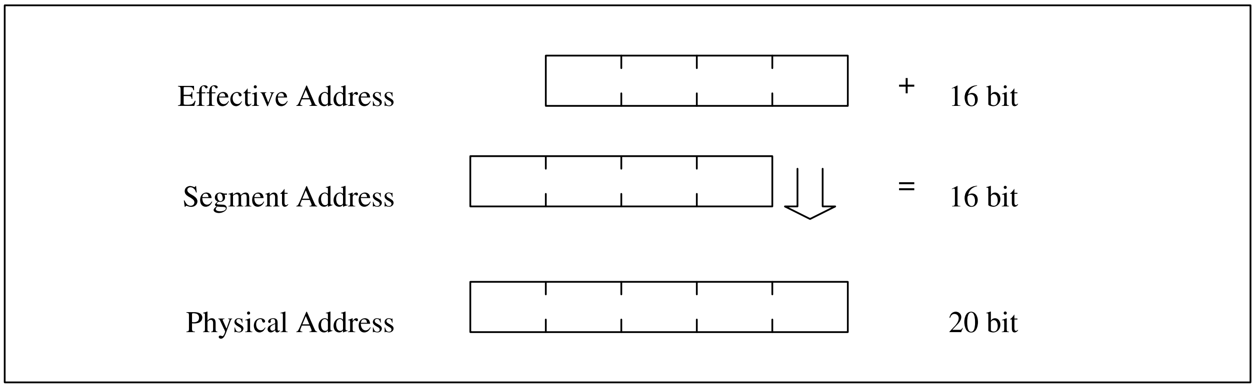


Calcolo dell'Indirizzo Fisico

L'indirizzo fisico di una cella di memoria è espresso da 20 bit; non è quindi possibile un indirizzamento mediante un *solo* registro a 16 bit. Esso è infatti ottenuto mediante la somma di due contributi:

- *il Segment Address:*
  è l'indirizzo di testa del segmento e viene ottenuto moltiplicando per 16 il numero del segmento.
- *l'Effective Address (EA):*
  è l'indirizzo effettivo all'interno del segmento, calcolato come *offset* (spostamento) rispetto all'inizio del segmento stesso.

NB: la moltiplicazione per 16 può essere notevolmente velocizzata da un semplice *shift* a sinistra di 4 posizioni della rappresentazione binaria del numero.

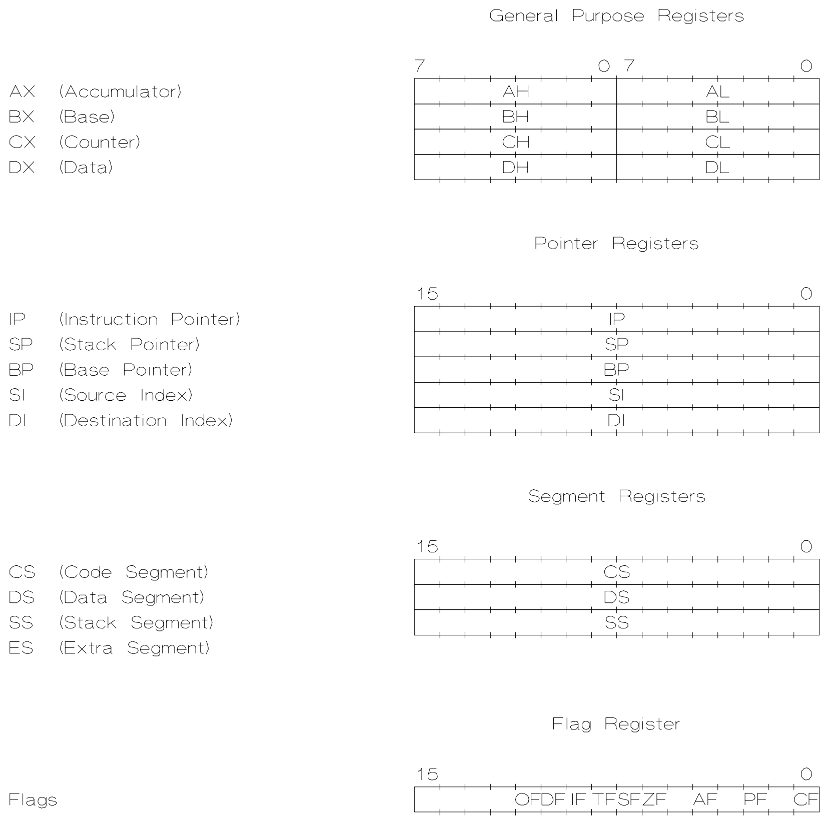


Registri e Flags

La CPU INTEL 8086 possiede i seguenti 14 registri:

- 4 General Purpose Registers
- 5 Pointer (o Index o Offset) Registers (+1)
- 4 Segment Registers
- 1 Flag Register

La precedente suddivisione è dettata dai tipi di operazioni che tali registri possono eseguire.



General Purpose Registers

Sono registri da 16 bit, ciascuno dei quali utilizzabile indifferentemente come un registro da 16 bit o come due registri da 8 bit. Essi sono:

- AX: *Accumulator*, utilizzabile anche come AH + AL
- BX: *Base*, utilizzabile anche come BH + BL
- CX: *Count*, utilizzabile anche come CH + CL
- DX: *Data*, utilizzabile anche come DH + DL

Pointer Registers

I registri Pointer (o Index o Offset) sono generalmente utilizzati come puntatori a dati in memoria. Si possono dividere in:

- *Base Pointer*
  - BX: *Data Segment Base Pointer* (è il BX precedente!)
  - BP: *Stack Segment Base Pointer*
- *Index Pointer*
  - SI: *Source Index Pointer*
  - DI: *Destination Index Ponter*
- *Stack Pointer*
  - SP: *Stack Pointer*
- *Instruction Pointer*
  - IP: *Instruction Pointer*

Segment Registers

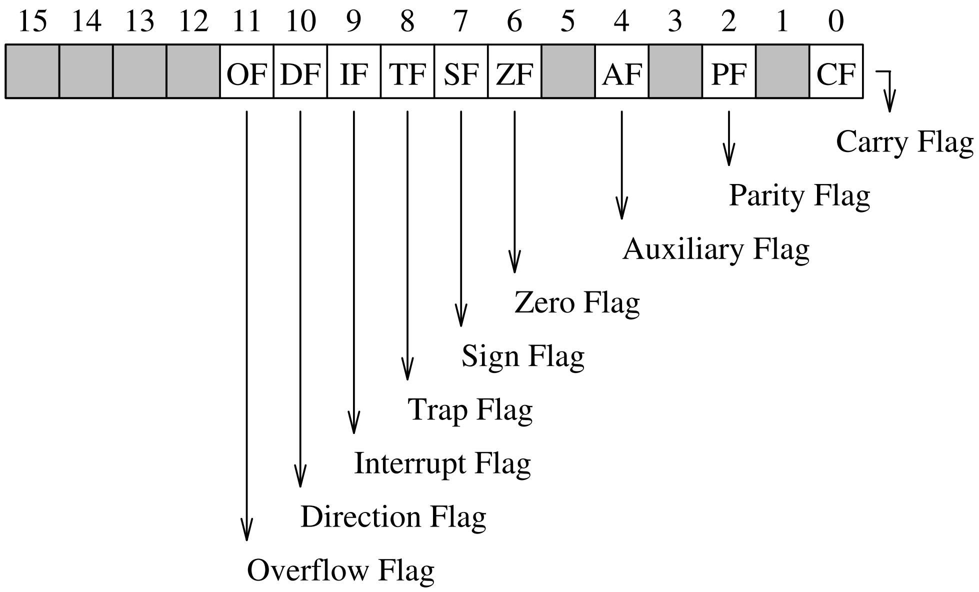
Sono 4 registri destinati a contenere l'indirizzo di testa dei segmenti usati dal programma.

- CS: *Code Segment Register*
  Contiene sempre l'indirizzo di testa del segmento contenente il codice; viene inizializzato dal Sistema Operativo e non deve essere utilizzato dal programmatore.
- SS: *Stack Segment Register*
  Contiene sempre l'indirizzo di testa del segmento contenente lo stack; viene inizializzato dal Sistema Operativo e non deve essere modificato dal programmatore.
- DS: *Data Segment Register*
  Di solito contiene l'indirizzo di testa del segmento dei dati utilizzati dal programma; deve essere inizializzato dal programmatore all'interno del suo programma.
- ES: *Extra Segment Register*
  Può essere utilizzato per definire un segmento ausiliario, per esempio per un ulteriore segmento dati; deve essere inizializzato dal programmatore all'interno del suo programma.

Flag Register

L'8086 dispone di 9 bit detti *flag*, organizzati all'interno di un registro a 16 bit (*Flag Register*).

I bit non usati sono fissi al valore 0.



I *flag* si possono suddividere in due categorie:

- *di stato*
- *di controllo*

Flag di Stato

Forniscono indicazioni relative al risultato dell'ultima istruzione eseguita. Vengono automaticamente aggiornati dal processore ed il loro valore può essere testato dall'utente tramite opportune istruzioni. Sono:

- *Carry Flag* (CF)
  Forzato ad 1 principalmente quando un'istruzione di somma (o sottrazione) produce un riporto (o un prestito)
- *Parity Flag* (PF)
  Forzato ad 1 quando il risultato di una operazione contiene un numero pari di 1; usato principalmente per trasmissione dati.
- *Auxiliary Carry Flag* (AF)
  Forzato ad 1 quando si produce un riporto tra il bit 3 e il bit 4 di un operando; usato per operazioni su numeri decimali *packed*
- *Zero Flag* (ZF)
  Forzato ad 1 quando il risultato di una operazione è un valore nullo; rimane a 0 altrimenti.
- *Sign Flag* (SF)
  Ripete il valore del bit più significativo del risultato di una operazione
- *Overflow Flag* (OF)
  Forzato ad 1 quando un'operazione aritmetica da origine ad una condizione di overflow.

Flag di Controllo

Il loro valore può essere forzato dall'utente attraverso apposite istruzioni.

In determinate situazioni sono testati dal processore che, a seconda del loro valore, si comporta in modi diversi.

- *Trap Flag* (TF)
  Usato in ambiente di *debug*, causa l'esecuzione *single-step* dei programmi
- *Interrupt Enable Flag* (IF)
  Usato per disabilitare (quando uguale a 0) eventuali richieste di interruzioni esterne (*interrupt*)
- *Direction Flag* (DF)
  Usato nelle operazioni sulle stringhe per regolare l'incremento (DF=0) o il decremento (DF=1) dei Registri Indice

Composizione Manuale di Istruzioni Macchina

Basandosi sulle tabelle fornite dal costruttore, si emuli il comportamento dell'Assemblatore, costruendo manualmente un'istruzione macchina: ad esempio

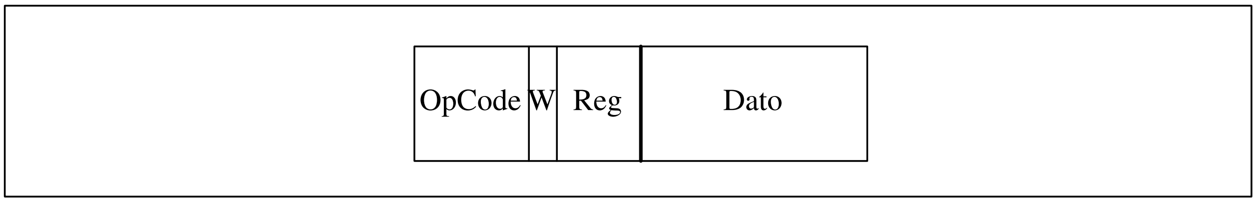
MOV AH,11

Si esamina la tabella relativa all'istruzione MOV, unitamente alla speigazione dei simboli in essa contenuti.

Esempio: MOV AH, 11

Dalla tabella fornita per l'istruzione MOV si osserva il formato nel caso di trasferimento di un dato contenuto nell'istruzione stessa (*immediate*) in un registro; si consideri quindi la terza riga della tabella.

Il formato macchina dell'istruzione occupa quindi due byte, così composti:



Il significato dei vari campi è il seguente:

- i primi 4 bit rappresentano l'*Operation Code*, che identifica il tipo di istruzione;
- il bit W vale 1 se l'operando è su 16 bit, 0 se è su 8 bit: nel caso in esame $W = 0$;
- i 3 bit *Reg* indicano il registro destinazione, secondo la tabella indicata: nel caso in esame $Reg = 100$;
- gli 8 bit di dato contengono l'operando:
  nel caso in esame $11_{(10)} = 00001011_{(2)}$.

Il formato macchina dell'istruzione MOV AH, 11 è quindi:

1011 0 100 00001011

IL LINGUAGGIO ASSEMBLY 8086

- Elementi di Base del Linguaggio
- Istruzioni
- Modi di Indirizzamento
- Tempo di Esecuzione delle Istruzioni
- Pseudo-Istruzioni

Come esempio si farà riferimento ad un semplice programma di conversione di formato di interi.

Elementi di base del linguaggio

- Un programma scritto in Assembly 8086 è composto di *Statements*; normalmente ognuno di essi occupa una riga fino ad un $< LF >$ o una coppia $< CR >< LF >$.
- Uno statement può proseguire sulla riga successiva, se questa comincia con il carattere '&'.
- L'insieme dei caratteri utilizzabili è composto da
  - caratteri alfanumerici (maiuscole, minuscole, cifre),
  - caratteri non stampabili (spazio, *TAB*, $< CR >$, $< LF >$),
  - caratteri speciali (+ − ∗/ = ()[] <>;'.",_:?@$&)
- All'interno del programma possono comparire:
  - Identificatori
  - Costanti
  - Espressioni

Identificatori

- Sono usati come nomi assegnati ad entità definite dal programmatore (segmenti, variabili, label, etc.)
- Sono composti da lettere, numeri o uno dei tre caratteri @ ? _, ma non possono iniziare con un numero
- Hanno lunghezza massima di 31 caratteri

Costanti

Si possono utilizzare costanti:

- binarie: 001101B
- ottali: 15O, 15Q
- esadecimali: 0Dh, 0BEACh (devono iniziare con un numero)
- decimali: 13, 13D
- ASCII: 'S', 'Salve'
- reali in base 10: 2.345678, 112E-3

Espressioni

Si possono utilizzare i seguenti operatori:

- artimetici
  ($+, -, *, /$, MOD, SHL, SHR)
- logici
  (AND, OR, XOR, NOT)
- relazionali
  (EQ, NE, LT, GT, LE, GE)
- che ritornano un valore
  ($, SEG, OFFSET, LENGTH, TYPE)
- attributi
  (PTR, DS:, ES:, SS:, CS:, HIGH, LOW)

Precedenze tra gli operatori

Gli operatori visti possono essere elencati in ordine di priorità decrescente nel modo che segue:

- LENGTH, SIZE, WIDTH, MASK, $()$, $[]$, $<>$
- PTR, OFFSET, SEG, TYPE, THIS, *segment override*
- HIGH, LOW
- $+$ (unario), $-$ (unario)
- $*$, $/$, MODE, SHL, SHR
- $+$, $-$
- EQ, NE, LT, LE, GT, GE
- NOT
- AND
- OR, XOR
- SHORT

La priorità può essere modificata tramite l'uso delle parentesi tonde.

Istruzioni

L'Assembly 8086 rende disponibili 92 tipi di istruzioni, raggruppabili nelle seguenti classi:

- Trasferimento Dati
- Aritmetiche
- Manipolazione di Bit
- Trasferimento di Controllo
- Manipolazione di Stringhe
- Manipolazione di Interruzioni
- Controllo del Processore

Istruzioni di Trasferimento Dati

	OpCode	Descrizione
General Purpose		
	MOV	Move (Byte or Word)
	POP	Pop a Word from the Stack
	PUSH	Push Word onto Stack
	XCHG	Exchange Registers
	XLAT	Translate
Input/Output		
	IN	Input Byte or Word
	OUT	Output to Port
Trasf. di indirizzi		
	LDS	Load Pointer Using DS
	LEA	Load Effective Address
	LES	Load Pointer Using ES
Trasf. Flag Register		
	LAHF	Load Register AH from
	SAHF	Store Register AH into
	POPF	Pop Flags from the Stack
	PUSHF	Push Flags onto Stack

Combinazioni non ammesse da MOV

Non sono ammessi i seguenti trasferimenti:

- *memoria* ← *memoria*
  Si deve passare attraverso un registro general-purpose: esempio:

```
MOV   AX, SRC
MOV   DEST, AX
```

- *segment register* ← *immediato*
  Si deve passare attraverso un registro general-purpose: esempio:

```
MOV   AX, DATA_SEG
MOV   DS, AX
```

- *segment register* ← *segment register*
  Si deve passare attraverso un registro general-purpose (4 cicli): esempio:

```
MOV   AX, ES
MOV   DS, AX
```

  oppure attraverso lo stack (26 cicli): esempio:

```
PUSH  ES
POP   DS
```

- Qualsiasi trasferimento che utilizzi CS come destinazione

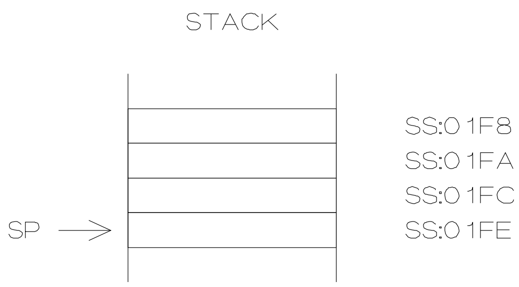
Uso dello Stack

Lo *Stack* è un potente mezzo per memorizzare dati *run-time*; è inoltre basilare ricordare che è una struttura dati LIFO (*Last In First Out*) e i dati devono venire estratti (POP) nell'ordine inverso a quello in cui erano stati inseriti (PUSH).

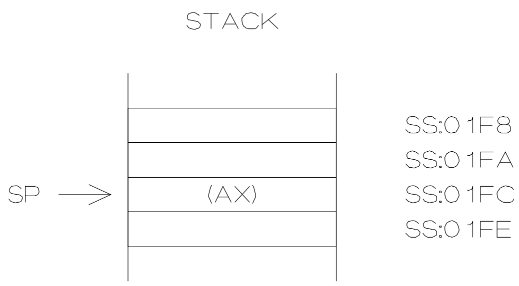
Esempio:

```
PUSH  AX
PUSH  ES
PUSH  DI
PUSH  SI
...
POP   SI
POP   DI
POP   ES
POP   AX
```

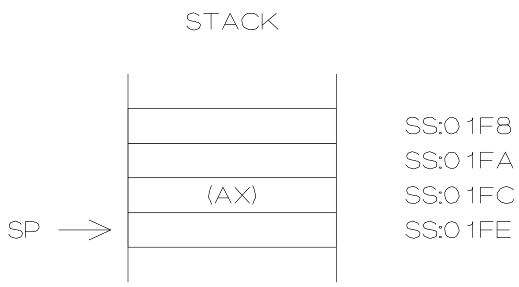
Effetti di PUSH e POP sullo Stack



Prima di PUSH AX



Dopo PUSH AX



Dopo POP AX

Istruzioni Artimetiche

	OpCode	Descrizione
Addizione		
	AAA	ASCII Adjust after Addition
	ADC	Add with Carry
	ADD	Addition
	DAA	Decimal Adjust after Addition
	INC	Increment
Sottrazione		
	AAS	ASCII Adjust after Subtraction
	SUB	Subtract
	SBB	Subtract with Borrow
	DAS	Decimal Adjust after Subtraction
	DEC	Decrement
	CMP	Compare
	NEG	Negate
Moltiplicazione		
	AAM	ASCII Adjust after Multiply
	IMUL	Integer Multiply, Signed
	MUL	Multiply, Unsigned
Divisione		
	AAD	ASCII Adjust before Division
	DIV	Divide, Unsigned
	IDIV	Integer Divide, Signed
Conversione		
	CBW	Convert Byte to Word
	CWD	Convert Word to Doubleword

Formato dei dati nelle Istruzioni Aritmetiche

Il processore 8086 può eseguire operazioni aritmetiche su numeri nei seguenti formati:

- numeri binari senza segno, su 8 o 16 bit
- numeri binari con segno, su 8 o 16 bit
- numeri decimali *packed*, in cui ogni byte contiene due numeri decimali codificati in BCD; la cifra più significativa è allocata nei 4 bit superiori
- numeri decimali *unpacked*, in cui ogni byte contiene un solo numero decimale BCD nei 4 bit inferiori; i 4 bit superiori devono essere a 0 se il numero è usato in una operazione di moltiplicazione o divisione

Operazioni su 32 bit

Le operazioni di somma e sottrazione possono essere facilmente eseguite anche su operandi di dimensioni superiori a 16 bit, usando le istruzioni

- **ADC** (ADd with Carry)
- **SBB** (SuBtract with Borrow)

che eseguono rispettivamente le seguenti operazioni:

- $destination \leftarrow destination + source + Carry$
- $destination \leftarrow destination - source - Carry$

Esempio 1:
Per sommare i 32 bit memorizzati in BX:AX con DX:CX, lasciando il risultato in BX:AX,

```
ADD     AX, CX     ;Somma i 16 bit meno significativi
ADC     BX, DX     ;Somma i 16 bit piu' significativi
```

Esempio 2:
Per sottrarre i 32 bit memorizzati in BX:AX a DX:CX, lasciando il risultato in BX:AX,

```
SUB     AX, CX     ;Sottrae le word meno significative
SBC     BX, DX     ;Sottrae le word piu' significative
```

Moltiplicazione e Divisione

Il processore 8086, a differenza di molti processori ad 8 bit, dispone delle istruzioni di moltiplicazione e divisione.

Per entrambe le operazioni esistono forme distinte a seconda che gli operandi siano interi senza segno (MUL o DIV) o interi con segno (IMUL e IDIV).

Le due operazioni hanno *un solo* operando, che deve essere un registro generale o una variabile (cioè il contenuto di una locazione di memoria).

A seconda delle dimensioni di tale operando (byte o word), si hanno operazioni di tipo *byte* oppure *word*.

Moltiplicazione

Si distinguono i due casi:

- operazioni di tipo *byte*:
  AX $\leftarrow$ AL * *source-8-bit*
- operazioni di tipo *word*:
  DX:AX $\leftarrow$ AX * *source-16-bit*

Divisione

Si distinguono i due casi:

- operazioni di tipo *byte*:
  AL $\leftarrow$ INT(AX / *source-8-bit*)
  AH $\leftarrow$ *resto*
- operazioni di tipo *word*:
  AX $\leftarrow$ INT(DX:AX / *source-16-bit*)
  DX $\leftarrow$ *resto*

Operazioni su Numeri Decimali

Il processore 8086 dispone di alcune istruzioni che permettono di eseguire le 4 operazioni fondamentali anche sui numeri decimali.

Esse non hanno operandi, in quanto lavorano sempre sul registro AL (AX per la moltiplicazione). Nel caso della divisione l'istruzione di conversione deve essere applicata al dividendo (in AX) prima della divisione.

Il risultato della conversione è memorizzato ancora nel registro AL (AX per la moltiplicazione e la divisione).

Esempio di diversa rappresentazione:

binario:	0000 0000 0010 0011
decimale packed:	0011 0101
decimale unpacked:	0000 0011 0000 0101

Le Istruzioni di conversione decimale-binario sono:

- **AAA**:
  converte il risultato di un'addizione in decimale *unpacked*
- **AAS**:
  converte il risultato di una sottrazione in decimale *unpacked*
- **AAM**:
  converte il risultato di una moltiplicazione in decimale *unpacked*
- **AAD**:
  converte il dividendo di una divisione da decimale *unpacked* a binario
- **DAA**:
  converte il risultato di un'addizione in decimale *packed*
- **DAS**:
  converte il risultato di una sottrazione in decimale *packed*

Istruzioni per la Manipolazione dei Bit

	OpCode	Descrizione
Logiche		
	AND	Logical AND
	OR	Logical OR
	XOR	Exclusive OR
	NOT	Logical NOT
	TEST	Test
Di Traslazione		
	SAL	Shift Arithmetic Left (=SHL)
	SAR	Shift Arithmetic Right
	SHL	Shift Logical Left (=SAL)
	SHR	Shift Logical Right
Di Rotazione		
	ROL	Rotate Left
	ROR	Rotate Right
	RCL	Rotate through Carry Left
	RCR	Rotate through Carry Right

Istruzioni di Shift e Rotate

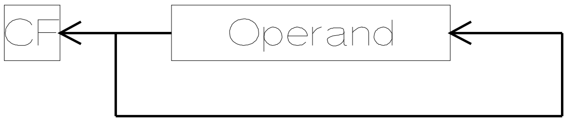
SHL
SHifr logical Left

SHR
SHifr logical Right

SAL
Shift Arithmetic Left

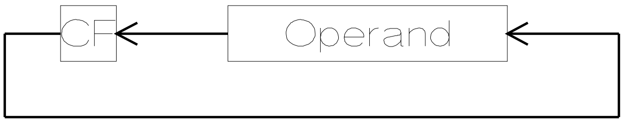
SAR
Shift Arithmetic Right

ROL
ROtate Left

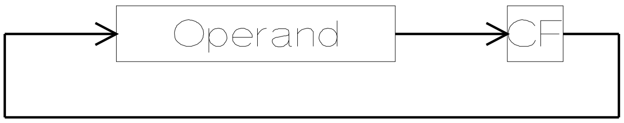


ROR
ROtate Right

RCL
Rotate trough Carry Left



RCR
Rotate trough Carry Right



Trasferimento del Controllo

	OpCode	Descrizione
Salti incondizionati		
	CALL	Call Procedure
	RET	Return from Procedure
	JMP	Jump Unconditionally
Salti condizionati		
	JA, JNBE	Jump If Above
	JAE, JNB	Jump If Above or Equal
	JB, JNAE, JC	Jump If Below
	JBE, JNA	Jump If Below or Equal
	JCXZ	Jump if CX Register Zero
	JE, JZ	Jump If Equal
	JG, JNLE	Jump If Greater
	JGE, JNL	Jump If Greater or Equal
	JL, JNGE	Jump If Less
	JLE, JNG	Jump If Less or Equal
	JNC	Jump If No Carry
	JNE, JNZ	Jump If Not Equal
	JNO	Jump If No Overflow
	JNP, JPO	Jump If No Parity
	JNS	Jump If No Sign
	JO	Jump If Overflow
	JP, JPE	Jump If Parity
	JS	Jump If Sign
Istruzioni iterative		
	LOOP	Loop on Count
	LOOPE, LOOPZ	Loop While Equal
	LOOPNE, LOOPNZ	Loop While Not Equal

Istruzioni di Salto

Permettono di effettuare:

- *Salti Condizionati:*
  - il salto viene eseguito o meno a seconda del valore corrente di uno o più flag; questi sono solitamente modificati da una precedente istruzione CMP, SUB, etc.;
  - la distanza relativa dell'istruzione cui saltare deve essere compresa tra -128 e $+127$;
  - non sono ammessi salti ad un segmento diverso da quello corrente.
- *Salti Incondizionati:*
  Il salto avviene comunque e ad istruzioni ovunque posizionate, anche all'interno di un segmento diverso da quello corrente.

Nota:

Alcune delle istruzioni di salto condizionato elencate sono ridondanti: ad esempio la **JA** *(Jump if Above)* è equivalente alla **JNBE** *(Jump if Not Below or Equal)*.

La seguente tabella riporta l'insieme minimo di istruzioni di salto condizionato da usare dopo un'istruzione

$$\text{CMP } NUM_1, NUM_2$$

dove NUM_1 e NUM_2 possono essere due numeri con o senza segno.

	numeri senza segno	numeri con segno
$NUM_1 > NUM_2$	JA	JG
$NUM_1 = NUM_2$	JE	JE
$NUM_1 \neq NUM_2$	JNE	JNE
$NUM_1 < NUM_2$	JB	JL
$NUM_1 \leq NUM_2$	JBE	JLE
$NUM_1 \geq NUM_2$	JAE	JGE

Le Istruzioni CALL e RET

L'Assembly 8086 permette l'uso delle *Procedure*, in modo simile a quanto avviene nei linguaggi ad alto livello: l'unica differenza sta nell'impossibilità di passare dei parametri nel modo consueto.

Mediante una *Procedura* è possibile scrivere una volta per tutte quelle parti di codice che vengono ripetutamente eseguite in un programma, ottenendo molti vantaggi:

- Maggiore leggibilità del codice
- Risparmio di tempo per il programmatore
- Risparmio di memoria occupata dal codice
- Possibilità di *ricorsione* e *annidamento* (limitate solo dalle dimensioni dello stack)
- Una *Procedura* è diversa da una *Macro*!

Le istruzioni CALL e RET permettono di effettuare una chiamata ad una procedura e di ritornare da essa.

Una procedura può essere:

- *NEAR:*
  può essere chiamata solo dall'interno dello stesso segmento di codice cui appartiene;
- *FAR:*
  può essere chiamata dall'interno di un segmento di codice qualsiasi.

Il tipo di procedura (*NEAR* o *FAR*) deve essere dichiarato all'atto della creazione della procedura stessa.

La **CALL** provvede a:

- salvare il valore corrente di IP (e di CS nel caso di procedura *FAR*) nello stack, tramite un'operazione di PUSH;
- caricare in IP (e in CS) il valore corrispondente all'indirizzo di partenza della procedura chiamata.

La **RET** provvede a:

- ripristinare tramite un'istruzione POP il valore di IP (e di CS nel caso di procedura *FAR*) salvato nello stack, riprendendo quindi l'esecuzione del programma a partire dell'istruzione successiva all'ultima CALL.

Istruzioni per il Controllo delle Iterazioni

Il processore 8086 prevede anche alcune istruzioni per la gestione dei loop. Il loro formato è:

OpCode Label

dove *OpCode* può essere:

- **LOOP**:
  decrementa il registro CX e salta a *Label* se questo è diverso da zero;
- **LOOPE / LOOPZ**:
  decrementa il registro CX e salta a *Label* se:
  - CX è diverso da zero e
  - lo Zero Flag vale 1;
- **LOOPNE / LOOPNZ**:
  decrementa il registro CX e salta a *Label* se:
  - CX è diverso da zero e
  - lo Zero Flag vale 0;

Istruzioni per la Manipolazione delle Stringhe

	OpCode	Descrizione
Istruzioni di Spostamento		
	MOVS	Move String (Byte or Word)
	MOVSB	Move String Byte
	MOVSW	Move String Word
Istruzioni di Confronto		
	CMPS	Compare String (Byte or Word)
	CMPSB	Compare String Byte
	CMPSW	Compare String Word
Istruzioni di Ricerca		
	SCAS	Scan String (Byte or Word)
	SCASB	Scan String Byte
	SCASW	Scan String Word
Istruzioni di Caricamento		
	LODS	Load String (Byte or Word)
	LODSB	Load String Byte
	LODSW	Load String Word
Istruzioni di Scrittura		
	STOS	Store String (Byte or Word)
	STOSB	Store String Byte
	STOSW	Store String Word

Le istruzioni per la manipolazione delle stringhe operano su blocchi di dati consecutivi in memoria, denominati *stringhe* e costituiti da *byte* o *word*.

Eseguono 5 tipi di operazioni:

- *Move:*
  sopstamento di un dato da una posizione di memoria ad un'altra
- *Compare:*
  confronto di due dati in memoria
- *Scan:*
  ricerca di un dato in memoria
- *Load:*
  caricamento di un dato dalla memoria in un registro accumulatore
- *Store:*
  scrittura in memoria di un dato presente in un registro accumulatore

Le istruzioni per la manipolazione delle stringhe si distinguono dalle normali istruzioni di MOV, CMP, etc. in quanto:

- l'accesso ai dati in memoria avviene attraverso i registri SI e DI, usati come offset rispettivamente all'interno del Data Segment (DS) e dell'Extra Segment (ES);
- SI e DI vengono automaticamente modificati al termine di ogni operazione, in modo da puntare al successivo dato all'interno della stringa; in particolare SI e DI vengono
  - incrementati se il Direction Flag (DF) vale 0;
  - decrementati se il Direction Flag (DF) vale 1;

  I registri SI e DI vengono incrementati (o decrementati)
  - di 1 nel caso di operazioni su *byte*;
  - di 2 nel caso di operazioni su *word*;

Il flag DF può essere settato e resettato tramite le istruzioni **STD** e **CLD**, rispettivamente.

Tutte le istruzioni di manipolazione di stringhe hanno tre forme:

- operazioni su byte (ad esempio **MOVSB**) che non hanno operandi;
- operazioni su word (ad esempio **MOVSW**) che non hanno operandi;
- operazioni su byte o word (ad esempio **MOVS**) che hanno uno o due operandi; vengono tradotte dall'assemblatore in una delle precedenti due forme, a seconda del tipo degli operandi.

Esempio: nel seguente programma

```
STR1     DB     100 DUP(?)
STR2     DB     100 DUP(?)

         MOVS   STR1, STR2
```

l'istruzione MOVS viene tradotta dall'assemblatore in MOVSB.

Prefissi di Ripetizione

	OpCode	Descrizione
Prefissi di Ripetizione		
	REP	Repeat
	REPE	Repeat While Equal
	REPNE	Repeat While Not Equal
	REPNZ	Repeat While Not Zero
	REPZ	Repeat While Zero

Le istruzioni di manipolazione delle stringhe agiscono sul singolo dato all'interno della stringa, ma sono costruite per essere utilizzate in costrutti iterativi attraverso l'uso dei *Prefissi di Ripetizione*.

Questi, inseriti nel programma sorgente prima dello mnemonico dell'istruzione, permettono di ripetere l'istruzione seguente fino a che non si verifica una determinata condizione: esistono 2 *Prefissi di Ripetizione*:

- **REP** o **REPE** o **REPZ**:
  Se usate in combinazione con
  - MOVS, LODS e STOS, causano la ripetizione dell'istruzione seguente per un numero di volte pari al contenuto iniziale di CX; ad ogni iterazione CX viene decrementato e l'istruzione viene ripetuta fin tanto che CX è diverso da 0; al termine CX vale 0;
  - CMPS e SCAS, il comportamento è uguale al caso precedente, con l'unica differenza che il ciclo viene abbandonato anche se lo Zero Flag (ZF) assume valore 0;
- **REPNE** o **REPNZ**:
  Se usate in combinazione con
  - MOVS, LODS e STOS, causano la ripetizione dell'istruzione seguente per un numero di volte pari al contenuto iniziale di CX; ad ogni iterazione CX viene decrementato e l'istruzione viene ripetuta fin tanto che CX è diverso da 0; al termine CX vale 0;
  - CMPS e SCAS, il comportamento è uguale al caso precedente, con l'unica differenza che il ciclo viene abbandonato anche se lo Zero Flag (ZF) assume valore 1;

Istruzione per la Manipolazione delle Interruzioni

	OpCode	Descrizione
Manipolazione di Interruzioni		
	INT	Interrupt
	INTO	Interrupt on Overflow
	IRET	Interrupt Return

Permettono di gestire gli *Interrupt software*. La richiesta di un'interruzione causa:

- il salvataggio sullo stack dei registri IP e CS;
- il salvataggio sullo stack del Flag Register;
- l'esecuzione di una routine di servizio dell'Interrupt (ISR, *Interrupt Service Routine*).

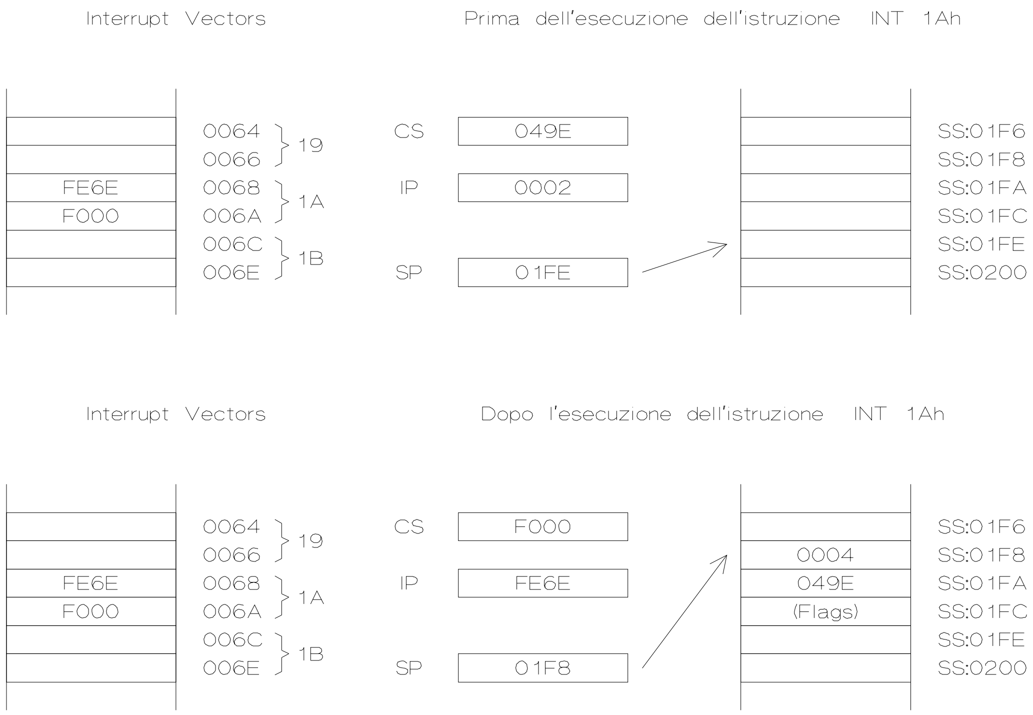
Un Interrupt è simile ad una chiamata a procedura; esistono tuttavia delle differenze:

- Mentre l'esecuzione di una procedura può essere attivata solo via software, l'esecuzione di una ISR può essere attivata anche via hardware;
- Una ISR è sempre *FAR*, in quanto comporta comunque il caricamento del registro CS.
- Un Interrupt causa il salvataggio nello stack anche del Flag Register.

L'attivazione di una ISR è una procedura abbastanza complessa:

l'indirizzo della routine di servizio dell'Interrupt viene ottenuto da una tabella allocata negli indirizzi bassi della memoria (nelle locazioni 0000:0000h ÷ 0000:03FFh); ogni elemento di tale tabella, detto *Interrupt Vector*, contiene i 32 bit dell'indirizzo di partenza della ISR relativa ad uno dei 256 differenti Interrupt riconosciuti.

Esempio:



Istruzione INT

Causa l'attivazione della routine di servizio relativa ad un certo Interrupt. Ha il seguente formato:

INT *interrupt-number*

dove *interrupt-number* identifica l'Interrupt, cioè il numero d'ordine dell'Interrupt Vector che si desidera attivare, all'interno della tabella che contiene i 256 Interrupt Vector ammessi dal sistema.

L'indirizzo fisico dell'Interrupt viene ottenuto semplicemente moltiplicando per 4 il parametro *interrupt-number*.

NB: come caso particolare si ha l'istruzione senza parametri **INTO**, che equivale a:

- una **INT 4** quando l'Overflow Flag (OF) è settato;
- una istruzione nulla, altrimenti.

Le operazioni causate da una **INT** sono:

- salvataggio nello stack del Flag Register
- azzeramento dei flag TF (Trap Flag) e IF (Int. Enable/Disable)
- salvataggio nello stack del registro CS
- caricamento in CS della seconda word dell'Interrupt Vector
- salvataggio nello stack del registro IP
- caricamento in IP della prima word dell'Interrupt Vector

Istruzione IRET

È l'istruzione conclusiva di una ISR; causa il ritorno del sistema nello stato precedente all'ultima INT e l'esecuzione delle istruzioni ad essa successive nel programma interrotto. Non ha operandi e provoca le seguenti operazioni:

- preleva dallo stack il valore di IP;
- preleva dallo stack il valore di CS;
- preleva dallo stack il valore del Register Flag.

Istruzioni per il Controllo del Processore

	OpCode	Descrizione
Modifica dei flag		
	CLC	Clear Carry Flag
	CLD	Clear Direction Flag
	CLI	Clear Interrupt-Enable Flag
	CMC	Complement Carry Flag
	STC	Set Carry Flag
	STD	Set Direction Flag
	STI	Set Interrupt Enable Flag
Sincronizzazione		
	ESC	Escape
	HLT	Halt
	LOCK	Lock the Bus
	WAIT	Wait
Istruzione nulla		
	NOP	No Operation

Servono per regolare il funzionamento del processore attraverso comandi software.

Modifica dei Flag

Permettono di modificare alcuni Flag; sono:

- **STC, CLC** e **CMC**:
  rispettivamente servono per forzare ad 1 o a 0 il Carry Flag; CMC complementa il valore del Carry Flag;
- **STD** e **CLD**:
  rispettivamente servono per forzare ad 1 o a 0 il Direction Flag;
- **STI** e **CLI**:
  rispettivamente servono per forzare ad 1 o a 0 l'Interrupt Flag;

Istruzioni di Sincronizzazione

Permettono al processore 8086 di sincronizzarsi con dispositivi esterni; sono:

- **HLT**:
  forza il processore nello stato *idle*, cioè il processore non esegue nessuna istruzione fino a che non:
  - riceve un Interrupt esterno;
  - viene attivata la linea RESET.
- **WAIT**:
  forza il processore nello stato *idle*; ogni 5 colpi di clock viene controllata la linea TEST: quando questa viene attivata, il processore procede con l'esecuzione dell'istruzione successiva alla WAIT;
- **ESC**:
  usata per inviare istruzioni al coprocessore matematico 8087;
- **LOCK**:
  è un prefisso che può precedere qualunque istruzione; si usa quando si vuole che nessun altro dispositivo utilizzi il bus durante l'esecuzione dell'istruzione prefissata. Viene utilizzata in sistemi multiprocessore per istruzioni che facciano uso di risorse critiche (memoria condivisa).

Istruzione nulla: NOP

Non esegue nessuna operazione, se non quella di far avanzare l'Instruction Pointer (IP) all'istruzione successiva.

Sue possibili applicazioni sono:

- occupare il posto di una istruzione cancellata quando si vuole modificare il codice macchina senza doverlo ri-assemblare;
- introdurre un periodo di attesa di durata nota prima dell'esecuzione dell'istruzione successiva.

Modi di Indirizzamento

Il Modo di Indirizzamento di un'istruzione definisce il metodo da utilizzare per determinare dove è memorizzato un particolare dato (operando).
Gli operandi possono essere contenuti:

- in registri;
- nell'istruzione stessa;
- in memoria;
- su una porta di I/O.

I Modi di Indirizzamento possono essere raggruppati in 7 classi:

- *Register*;
- *Immediate*;
- *Direct*;
- *Register Indirect*;
- *Base Relative*;
- *Direct Indexed*;
- *Base Indexed*.

Register Addressing

Nell'istruzione è specificato il registro da utilizzare come operando.

Formato Assembly:

$$< registro >$$

Esempio:

```
MOV AX, BX
```

Immediate Addressing

Nell’istruzione stessa è indicato il dato da utilizzare come operando.

Formato Assembly:

$$< espressione >$$

Esempi:

```
MOV  AX, 10h
MOV  BL, (4*17h)/10b
```

Note:

- L'operando può anche essere un simbolo definito mediante una pseudo-istruzione EQU.

  Esempio:

```
K     EQU     1234
      ...
      MOV     CX, K
```

- Il dato indicato nell'istruzione viene trasformato dall'assemblatore in formato binario su 8 o 16 bit, e scritto nel campo opportuno dell'istruzione macchina.

  Si noti, come regola generale, che, se il dato è su 16 bit, gli 8 bit più significativi sono scritti nel secondo byte, mentre quelli meno significativi nel primo.

Direct Addressing

Nell'istruzione è specificato il nome di una variabile, corrispondente all'Effective Address della parola di memoria da utilizzare come operando. Alla variabile può essere sommata o sottratta un'espressione.

L'indirizzo fisico è ottenuto sommando l'EA con il contenuto del Data Segment DS moltiplicato per 16.

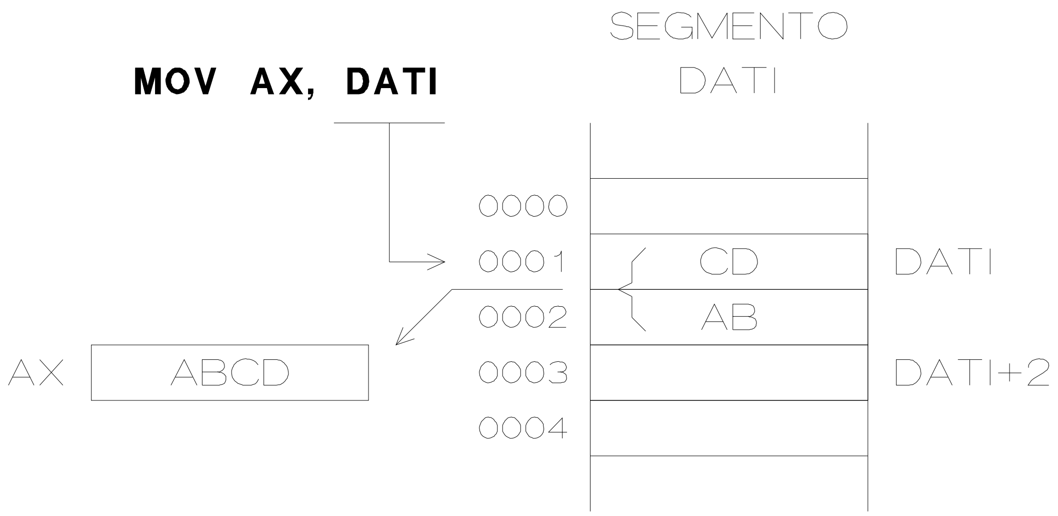
Formato Assembly:

$$< \textit{variabile} > \pm < \textit{espressione} >$$

Esempio:

```
MOV  AX, TABLE
MOV  AX, TABLE + 18
```

Esempio di Direct Addressing



Register Indirect Addressing

L'Effective Address dell'operando è contenuto in uno dei seguenti registri:

- Base;
- Index Register (DI oppure SI);
- Base Pointer (BP).

Viene detto *Indirect* perchè nell'istruzione è indicato dove trovare l'indirizzo dell'operando.

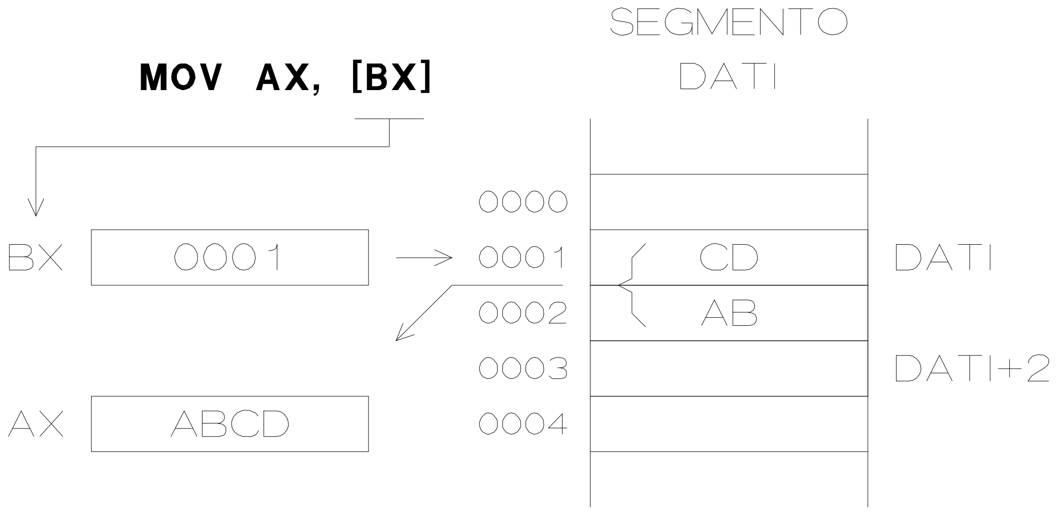
Formato Assembly:

$$[< registro >]$$

Esempio:

```
MOV  AX, [BX]
MOV  AX, [SI]
```

Esempio di Register Indirect Addressing



Esempio

Codice per il trasferimento di un vettore in un altro usando il Register Indirect Addressing.

```
      MOV   SI,   OFFSET SOURCE_VECT
      MOV   DI,   OFFSET DEST_VECT
      MOV   CX,   VECT_LENGTH
QUI:  MOV   AX,   [SI]
      MOV   [DI], AX
      ADD   SI,   2
      ADD   DI,   2
      LOOP  QUI
```

Base Relative Addressing

L'Effective Address dell'operando è calcolato sommando il contenuto di uno dei Base Register (BX o BP) ad un *displacement* rappresentato da una costante presente nell'istruzione stessa.

Formato Assembly:

$$[< registro > + < displacement >]$$

Esempio:

```
MOV AX, [BX + 4]
```

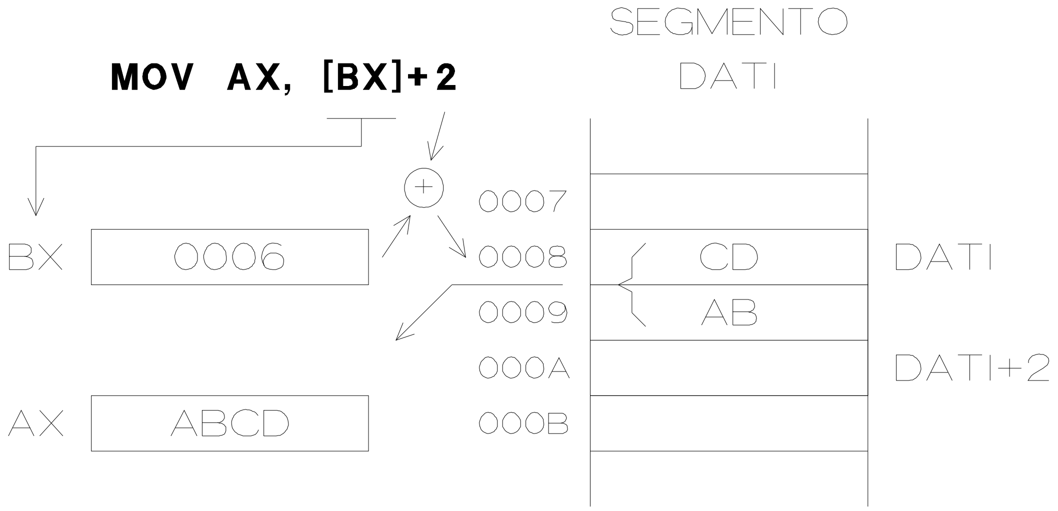
Nota:

le 3 forme

```
MOV AX, [BX + 4]
MOV AX, 4 [BX]
MOV AX, [BX] + 4
```

sono equivalenti, ma **è da preferire** la prima.

Esempio di Base Relative Addressing



Direct Indexed Addressing

L'Effective Address dell'operando è calcolato sommando il valore di un *offset*, contenuto in una variabile, ad un *displacement*, contenuto in uno degli Index Register (SI o DI).

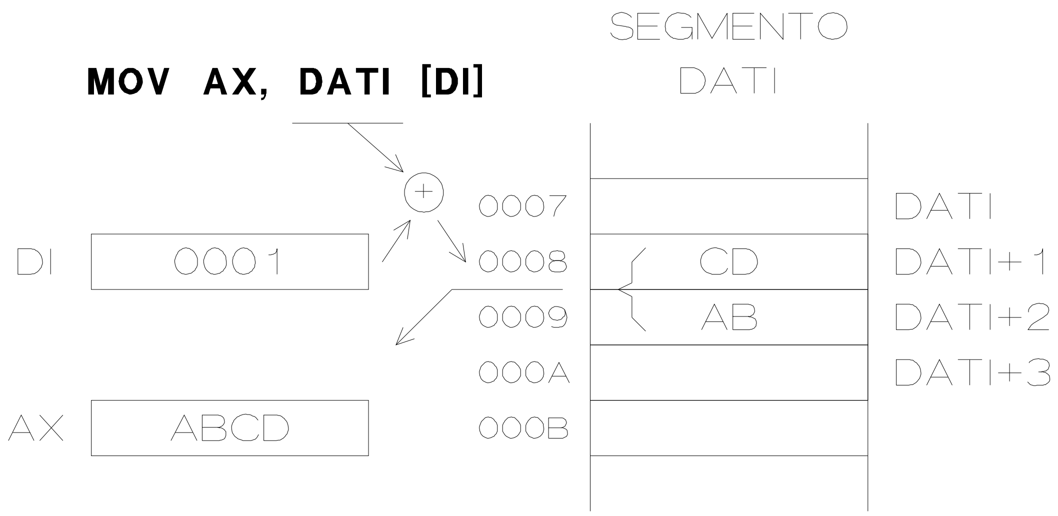
Formato Assembly:

$$< variabile > [SI]$$
$$< variabile > [DI]$$

Esempio:

```
MOV  AX, TABLE[DI]
```

Esempio di
Direct Indexed Addressing



Esempio

Codice per il trasferimento di un vettore in un altro usando il Direct Indexed Addressing.

```
        MOV     SI,   0
        MOV     CX,   VECT_LENGTH
QUI:    MOV     AX,   SOURCE_VECT [SI]
        MOV     DEST_VECT [SI], AX
        ADD     SI,   2
        LOOP    QUI
```

Base Indexed Addressing

L'Effective Address dell'operando è calcolato come somma dei seguenti termini:

- contenuto di uno dei Base Register (BX o BP);
- contenuto di uno degli Index Register (SI o DI);
- un *displacement* contenuto nell'istruzione stessa.

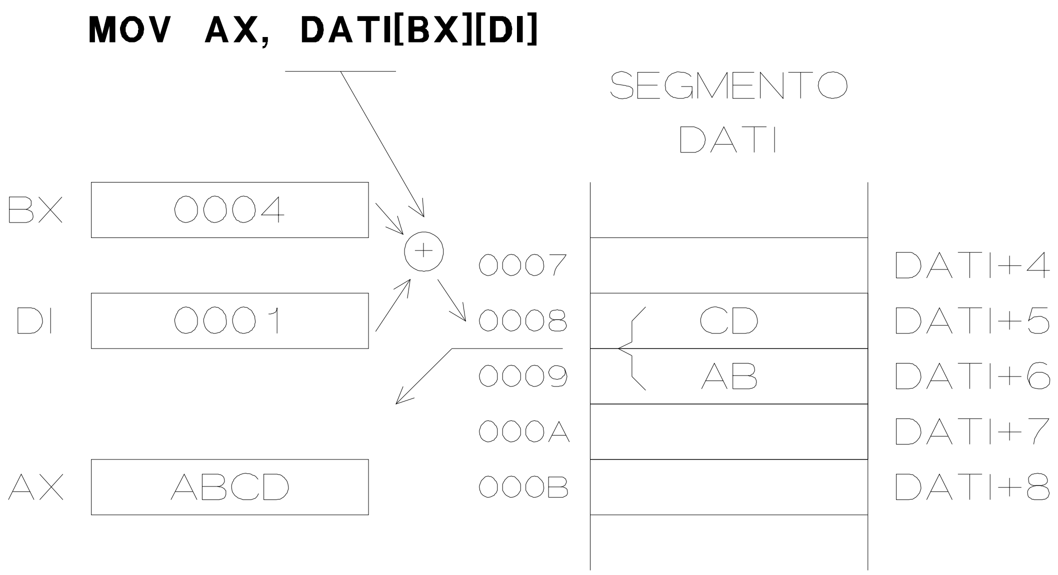
Formato Assembly:

$$< variabile > [BX] + [SI]$$
$$< variabile > [BX] + [DI]$$
$$< variabile > [BP] + [SI]$$
$$< variabile > [BP] + [DI]$$

Esempio:

```
MOV AX, ARRAY[BX][DI]
```

Esempio di Base Indexed Addressing



Segment Register Impiegato

La seguente tabella riassume i vari modi di indirizzamento con il Segment Register impiegato da ognuno.

Indirizzamento	Formato	SegReg
Register	registro	nessuno
Immediate	dato	nessuno
Direct	variabile	DS
Register Indirect	[BX]	DS
	[BP]	SS
	[DI]	DS
	[SI]	DS
Base Relative	label [BX]	DS
	label [BP]	SS
Direct Indexed	label [DI]	DS
	label [SI]	DS
Base Indexed	label [BX] [SI]	DS
	label [BX] [DI]	DS
	label [BP] [SI]	SS
	label [BP] [DI]	SS

NB: i segmenti utilizzati possono essere modificati mediante il prefisso di *Segment Override*.

Tempo di esecuzione delle istruzioni

Il tempo richiesto per eseguire un'istruzione si ricava sommando:

- il tempo di esecuzione vero e proprio (fornito dal costruttore per ogni diverso impiego di ogni istruzione)
- l'eventuale calcolo dell'Effective Address.

La seguente tabella mostra i tempi di calcolo dell'Effective Address:

Indirizzamento	Clock	Esempio
Direct	6	MOV AX, ADDR
Register Indirect	5	MOV AX, [BP]
Base Relative	9	MOV AX, ADDR[BP]
Direct Indexed BP + DI, BX + SI BP + SI, BX + DI	 7 8	 MOV AX, [BP+DI] MOV AX, [BX+DI]
Base Indexed BP+DI+Disp, BX+SI+Disp BP+SI+Disp, BX+DI+Disp	 11 12	 MOV AX, ADDR[BP+DI] MOV AX, ADDR[BP+SI]

Il prefisso di *Segment Override* aggiunge 2 colpi di clock ai tempi specificati nella tabella.

Esempio: Istruzione MOV

Operandi	Clocks byte(word)	Bytes	Esempio
reg, reg	2	2	MOV BX,CX
mem, acc	10 (14)	3	MOV MEM_DEST,AL
acc, mem	10 (14)	3	MOV AX,MEM_SOURCE
mem, reg	9 (13) + EA	2-4	MOV MEM_DEST,BX
reg, mem	8 (12) + EA	2-4	MOV BX,MEM_SOURCE
reg, imm	4	2-3	MOV BX,0F6CDh
mem, imm	10 (14) + EA	3-6	MOV MASK,0F6CDh
seg-reg, reg16	2	2	MOV DS,BX
seg-reg, mem16	8 (12) + EA	2-4	MOV DS,SEGMENT_VAL
reg16, seg-reg	2	2	MOV BP,SS
mem, seg-reg	9 (13) + EA	2-4	MOV CODE_VAR,CS

Esempio: Istruzione IDIV

Operandi	Clocks	Bytes	Esempio
reg8	101-112	2	IDIV CL
reg16	165-184	2	IDIV DX
mem8	(107-118) + EA	2-4	IDIV BYTE[SI]
mem16	(175-194) + EA	2-4	IDIV [BX].WORD_ARRAY

Pseudo-Istruzioni

Le Pseudo-Istruzioni sono comandi per l'assemblatore. Non vengono tradotte in istruzioni macchina, ma sono interpretate come operazioni che l'assemblatore deve compiere al momento dell'assemblaggio.

Vi sono circa 60 Pseudo-Istruzioni, ma verranno sommariamente analizzate solo quelle di uso comune, utilizzando come riferimento un programma dimostrativo.

LE FUNZIONI MS-DOS

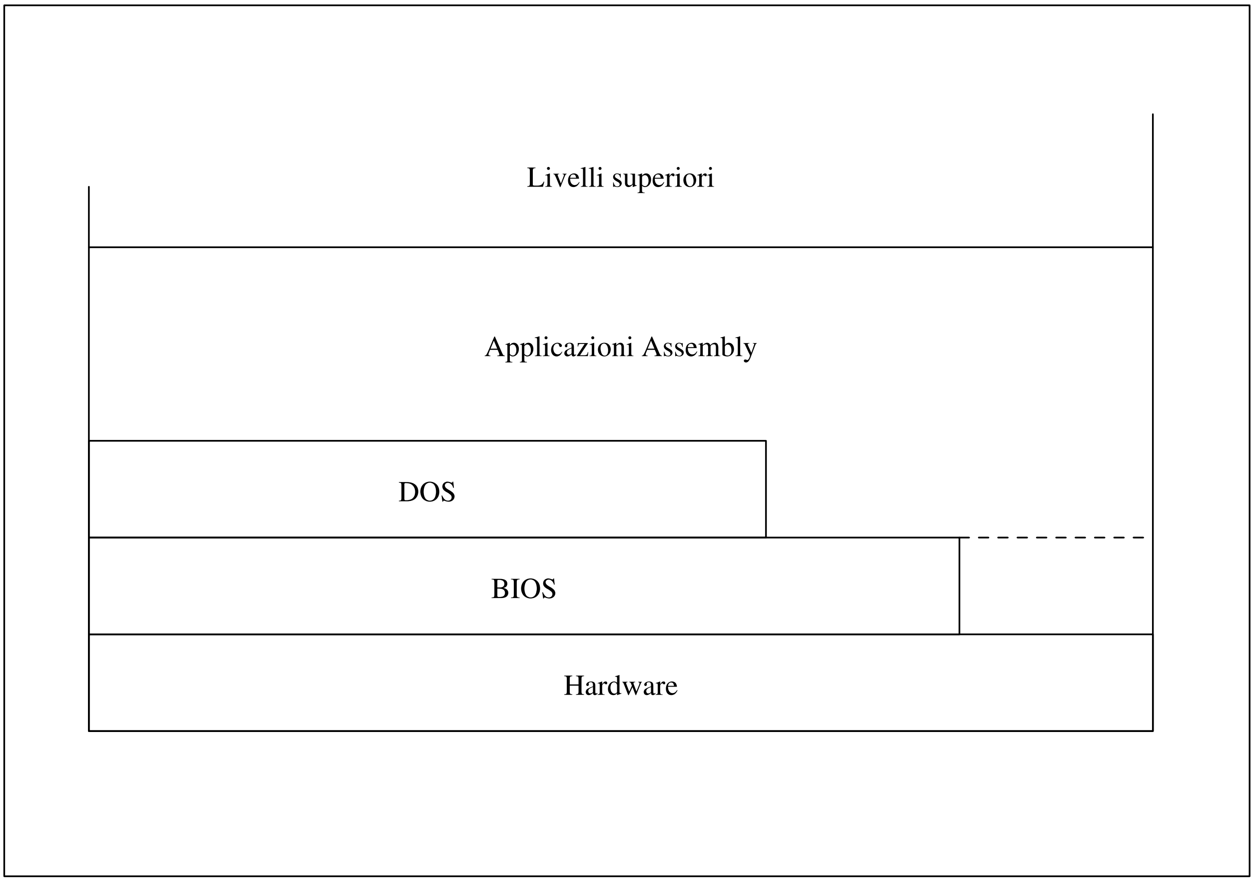
Il Sistema Operativo MS-DOS offre al programmatore assembly un insieme di *funzioni* che permettono di eseguire le più comuni operazioni di gestione del sistema, permettendo al programmatore di trascurare i dettagli relativi all'implementazione a basso livello delle operazioni richieste. Ad esempio:

- operazioni di I/O (da tastiera, schermo, stampante);
- lettura e scrittura su disco;
- mantenimento del file-system;
- gestione della configurazione del sistema (data, ora, periferiche);
- allocazione e deallocazione di aree di memoria;

Inoltre, per garantire la portabilità dei programmi su tutte le macchine dotate di MS-DOS prescindendo dall'hardware usato, ogni costruttore di hardware fornisce uno strato di software di interfaccia hw-sw. Questo è detto BIOS *(Basic Input Output System)* e permette la gestione a basso livello di:

- video (modalità grafiche, colori, palette, accensione pixels,...);
- tastiera, mouse (codice tasti, stato degli shift,...);
- stampante;
- dischi (cilindri, tracce, motore,...);
- porte seriali e parallele.

Visione Stratificata del Sistema DOS



Accesso alle funzioni DOS e BIOS

Le funzioni **DOS** sono richiamabili mediante l'istruzione **INT 21h**, e si distinguono tra loro dal valore presente nel registro AH al momento della chiamata.

Quasi tutte le funzioni DOS azzerano il Carry Flag (CF) se l'operazione si è conclusa correttamente, mentre lo forzano ad 1 se si sono verificati degli errori.

L'insieme delle funzioni disponibili varia in funzione della versione del DOS usata; è auspicabile che l'insieme delle funzioni corrispondente ad una determinata versione contenga gli insiemi delle funzioni corrispondenti alle versioni precedenti.

Le funzioni **BIOS** sono richiamabili mediante l'invocazione di un particolare Interrupt con l'istruzione **INT**; molte di esse possono fornire diversi servizi che si distinguono dal valore presente nel registro AH al momento della chiamata.

Esempio:

```
MOV   AH, 07h        ;Funzione DOS 'Read Keyboard
INT   21h            ; Without Echo'
MOV   AH, 0Ah        ;Servizio BIOS 'Write Char
INT   10h            ; at Cursor'
```

Accesso diretto alla memoria video

L'accesso diretto alla memoria video è ovviamente **sconsigliato** per problemi di compatibilità; ciononostante, talvolta è richiesto per motivi di velocità.

Le principali caratteristiche delle diverse modalità video sono:

- Modalità testo: inizia a B8000, e segue linearmente, carattere e attributo;
- Modalità grafica 320×200, 256 colori: inizia a A0000, e segue linearmente, 8 bit/pixel;
- Modalità grafica 640×480, 16 colori: inizia a A0000, e prosegue su moduli differenti, 4 bit/pixel; le 4 mappe si accedono con l'istruzione OUT.
- Modalità superiori a 256 colori o 'truecolor': iniziano a A0000, e proseguono su moduli differenti.

LA FAMIGLIA DI PROCESSORI 80X86

Verranno considerati i seguenti processori:

- Il processore 8008
- Il processore 8080
- Il processore 8085
- Il processore 8086
- Il processore 8088
- Il processore 80286
- Il processore 80386
- Il processore 80486
- Il processore Pentium
- La tecnologia MMX

Il processore 8008

- Progettato del 1971 da Intel Corp.;
- versione estesa del 4004 (a 4 bit), principalmente usato per i primi videogiochi;
- indirizzamento in memoria di 16k bytes;
- 48 istruzioni macchina.

Il processore 8080

- Introdotto nel 1973;
- è il primo processore a 8 bit;
- 10 volte più veloce dell'8008;
- indirizza 64k bytes invece di 16k;
- utilizzato per il primo personal computer (MITS Altair 8800) nel 1974.

Il processore 8085

- Nato nel 1977;
- 65% più veloce dell'8080;
- altre case hanno sviluppato processori compatibili con l'8085 (Zilog Z80);
- l'ultimo processore a 8 bit, venduto in 700 milioni di copie.

Il processore 8086

- Nato nel 1978;
- processore a 16 bit;
- 2.5 milioni di istruzioni per secondo (2.5 MIPS);
- indirizza 1 M Byte di memoria;
- per la prima volta un processore comprende un'unità di prefetching;
- include operazioni complesse come la moltiplicazione e la divisione.

Il processore 8088

- Nato 1 anno dopo l'8086;
- l'8086 e l'8088 differiscono in quanto il primo lavora su 16 bit a livello sia di struttura interna sia di connessioni esterne, mentre il secondo è un processore esternamente ad 8 bit ed internamente identico all'8086 (16 bit);
- l'insieme delle istruzioni riconosciute dall'8086 coincide con quello dell'8088, portando completa compatibilità;
- l'8088 è stato affiancato all'8086 perchè rende possibile l'impiego di periferici ad 8 bit.
- nel 1981 IBM include l'8088 nel proprio personal computer.

Il processore 80286

- Rappresenta l'evoluzione dell'8086, introdotto nel 1983;
- indirizza 16 M byte di memoria;
- l'instruction set è praticamente identico a quello dell'8086 a meno di poche istruzioni per l'utilizzo dei rimanenti 15 M byte di memoria;
- viene introdotta la modalità protetta;
- 4.0 MIPS a 8.0 MHz.

Il processore 80386

- Viene introdotto nel 1986;
- è un grosso passo in avanti rispetto all’80286;
- è un processore a 32 bit (sia per il bus dei dati sia per l‘accesso alla memoria);
- indirizza 4 G byte di memoria;
- nascono diverse versioni dell’80386: 386SL, 386DX, 386EX, 386SLC,...
- include istruzioni per il memory management (che i processori precedenti demandavano completamente al software);
- è compatibile con i precedenti, quindi può eseguire il codice per i processori a 16 bit.

Il processore 80486

- Nel 1989 viene introdotto l'80486;
- non è molto differente dal precedente se non per l'inclusione della parte matematica (80387) e di 8 k bytes di cache;
- funziona a 50 MHz, e arriva a 50 MIPS (50% più veloce dell'80386);
- anche in questo caso escono versioni diverse: 486SX, 486DX, 486DX2, 486DX4,...

Il processore Pentium

- Nasce nel 1993;
- non viene chiamato 80586 per problemi di copyright su numeri;
- la prima versione funziona a 60 MHz, e produce 110 MIPS;
- la cache viene aumentata da 8 a 16 k bytes;
- il data bus viene portato a 64 bit;
- la principale caratteristica è la comparsa di una doppia *pipeline* per le operazioni su interi;
- inoltre viene implementata una politica di *branch prediction*.

La tecnologia MMX

- Viene introdotta per la prima volta sul processore Pentium;
- MMX (MultiMedia eXtension) serve per migliorare la gestione di particolari tipi di dati;
- con un piccolo aumento dell'area del processore, vengono implementate numerose nuove funzioni (aritmetica a saturazione);
- i registri MMX registri sono sovrapposti ai registri floating-point;
- sfrutta modalità di elaborazione SIMD suddividendo i 64 bit in pacchetti.

Modalità reale e protetta

- Il modo di funzionamento del processore visto fin'ora è detto *Modalità reale*;
- Per accedere alla memoria ad indirizzi maggiori di 1 M byte occorre un nuovo meccanismo, chiamato *Modalità protetta*.

In modalità protetta esiste ancora il concetto di *offset*, ma non più quello di *segmento*: il registro di segmento contiene un *selettore*.

Il *selettore* è usato per selezionare un *descrittore* in una tablella di descrittori.

Il *descrittore* precisa la zona di memoria: segmento, lunghezza, e diritti di accesso. Quindi il registro di segmento è ancora usato per selezionare un segmento, ma questa volta in maniera indiretta.

I programmi scritti per funzionare in modalità reale funzionano ancora anche in modalità protetta, con l'accortezza di inserire il valore corretto nel descrittore della zona di memoria utilizzata.

Ci sono due tabelle di 8192 descrittori l'una; la prima per descrittori validi per tutti i programmi (*Global descriptors*), la seconda per descrittori validi solo per particolari applicazioni (*Local descriptors*).

CREAZIONE DI UN PROGRAMMA ASSEMBLY

Il processo di creazione di un programma Assembly passa attraverso le seguenti fasi:

- scrittura, tramite un normale *EDITOR*, di uno o più file di caratteri ASCII con estensione **.ASM** contenenti il programma *sorgente*.
- assemblaggio, tramite un *ASSEMBLATORE*, dei singoli file sorgente, e generazione di altrettanti file *oggetto*, con estensione **.OBJ**.
- creazione del file *eseguibile*, con estensione **.EXE**, tramite un *LINKER*.
- verifica e correzione degli eventuali errori tramite un *DEBUGGER*.

Suddivisione in moduli differenti

Quando i programmi assumono dimensioni ragguardevoli, risulta vantaggioso suddividerli in più moduli. I vantaggi che si ottengono da questa suddivisione sono principalmente:

- essi possono essere maneggiati con più facilità;
- quando si modifica un solo modulo è possibile assemblare solo quel modulo.

Le direttive utilizzate sono:

- PUBLIC: comunica all'assemblatore di rendere disponibili ad altri moduli le etichette ad essa associate;
- EXTERN: viene utilizzata per rendere disponibili nel modulo corrente le etichette dichiarate pubbliche in altri moduli;
- GLOBAL: serve per sostituire entrambe le precedenti.

Esempio: file 1

```
        PUBLIC  ARRAY
ARRAY     DB    100
```

```
        GLOBAL  ARRAY:BYTE
ARRAY     DB    100
```

Esempio: file 2

```
        EXTERN  ARRAY:BYTE
        MOV   [ARRAY], AX
```

```
        GLOBAL  ARRAY:BYTE
        MOV   [ARRAY], AX
```

L'Assemblatore

L'Assemblatore trasforma i file di caratteri contenenti il programma *sorgente* (.ASM) in altrettanti file oggetto (.OBJ) contenenti il codice in linguaggio macchina.

Il processo di assemblaggio viene compiuto in due passi:

- il primo passo definisce il valore dell'offset relativo ad ogni riga ed ogni simbolo nel codice sorgente;
- il secondo passo traduce le istruzioni Assembly in istruzioni macchina e genera il file oggetto.

Il Linker

Il Linker combina i moduli oggetto e produce un unico file *eseguibile* (.EXE); in particolare:

- unisce i moduli oggetto, risolvendo i riferimenti a simboli esterni;
- ricerca i file di libreria contenenti le procedure esterne utilizzate dai vari moduli;
- produce un modulo rilocabile ed eseguibile.

Nota: il passo di *Linking* deve essere effettuato anche se il programma è composto da un solo modulo oggetto.

Il Debugger

Il Debugger è un ambiente di verifica e messa a punto dei programmi Assembly. Esso permette di:

- visualizzare, ed eventualmente modificare, il contenuto dei registri e della memoria;
- eseguire passo-passo un programma;
- inserire delle *breakpoint* in un programma;
- caricare in memoria file o settori di disco, e viceversa;
- assemblare e disassemblare un programma.

Struttura e Documentazione di un Programma Assembly

Tipicamente un programma Assembly è formato da:

- il segmento contenente i dati (DATASEG):
  contiene le pseudo-istruzioni per l'allocazione delle variabili e le definizioni delle costanti; i commenti che lo corredano sono relativi alla spiegazione dei simboli utilizzati;
- il segmento contenente lo stack (STACKSEG):
  è formato da una sola pseudo-istruzione per l'allocazione dello spazio sufficiente per lo stack del programma; non necessita di commenti aggiuntivi;
- il segmento contenente il codice (CODESEG):
  è la parte principale del file .ASM e necessita di commenti molto chiari ed esaurienti; inoltre è **consigliabile** che tutti utilizzino una strutturazione standardizzata delle procedure.

Al fine della documentazione delle procedure, esse sono suddivisibili nelle seguenti classi:

- Procedura principale (o MAIN)
- Procedure di medio livello
- Procedure di basso livello
  - di interfaccia
  - di calcolo

Strutturazione delle procedure

Alcune regole per la standardizzazione della codifica delle procedure sono:

- Il corpo principale del programma (MAIN) deve essere corto, e formato *esclusivamente* da chiamate a procedure; inoltre, deve comparire come prima procedura all'interno del CODESEG. I commenti devono servire per individuare lo scopo di ciascuna chiamata. Le prime 3 righe **devono** essere sempre presenti per inserire nello stack l'indirizzo corretto per ritornare al sistema operativo.
- Ogni altra procedura deve essere preceduta da un'indicazione delle operazioni che esegue, dei parametri e delle variabili che utilizza, dei registri e dei flag che modifica, e infine dei codici di ritorno che riporta: esempio:

```
;------------------------------------------------------------------+
; Procedura per l'invio su stampante di un numero compreso tra 0 e 99 |
;                                                                  |
; PARAMETRI: il numero e' memorizzato in AL                        |
; VARIABILI: il numero viene stampato in decimale se nella locazione |
;            DS:NOTAZ e' memorizzato 1, altrimenti in esadecimale  |
; REGISTRI UTILIZZATI: AL, BX, CX, SI                              |
; CODICI DI RITORNO: CF=0 se l'operazione e' terminata con successo, |
;            CF=1 se vi e' stato un errore; inoltre con CF=1, si ha: |
;              AH=0: numero fuori range                            |
;              AH=1: stampante non in linea                        |
;              AH=2: carta terminata                               |
;------------------------------------------------------------------+
```

- Ogni procedura di medio livello deve essere corta e deve servirsi di altre chiamate a procedura.
- Le procedure di basso livello devono essere riunite nella parte finale del codice (meglio se in un modulo differente).
- Le chiamate agli Interrupt devono essere presenti *solo* nelle procedure di basso livello di interfaccia.

APPENDICE A: Programma di Esempio

```
            TITLE   -  Esercitazione di Calcolatori Elettronici

DSEG              SEGMENT PARA PUBLIC 'DATA'
PAGINA            EQU    00h
LF                EQU    0Ah
CR                EQU    0Dh
HT                EQU    09h
TITOLO            DB     'CORSO di CALCOLATORI ELETTRONICI$'
ISTRUZIONI        DB     CR,HT,HT,HT,HT,HT,CR,'Premere un tasto: $'
CODICE_TASTO      DB     ?
BIN_MESS          DB     CR,LF,HT,HT,'Rappresentazione Binaria: $'
DEC_MESS          DB     CR,LF,HT,HT,'Rappresentazione Decimale: $'
TERMINE           DB     CR,LF,LF,HT,'Un''altra iterazione? [S/N]$'
DSEG              ENDS


STACKM            SEGMENT PARA STACK 'STACK'  ;Viene allocata una zona di
                  DB      64 DUP('12345678')  ; memoria per lo Stack: in
STACKM            ENDS                        ; tutto 64*8 bytes.


                  ASSUME CS:CSEG,DS:DSEG,SS:STACKM
CSEG              SEGMENT PARA PUBLIC 'CODE'


;-----------------------------------------------------------------;
;                 Corpo principale del programma                  ;
;-----------------------------------------------------------------;

MAIN              PROC   FAR
                  PUSH   DS              ;Istruzioni da lasciare SEMPRE
                  MOV    AX,00h          ; al principio dei programmi!
                  PUSH   AX              ;
```

```
                CALL   INIZIALIZZAZIONE
CICLO_PRINCIPALE: CALL   PROMPT
                CALL   LETTURA_DATI
                CALL   STAMPA_DEC
                CALL   STAMPA_BIN
                CALL   TEST_FINALE
                JNZ    CICLO_PRINCIPALE
                RET                     ;Ritorno al Sistema Operativo
MAIN             ENDP


;-----------------------------------------------------------------;
;                 Procedura di inizializzazione                    ;
;                                                                  ;
; REGISTRI UTILIZZATI: AX, DX, DS                                  ;
;-----------------------------------------------------------------;

INIZIALIZZAZIONE  PROC  NEAR
                MOV    AX,DSEG          ;Inizializzazione segmento dati
                MOV    DS,AX            ; tramite il registro AX.

                MOV    AH,00h           ;Servizio BIOS 'Set Video Mode':
                MOV    AL,03h           ; modo testo 80 x 25, colori
                INT    10h              ;(Era sufficiente un MOV AX,3)

                MOV    DX,0315h         ;Imposta riga (DH) e colonna (DL)
                CALL   SPOSTA_CURSORE   ;Muove il cursore nella pos scelta

                MOV    DX,OFFSET TITOLO ;Sceglie la stringa (DS:DX)
                CALL   STAMPA_STRINGA   ; e la stampa.

                MOV    DX,0909h         ;Imposta riga (DH) e colonna (DL)
                CALL   SPOSTA_CURSORE   ;Muove il cursore nella pos scelta
                RET                     ;Ritorno alla procedura chiamante
INIZIALIZZAZIONE  ENDP


;-----------------------------------------------------------------;
;            Procedura per stampare il messaggio iniziale          ;
;                                                                  ;
; REGISTRI UTILIZZATI: DX, AH                                      ;
;-----------------------------------------------------------------;

PROMPT            PROC  NEAR
```

```
                MOV   DX,OFFSET ISTRUZIONI ;Sceglie la stringa (DS:DX)
                CALL  STAMPA'STRINGA  ; e la stampa.
                RET                   ;Ritorno alla procedura chiamante
PROMPT          ENDP


;----------------------------------------------------------------;
;   Procedura per stampare un numero in formato binario          ;
;                                                                ;
; VARIABILI: il numero e' memorizzato nella locazione CODICE'TASTO ;
; REGISTRI UTILIZZATI: AX, BL, CX, DX                            ;
;----------------------------------------------------------------;

STAMPA'BIN      PROC  NEAR
                MOV   DX,OFFSET BIN'MESS ;Sceglie la stringa (DS:DX)
                CALL  STAMPA'STRINGA  ; e la stampa.
                MOV   DL,[CODICE'TASTO] ;Carica il codice del tasto
                MOV   DH,08h            ;Contatore dei bits
CICLO'BINARIO:  XOR   AL,AL             ;Azzera il registro AL
                SHL   DL,01h            ;Shift Logico a Sinistra
                ADC   AL,AL             ;In AL si pone il CARRY
                CALL  STAMPA'NUMERO     ;Stampa la cifra binaria
                DEC   DH                ;Decrementa il contatore del ciclo
                JNZ   CICLO'BINARIO     ;Se non e' 0 itera un'altra volta
                RET                     ;Ritorno alla procedura chiamante
STAMPA'BIN      ENDP


;----------------------------------------------------------------;
;   Procedura per stampare un numero in formato decimale         ;
;                                                                ;
; VARIABILI: il numero e' memorizzato nella locazione CODICE'TASTO ;
; REGISTRI UTILIZZATI: AX, BX, DX                                ;
;----------------------------------------------------------------;

STAMPA'DEC      PROC  NEAR
                MOV   DX,OFFSET DEC'MESS ;Sceglie la stringa (DS:DX)
                CALL  STAMPA'STRINGA  ; e la stampa.
                MOV   AL,[CODICE'TASTO] ;Carica il codice del tasto
                XOR   AH,AH             ;Azzera il registro AH
                MOV   BL,0Ah            ;Setta il dividendo (10 Decimale)
                IDIV  BL                ;Divisione tra interi
                MOV   DX,AX             ;Salva i risultati in DX per dopo
                CMP   AL,0Ah            ;Ci sono anche le centinaia?
```

```
                JB      DUE˙CIFRE        ;Se non ci sono, salta avanti
                XOR     AH,AH            ;Azzera il registro AH
                IDIV    BL               ;Divisione tra interi
                MOV     DL,AH            ;Salva il risultato in DL per dopo
                CALL    STAMPA˙NUMERO    ;Stampa le centinaia se necessario
                MOV     AL,DL            ;Decine
DUE˙CIFRE:      CALL    STAMPA˙NUMERO    ;Stampa le decine
                MOV     AL,DH            ;Unita'
                CALL    STAMPA˙NUMERO    ;Stampa le unita'
                RET                      ;Ritorno alla procedura chiamante
STAMPA˙DEC      ENDP


;------------------------------------------------------------------;
;       Procedura per determinare se occorre un'altra iterazione   ;
;                                                                  ;
; REGISTRI UTILIZZATI: AX, DX                                      ;
; CODICI DI RITORNO:                                               ;
;               Zero flag = 0  -¿ un'altra iterazione              ;
;               Zero flag = 1  -¿ esci dal programma               ;
;------------------------------------------------------------------;

TEST˙FINALE     PROC  NEAR
                MOV     DX,OFFSET TERMINE ;Sceglie la stringa (DS:DX)
                CALL    STAMPA˙STRINGA   ; e la stampa.
                CALL    LETTURA˙SN       ;Legge da tastiera senza echo.
                CMP     AL,'n'           ;Modifica il FLAG 'Z'
                RET                      ;Ritorno alla procedura chiamante
TEST˙FINALE     ENDP


;******************************************************************;
;*              Procedure di basso livello di interfaccia         *;
;******************************************************************;


;------------------------------------------------------------------;
;          Procedura per stampare un numero da 0 a 9               ;
;                                                                  ;
; PARAMETRI: il numero e' memorizzato in AL                        ;
; VARIABILI: il numero della pagina e` memorizzato in PAGINA       ;
; REGISTRI UTILIZZATI: AX, CX, BL                                  ;
;------------------------------------------------------------------;
```

```
STAMPA_NUMERO     PROC  NEAR
                  ADD   AL,'0'          ;La base da cui partire e' '0'
                  MOV   AH,0Ah          ;Servizio BIOS 'Write Char'
                  MOV   BH,PAGINA       ;Pagina attiva
                  MOV   CX,0001h        ;Stampa un solo carattere
                  INT   10h
                  RET                   ;Ritorno alla procedura chiamante
STAMPA_NUMERO     ENDP


;-----------------------------------------------------------------;
;     Procedura per leggere da tastiera uno dei tasti 'S' o 'N'   ;
;     ritorna solo dopo aver letto almeno uno dei due tasti       ;
;                                                                 ;
; REGISTRI UTILIZZATI: AX                                         ;
; CODICI DI RITORNO: in AL il codice ASCII letto                  ;
;-----------------------------------------------------------------;

LETTURA_SN        PROC  NEAR
NUOVA_LETTURA:    MOV   AH,07h          ;Servizio DOS 'Read Keyboard Char
                  INT   21h             ; Without Echo'
                  OR    AL,20h          ;Converte in minuscolo
                  CMP   AL,'n'          ;Se il tasto premuto e' 'N' esce
                  JZ    FINE_LETTURA    ; dalla procedura
                  CMP   AL,'s'          ;Se non e' 'S', ne legge
                  JNZ   NUOVA_LETTURA   ; un altro
FINE_LETTURA:     RET                   ;Ritorno alla procedura chiamante
LETTURA_SN        ENDP


;-----------------------------------------------------------------;
;                 Procedura per spostare il cursore               ;
;                                                                 ;
; PARAMETRI: le coordinate della posizione del cursore sono memoriz- ;
;            zate nel registro DX: DH riga, DL colonna.           ;
; VARIABILI: il numero della pagina e` memorizzato in PAGINA      ;
; REGISTRI UTILIZZATI: AH, BH                                     ;
;-----------------------------------------------------------------;

SPOSTA_CURSORE    PROC  NEAR
                  MOV   BH,PAGINA       ;Pagina video attiva.
                  MOV   AH,02h          ;Servizio BIOS 'Set Cursor
                  INT   10h             ; Position'
                  RET                   ;Ritorno alla procedura chiamante
```

```
SPOSTA_CURSORE      ENDP



;------------------------------------------------------------------;
;   Procedura per leggere un tasto e memorizzarlo in CODICE_TASTO      ;
;                                                                      ;
; PARAMETRI: il numero e' memorizzato in CODICE_TASTO                  ;
; REGISTRI UTILIZZATI: AX                                              ;
;------------------------------------------------------------------;

LETTURA_DATI        PROC   NEAR
                    MOV    AH,01h             ;Servizio DOS 'Read Keyboard Char'
                    INT    21h               ;Memorizza il carattere alla loca-
                    MOV    [CODICE_TASTO],AL  ;zione CODICE_TASTO.
                    RET                      ;Ritorno alla procedura chiamante
LETTURA_DATI        ENDP


;------------------------------------------------------------------;
;                   Procedura per stampare una stringa                 ;
;                                                                      ;
; PARAMETRI: l'indirizzo della stringa e` memorizzato in DS:DX         ;
; REGISTRI UTILIZZATI: AH, DX                                          ;
;------------------------------------------------------------------;

STAMPA_STRINGA      PROC   NEAR
                    MOV    AH,09h             ;Servizio DOS 'Print String'; la
                    INT    21h               ; stringa e' puntata da DS:DX.
                    RET                      ;Ritorno alla procedura chiamante
STAMPA_STRINGA      ENDP


CSEG                ENDS
                    END    MAIN
```

APPENDICE B: Esempio di Documentazione dell'Istruzione AND

AND Logical AND

Flags:	O	D	I	T	S	Z	A	P	C
	0				*	*	?	*	0

AND destination,source

Logic: destination ¡- destination AND source

AND performs a bit-by-bit logical AND operation on its operands and stores the result in destination. The operands may be words or bytes.

AND Instruction Logic

Destination	Source	Result
0	0	0
0	1	0
1	0	0
1	1	1

AND sets each bit of the result to 1 if both of the corresponding bits of the operands are 1.

Operands	Clocks byte(word)	Transfers	Bytes	Example
register, register	3	-	2	AND AL,DL
register, immediate	4	-	3-4	AND CX,0FFh
accumulator, immediate	4	-	2-3	AND AX,01000010b
register, memory	9(13) + EA	1	2-4	AND CX,MASK
memory, register	16(24) + EA	2	2-4	AND VALUE,BL
memory, immediate	17(25) + EA	2	3-6	AND STATUS,01h

APPENDICE C:
Instruction Set della CPU INTEL 8086

AAA	ASCII Adjust after Addition
AAD	ASCII Adjust before Division
AAM	ASCII Adjust after Multiply
AAS	ASCII Adjust after Subtraction
ADC	Add with Carry
ADD	Addition
AND	Logical AND
CALL	Call Procedure
CBW	Convert Byte to Word
CLC	Clear Carry Flag
CLD	Clear Direction Flag
CLI	Clear Interrupt-Enable Flag
CMC	Complement Carry Flag
CMP	Compare
CMPS	Compare String (Byte or Word)
CMPSB	Compare String Byte
CMPSW	Compare String Word
CWD	Convert Word to Doubleword
DAA	Decimal Adjust after Addition
DAS	Decimal Adjust after Subtraction
DEC	Decrement
DIV	Divide, Unsigned
ESC	Escape
HLT	Halt
IDIV	Integer Divide, Signed
IMUL	Integer Multiply, Signed
IN	Input Byte or Word
INC	Increment
INT	Interrupt
INTO	Interrupt on Overflow
IRET	Interrupt Return
JA	Jump If Above
JAE	Jump If Above or Equal

JB	Jump If Below
JBE	Jump If Below or Equal
JC	Jump If Carry
JCXZ	Jump if CX Register Zero
JE	Jump If Equal
JG	Jump If Greater
JGE	Jump If Greater or Equal
JL	Jump If Less
JLE	Jump If Less or Equal
JMP	Jump Unconditionally
JNA	Jump If Not Above
JNAE	Jump If Not Above or Equal
JNB	Jump If Not Below
JNBE	Jump If Not Below or Equal
JNC	Jump If No Carry
JNE	Jump If Not Equal
JNG	Jump If Not Greater
JNGE	Jump If Not Greater or Equal
JNL	Jump If Not Less
JNLE	Jump If NOt Less or Equal
JNO	Jump If No Overflow
JNP	Jump If No Parity
JNS	Jump If No Sign
JNZ	Jump If Not Zero
JO	Jump If Overflow
JP	Jump If Parity
JPE	Jump If Parity Even
JPO	Jump If Parity Odd
JS	Jump If Sign
JZ	Jump If Zero
LAHF	Load Register AH from
LDS	Load Pointer Using DS
LEA	Load Effective Address
LES	Load Pointer Using ES
LOCK	Lock the Bus
LODS	Load String (Byte or Word)
LODSB	Load String Byte
LODSW	Load String Word
LOOP	Loop on Count
LOOPE	Loop While Equal
LOOPNE	Loop While Not Equal
LOOPNZ	Loop While Not Zero
LOOPZ	Loop While Zero
MOV	Move (Byte or Word)

MOVS	Move String (Byte or Word)
MOVSB	Move String Byte
MOVSW	Move String Word
MUL	Multiply, Unsigned
NEG	Negate
NOP	No Operation
NOT	Logical NOT
OR	Logical OR
OUT	Output to Port
POP	Pop a Word from the Stack
POPF	Pop Flags from the Stack
PUSH	Push Word onto Stack
PUSHF	Push Flags onto Stack
RCL	Rotate through Carry Left
RCR	Rotate through Carry Right
REP	Repeat
REPE	Repeat While Equal
REPNE	Repeat While Not Equal
REPNZ	Repeat While Not Zero
REPZ	Repeat While Zero
RET	Return from Procedure
ROL	Rotate Left
ROR	Rotate Right
SAHF	Store Register AH into
SAL	Shift Arithmetic Left
SAR	Shift Arithmetic Right
SBB	Subtract with Borrow
SCAS	Scan String (Byte or Word)
SCASB	Scan String Byte
SCASW	Scan String Word
SHL	Shift Logical Left
SHR	Shift Logical Right
STC	Set Carry Flag
STD	Set Direction Flag
STI	Set Interrupt Enable Flag
STOS	Store String (Byte or Word)
STOSB	Store String Byte
STOSW	Store String Word
SUB	Subtract
TEST	Test
WAIT	Wait
XCHG	Exchange Registers
XLAT	Translate
XOR	Exclusive OR

APPENDICE D: Funzioni DOS (INT 21h)

00h (0)	Terminate Program
01h (1)	Read Keyboard Character and Echo
02h (2)	Character Output
03h (3)	Auxiliary Character Input
04h (4)	Auxiliary Character Output
05h (5)	Printer Character Output
06h (6)	Direct Console Character I/O
07h (7)	Direct Console Character Input w/o Echo
08h (8)	Console Character Input without Echo
09h (9)	Print String
0Ah (10)	Buffered Input
0Bh (11)	Check Standard Input Status
0Ch (12)	Clear Input Buffer, then Invoke Function
0Dh (13)	Disk Reset
0Eh (14)	Select Default Drive
0Fh (15)	Open File, Using FCBs
10h (16)	Close File, Using FCBs
11h (17)	Search for First Matching File, with FCBs
12h (18)	Search for Next Matching File, with FCBs
13h (19)	Delete File, Using FCBs
14h (20)	Sequential Read, Using FCBs
15h (21)	Sequential Write, Using FCBs
16h (22)	Create File, using FCBs
17h (23)	Rename File, using FCBs
18h (24)	Reserved
19h (25)	Get Current Disk
1Ah (26)	Set Disk Transfer Address (DTA)
1Bh (27)	Get FAT Information for Current Drive
1Ch (27)	Get FAT Information for Specified Drive
1Dh (29)	Reserved
1Eh (30)	Reserved
1Fh (31)	Reserved
20h (32)	Reserved
21h (33)	Random Read, Using FCBs

22h (34)	Random Write, Using FCBs
23h (35)	Get File Size, Using FCBs
24h (36)	Set Random-Record Field, Using FCBs
25h (37)	Set Interrupt Vector
26h (38)	Create Program Segment
27h (39)	Random Block Read, Using FCBs
28h (40)	Random Block Write, Using FCBs
29h (41)	Parse Filename
2Ah (42)	Get System Date
2Bh (43)	Set System Date
2Ch (44)	Get System Time
2Dh (45)	Set System Time
2Eh (46)	Set or Reset VERIFY Switch
2Fh (47)	Get Disk Transfer Address (DTA)
30h (48)	Get DOS Version Number
31h (49)	Terminate and Stay Resident
32h (50)	Reserved
33h (51)	Get or Set Ctrl-Break Status
34h (52)	Reserved
35h (53)	Get Interrupt Vector
36h (54)	Get Disk Free Space
37h (55)	Reserved
38h (56)	Get or Set Country-Dependent Information
39h (57)	Create Directory (MKDIR)
3Ah (58)	Remove Directory (RMDIR)
3Bh (59)	Change Directory (CHDIR)
3Ch (60)	Create a File (CREAT)
3Dh (61)	Open a File
3Eh (62)	Close a File Handle
3Fh (63)	Read from File or Device, Using a Handle
40h (64)	Write to File or Device, Using a Handle
41h (65)	Delete File (UNLINK)
42h (66)	Move File Pointer (LSEEK)
43h (67)	Get or Set File Attributes (CHMOD)
4400h (68-0)	IOCTL: Get Device Information
4401h (68-1)	IOCTL: Set Device Information
4402h (68-2)	IOCTL: Read from Character Device
4403h (68-3)	IOCTL: Write to Character Device
4404h (68-4)	IOCTL: Read from Block Device
4405h (68-5)	IOCTL: Write to Block Device
4406h (68-6)	IOCTL: Get Input Status
4407h (68-7)	IOCTL: Get Output Status
4408h (68-8)	IOCTL: Is Device Removable?
4409h (68-9)	IOCTL: Is Logical Device Remote?

440Ah (68-10)	IOCTL: Is Handle Remote?
440Bh (68-11)	IOCTL: Change Sharing Retry Count
440Dh (68-13)	IOCTL: Generic IOCTL Request
440Eh (68-14)	IOCTL: Get Logical Drive
440Fh (68-15)	IOCTL: Set Logical Drive
45h (69)	Duplicate a File Handle (DUP)
46h (70)	Force Handle Duplication (FORCDUP)
47h (71)	Get Current Directory
48h (72)	Allocate Memory
49h (73)	Free Allocated Memory
4Ah (74)	Modify Memory Allocation (SETBLOCK)
4Bh (75)	Load or Execute a Program (EXEC)
4Ch (76)	Terminate a Process (EXIT)
4Dh (77)	Get Return Code of a Subprocess (WAIT)
4Eh (78)	Find First Matching File (FIND FIRST)
4Fh (79)	Find Next Matching File (FIND NEXT)
50h (80)	Reserved
51h (81)	Reserved
52h (82)	Reserved
53h (83)	Reserved
54h (84)	Get VERIFY Setting
55h (85)	Reserved
56h (86)	Rename a File
57h (87)	Get or Set a File's Date and Time
59h (89)	Get Extended Error Information
5Ah (90)	Create Unique File
5Bh (91)	Create New File
5Ch (92)	Lock/Unlock File Access
5Dh (93)	Reserved
5E00h (94-0)	Get Machine Name
5E02h (94-2)	Set Printer Setup
5E03h (94-3)	Get Printer Setup
5F02h (95-2)	Get Redirection List Entry
5F03h (95-3)	Redirect Device
5F04h (95-4)	Cancel Redirection
60h (96)	Reserved
61h (97)	Reserved
62h (98)	Get PSP Address

APPENDICE E: Funzioni BIOS

INT 00h (0)	Divide by 0
INT 01h (1)	Single Step
INT 02h (2)	Non-Maskable Interrupt (NMI)
INT 03h (3)	Breakpoint
INT 04h (4)	Overflow
INT 05h (5)	Print Screen
INT 08h (8)	System Timer
INT 09h (9)	Keyboard
INT 10h (16)	Video and Screen Services
INT 11h (17)	Read Equipment-List
INT 12h (18)	Report Memory Size
INT 13h (19)	Disk I/O Services, Floppy and Hard Disks
INT 14h (20)	Serial I/O Services (Comm. Ports)
INT 15h (21)	Cassette and Extended Services
INT 16h (22)	Keyboard I/O Services
INT 17h (23)	Printer I/O Services
INT 18h (24)	BASIC Loader Service
INT 19h (25)	Bootstrap Loader Service
INT 1Ah (26)	System Timer and Clock Services
INT 1Bh (27)	Keyboard Break
INT 1Ch (28)	User Timer Tick
INT 4Ah (74)	User Alarm
INT 70h (112)	Real-Time Clock

APPENDICE F: Pseudo-Istruzioni del MACRO ASSEMBLER

!	Literal-Character Operator
$	Location Counter Operand
%	Expression Operator
&	Substitute Operator
*	Multiplication
+	Addition or Unary Plus
-	Subtraction or Unary Minus
.	Structure Field-Name Operator
/	Division
:	Segment-Override Operator
;;	Macro Comment
¡¿	Literal-Text Operator
[]	Index Operator
=	Create Absolute Symbol
.186	Enable 80186 Instructions
.286c	Enable Real Mode 80286 Instructions
.286p	Enable Protected Mode 80286 Instructions
.287	Enable 80287 Instructions
.8086	Enable 8086 Instructions
.8087	Enable 8087 Instructions
AND	Bitwise Logical AND
ASSUME	Associate Segment with Segment Register
AT	Define Absolute Segment
BYTE	Align Segment on Any Byte Address
BYTE	Data Type for 1 byte
COMMENT	Enter Multi-Line Comment
COMMON	Define Overlapping Segments
.CREF	Enable Cross-Reference Listings
DB	Define Byte
DD	Define Doubleword
DQ	Define Quadword

DT	Define Ten-byte Unit
DW	Define Word
DWORD	Data Type for 4 bytes
DUP	Duplicate Occurrences
ELSE	Assemble If Condition Not Met
END	Terminate Module
ENDIF	Terminate Conditional Block
ENDM	Terminate Macro or Repeat Block
ENDP	Terminate Procedure Definition
ENDS	End Segment or Structure Definition
EQ	Equal Relational Operator
EQU	Create Symbol
.ERR	Force Error
.ERR1	Force Error during Pass 1
.ERR2	Force Error during Pass 2
.ERRB	Error If String Is Blank
.ERRDEF	Error If Name Is Defined
.ERRDIF	Error If Strings Differ
.ERRE	Error If False
.ERRIDN	Error If Strings Are Identical
.ERRNB	Error If String Is Not Blank
.ERRNDEF	Error If Name Is Not Defined
.ERRNZ	Error If True
EVEN	Align on Word Boundary
EXITM	Immediate Macro Exit
EXTRN	Define External
FAR	Data Type for Label in Different Segment
GE	Greater Than or Equal Relational Operator
GROUP	Define Segment Group
GT	Greater Than Relational Operator
HIGH	Return High-Order 8 Bits
IF	Initiate Conditional Block
IF1	Assemble If Pass 1
IF2	Assemble If Pass 2
IFB	Assemble If Argument Is Blank
IFDEF	Assemble If Name Is Defined
IFDIF	Assemble If Arguments Differ
IFE	Assemble If False
IFIDN	Assemble If Arguments Are Identical
IFNB	Assemble If Argument Is Not Blank
IFNDEF	Assemble If Name Is Not Defined
INCLUDE	Process Code from External File
IRP	Assemble Once for Each Parameter
IRPC	Assemble Once for Each Character

LABEL	Create Variable or Label
.LALL	List All Macro Expansions
LE	Less Than or Equal Relational Operator
LENGTH	Return Length of Item
.LFCOND	List False Conditionals
.LIST	Restore Source-Code Listing
LOCAL	Create Symbol for Use in Macro
LOW	Return Low-Order 8 Bits
LT	Less Than Relational Operator
MACRO	Initiate Macro Definition
MASK	Return a Bit Mask
MEMORY	Locate segment as last segment possible
MOD	Modulus
NAME	Assign Name to Module
NE	Not Equal Relational Operator
NEAR	Data Type for Label in Same Segment
NOT	Bitwise NOT
OFFSET	Offset of Expression
OR	Bitwise Logical OR
ORG	Assign Location Counter
%OUT	Display Text during Assembly
PAGE	Align on 256-byte Boundary
PAGE	Page Control for Listings
PARA	Align on 16-byte Boundary
PROC	Initiate Procedure Definition
PTR	Change Type of Variable
PUBLIC	Concatenate All Like-Named Segments
PUBLIC	Make Symbol Available to All Modules
PURGE	Delete Macro Definition
QWORD	Data Type for 8 bytes
.RADIX	Set Input Radix
RECORD	Define Record Type
REPT	Initiate Repeat Block
.SALL	Suppress All Macro Expansion Listings
SEG	Return Segment Value
SEGMENT	Initiate Segment Definition
.SFCOND	Suppress Listing of False Conditionals
SHL	Shift Left
SHORT	Sets Label To SHORT Type
SHR	Shift Right
SIZE	Return Bytes Used by Variable
STACK	Define a Stack Segment
STRUC	Define Structure Type
SUBTTL	Specify Listing Subtitle

TBYTE	Data Type for 10 bytes
.TFCOND	Toggle False Conditional Listing
THIS	Create Operand at Current Location
TITLE	Specify Listing Title
TYPE	Return Size of Type
.TYPE	Return Mode and Scope of an Expression
WIDTH	Return Width in Bits
WORD	Align on 2-byte boundary
WORD	Data Type for 2 bytes
.XALL	List Macro Expansions That Produce Code
.XCREF	Suppress Cross-Reference Listings
.XLIST	Suppress Source-Code Listing
XOR	Bitwise Logical XOR

APPENDICE G: Tracce per la Risoluzione di Alcune Prove Scritte

In questa Appendice sono riportati alcuni segmenti di codice relativi a problemi assegnati come prove scritte per l'esame di Calcolatori Elettronici.

Tali risoluzioni sono da considerare solo a titolo di esempio, in quanto nella maggioranza dei casi l'enfasi è stata posta sulla modularità e quindi chiarezza del codice, piuttosto che sull'ottimizzazione degli algoritmi utilizzati.

CALCOLATORI ELETTRONICI

Prova scritta

6 novembre 1989

Scrivere un programma assembler che visualizzi sullo schermo:

- in alto a destra, il contenuto di una parola di memoria a 16 bit, in formato esadecimale,
- sulla riga successiva il numero dei bit con valore 1 presenti nella parola considerata. Tale numero deve essere espresso in forma decimale.

Traccia per la risoluzione

```
                TITLE  -  Prova Scritta del 6 novembre 1989

DSEG            SEGMENT PARA PUBLIC 'DATA'
LOCAZIONE        DW      ?
DSEG            ENDS


STACKM           SEGMENT PARA STACK 'STACK'  ;Viene allocata una zona di
                DB      64 DUP('12345678')  ; memoria per lo Stack: in
STACKM           ENDS                        ; tutto 64*8 bytes.


                ASSUME CS:CSEG,DS:DSEG,SS:STACKM
CSEG            SEGMENT PARA PUBLIC 'CODE'

;-----------------------------------------------------------------;
;                Corpo principale del programma                    ;
;-----------------------------------------------------------------;
```

```
MAIN            PROC  FAR
                PUSH  DS                ;Istruzioni da lasciare SEMPRE
                MOV   AX,00h            ; al principio dei programmi!
                PUSH  AX                ;
                CALL  INIZIALIZZAZIONE
                CALL  PRESENTAZIONE
                MOV   DH, ...
                MOV   DL, ...
                CALL  SPOSTA_CURSORE    ;Posizione il cursore secondo
                                        ; i valori presenti in DH e DL
                MOV   AX, LOCAZIONE     ;Carica il valore da stampare
                CALL  STAMPA_HEX        ;Stampa AX in formato esadec.
                MOV   DH, ...
                MOV   DL, ...
                CALL  SPOSTA_CURSORE    ;Riga successiva
                CALL  CALCOLA_BIT       ;Pone il numero di bit presenti
                                        ; in AX nel registro BX.
                MOV   AX, BX            ;Carica il valore da stampare
                CALL  STAMPA_DEC        ;Stampa AX in formato decimale
                RET                     ;Ritorno al Sistema Operativo
MAIN            ENDP

CALCOLA_BIT     PROC  NEAR
                MOV   CX, 16            ;Contatore
                XOR   BX, BX            ;Azzera il contatore di bit
CICLO:          SAL   AL, 1             ;Ruota attraverso il Carry
                RCL   AH, 1             ; i 16 bit di AX
                ADC   BX, 0             ;Se il Carry e' 1, incrementa BX
                LOOP  CICLO
                RET
CALCOLA_BIT     ENDP

INIZIALIZZAZIONE PROC NEAR
                ...
                RET
INIZIALIZZAZIONE ENDP

PRESENTAZIONE   PROC  NEAR
                ...
                RET
PRESENTAZIONE   ENDP
```

```
SPOSTA_CURSORE     PROC   NEAR
                   ...
                   RET
SPOSTA_CURSORE     ENDP

STAMPA_HEX         PROC   NEAR
                   ...
                   RET
STAMPA_HEX         ENDP

STAMPA_DEC         PROC   NEAR
                   ...
                   RET
STAMPA_DEC         ENDP


CSEG               ENDS
                   END   MAIN
```

CALCOLATORI ELETTRONICI

Prova scritta

24 settembre 1990

Scrivere un programma assembler che accetti da tastiera un carattere e visualizzi sullo schermo:

- il carattere se stampabile,
- il codice ASCII corrispondente.

Traccia per la risoluzione

```
                TITLE  -  Prova Scritta del 24 settembre 1990

DSEG            SEGMENT PARA PUBLIC 'DATA'
DSEG            ENDS


STACKM          SEGMENT PARA STACK 'STACK'  ;Viene allocata una zona di
                DB      64 DUP('12345678')  ; memoria per lo Stack: in
STACKM          ENDS                        ; tutto 64*8 bytes.


                ASSUME CS:CSEG,DS:DSEG,SS:STACKM
CSEG            SEGMENT PARA PUBLIC 'CODE'

;-----------------------------------------------------------------;
;                Corpo principale del programma                   ;
;-----------------------------------------------------------------;
```

```
MAIN            PROC  FAR
                PUSH  DS              ;Istruzioni da lasciare SEMPRE
                MOV   AX,00h          ; al principio dei programmi!
                PUSH  AX              ;
                CALL  INIZIALIZZAZIONE
                CALL  PRESENTAZIONE
                CALL  LEGGI_CHAR      ;Legge un carattere e lo pone in AL
                CALL  STAMPA_CHAR     ;Stampa il contenuto di AL
                CALL  STAMPA_SPAZIO   ;Stampa uno spazio
                CALL  STAMPA_DEC      ;Stampa AL in decimale
                RET                   ;Ritorno al Sistema Operativo
MAIN            ENDP


STAMPA_DEC      PROC  NEAR
                XOR   AH,AH           ;Azzera il registro AH
                MOV   BL,0Ah          ;Setta il dividendo (10 Decimale)
                IDIV  BL              ;Divisione tra interi
                MOV   DX,AX           ;Salva i risultati in DX per dopo
                CMP   AL,0Ah          ;Ci sono anche le centinaia?
                JB    DUE_CIFRE       ;Se non ci sono, salta avanti
                XOR   AH,AH           ;Azzera il registro AH
                IDIV  BL              ;Divisione tra interi
                MOV   DL,AH           ;Salva il risultato in DL per dopo
                ADD   AL, '0'         ;Addiziona la base dei numeri ASCII
                CALL  STAMPA_CHAR     ;Stampa le centinaia se necessario
                MOV   AL,DL           ;Decine
DUE_CIFRE:      ADD   AL, '0'         ;Addiziona la base dei numeri ASCII
                CALL  STAMPA_CHAR     ;Stampa le decine
                MOV   AL,DH           ;Unita'
                ADD   AL, '0'         ;Addiziona la base dei numeri ASCII
                CALL  STAMPA_CHAR     ;Stampa le unita'
                RET                   ;Ritorno alla procedura chiamante
STAMPA_DEC      ENDP


STAMPA_CHAR     PROC  NEAR
                PUSH  AX              ;Salva AX
                MOV   AH,0Eh          ;Servizio BIOS 'Write Char'
                MOV   BL,0            ;Pagina attiva
                INT   10h
                POP   AX              ;Ripristina AX
                RET                   ;Ritorno alla procedura chiamante
```

```
STAMPA_CHAR     ENDP


LEGGI_CHAR      PROC  NEAR
                MOV   AH,07h          ;Servizio DOS 'Read Keyboard Char
                INT   21h             ; Without Echo'
                RET                   ;Ritorno alla procedura chiamante
LEGGI_CHAR      ENDP


STAMPA_SPAZIO   PROC  NEAR
                PUSH  AX              ;Salva AX
                MOV   AL, ' '         ;Carica in AL il codice di ' '
                CALL  STAMPA_CHAR     ;Stampa AL
                POP   AX              ;Ripristina AX
                RET
STAMPA_SPAZIO   ENDP


INIZIALIZZAZIONE  PROC  NEAR
                ...
                RET
INIZIALIZZAZIONE  ENDP


PRESENTAZIONE   PROC  NEAR
                ...
                RET
PRESENTAZIONE   ENDP



CSEG            ENDS
                END   MAIN
```

CALCOLATORI ELETTRONICI

Prova scritta

22 ottobre 1990

Scrivere un programma in assembler che accetti da tastiera un numero positivo di 4 cifre in codice binario e che visualizzi sullo schermo:

- Il numero stesso in codice binario.
- Il suo quadrato sempre in codice binario.

Traccia per la risoluzione

```
                TITLE  -  Prova Scritta del 22 ottobre 1990

DSEG            SEGMENT PARA PUBLIC 'DATA'
DSEG            ENDS


STACKM          SEGMENT PARA STACK 'STACK'   ;Viene allocata una zona di
                DB      64 DUP('12345678')   ; memoria per lo Stack: in
STACKM          ENDS                         ; tutto 64*8 bytes.


                ASSUME CS:CSEG,DS:DSEG,SS:STACKM
CSEG            SEGMENT PARA PUBLIC 'CODE'

;-----------------------------------------------------------------;
;                   Corpo principale del programma                ;
;-----------------------------------------------------------------;
```

```
MAIN                PROC   FAR
                   PUSH   DS                ;Istruzioni da lasciare SEMPRE
                   MOV    AX,00h            ; al principio dei programmi!
                   PUSH   AX                ;
                   CALL   INIZIALIZZAZIONE
                   CALL   PRESENTAZIONE
                   CALL   LEGGI_NUMERO        ;Il numero letto va in AL
                   CALL   STAMPA_BIN          ;Stampa il contenuto di AL
                   MUL    AL                ;Calcola il quadrato
                   CALL   STAMPA_BIN          ;Stampa il contenuto di AL
                   RET                      ;Ritorno al Sistema Operativo
MAIN                ENDP


STAMPA_BIN            PROC   NEAR
                   MOV    CX,08h            ;Contatore dei bits
CICLO_BINARIO:       XOR    AH,AH             ;Azzera il registro AH
                   SHL    AL,01h            ;Shift Logico a Sinistra
                   ADC    AH,AH             ;In AH si pone il CARRY
                   CALL   STAMPA_CIFRA        ;Stampa la cifra binaria in AH
                   LOOP   CICLO_BINARIO      ;Itera fino a che CX=0
                   RET                      ;Ritorno alla procedura chiamante
STAMPA_BIN            ENDP


INIZIALIZZAZIONE   PROC   NEAR
                   ...
                   RET
INIZIALIZZAZIONE   ENDP


PRESENTAZIONE         PROC   NEAR
                   ...
                   RET
PRESENTAZIONE         ENDP


LEGGI_NUMERO          PROC   NEAR
                   ...
                   RET
LEGGI_NUMERO          ENDP
```

```
STAMPA_CIFRA    PROC   NEAR
                ...
                RET
STAMPA_CIFRA    ENDP


CSEG            ENDS
                END    MAIN
```

CALCOLATORI ELETTRONICI

Prova scritta

3 giugno 1991

Scrivere un programma in assembler che:

- Accetti da tastiera una sequenza di numeri (max 60) e la visualizzi sullo schermo.
- Accetti una sequenza di tre numeri e la visualizzi sullo schermo.
- Verifichi quante volte la sequenza dei tre numeri compare nella prima sequenza e stampi il risultato.

Traccia per la risoluzione

```
                    TITLE  -  Prova Scritta del 3 giugno 1991


DSEG                SEGMENT PARA PUBLIC 'DATA'
ENTER               EQU     13
MAX_LUNGH           EQU     60
SEQUENZA_1          DB      MAX_LUNGH DUP(?)    ;Sequenza di MAX_LUNGH numeri
SEQUENZA_2          DB      3  DUP(?)           ;Sequenza di  3 numeri
LUNGH_PRIMA_STR     DW      ?                   ;Lunghezza prima stringa
DSEG                ENDS



STACKM              SEGMENT PARA STACK 'STACK'  ;Viene allocata una zona di
                    DB      64 DUP('12345678')  ; memoria per lo Stack: in
STACKM              ENDS                        ; tutto 64*8 bytes.


                    ASSUME CS:CSEG,DS:DSEG,SS:STACKM
CSEG                SEGMENT PARA PUBLIC 'CODE'
```

```
;-----------------------------------------------------------------;
;                    Corpo principale del programma                    ;
;-----------------------------------------------------------------;

MAIN            PROC   FAR
                PUSH  DS               ;Istruzioni da lasciare SEMPRE
                MOV   AX,00h           ; al principio dei programmi!
                PUSH  AX               ;
                CALL  INIZIALIZZAZIONE
                CALL  PRESENTAZIONE
                CALL  LETTURA˙1        ;Legge la prima stringa
                CALL  LETTURA˙2        ;Legge la seconda stringa
                CALL  CONFRONTO        ;Confronta le due stringhe e pone
                                       ; il risultato in AX
                CALL  STAMPA˙NUMERO    ;Stampa il contenuto di AX
                RET                    ;Ritorno al Sistema Operativo
MAIN            ENDP


LETTURA˙1       PROC   NEAR
                MOV   BX, 0            ;Azzera la lunghezza
CICLO:          CALL  LEGGE˙CHAR       ;Legge un carattere da tastiera
                                       ; e lo pona in AL.
                CMP   AL, ENTER        ;Lo confronta con ENTER
                JZ    FINE˙LETTURA     ; se e' ENTER termina la lettura
                MOV   [SEQUENZA˙1+BX], AL ;Memorizza il carattere
                INC   BX               ;Incrementa la lunghezza
                CMP   BX, MAX˙LUNGH    ;Confronta: se non si e' raggiunta
                JNZ   CICLO            ; la max lunghezza, reitera
FINE˙LETTURA:   MOV   LUNGH˙PRIMA˙STR, BX
                RET
LETTURA˙1       ENDP


LETTURA˙2       PROC   NEAR
                CALL  LEGGE˙CHAR       ;Legge un carattere e lo pone in AL
                MOV   [SEQUENZA˙2+0], AL ;Memorizza il carattere
                CALL  LEGGE˙CHAR       ;Legge un carattere e lo pone in AL
                MOV   [SEQUENZA˙2+1], AL ;Memorizza il carattere
                CALL  LEGGE˙CHAR       ;Legge un carattere e lo pone in AL
                MOV   [SEQUENZA˙2+2], AL ;Memorizza il carattere
                RET
LETTURA˙2       ENDP
```

```
CONFRONTO           PROC   NEAR
                MOV    AX, 0              ;Numero di ricorrenze
                MOV    CX, LUNGH_PRIMA_STR ;Numero di caratteri da confr.
                SUB    CX, 2              ;Numero di confronti da eseguire
                MOV    BX, OFFSET SEQUENZA_1
CICLO_CONFRONTO:  MOV    DX, WORD PTR [SEQUENZA_2]
                CMP    DX, [BX]           ;Confronta i primi 2 bytes
                JNZ    NON_UGUALI         ;Salta se non sono uguali
                MOV    DH, [SEQUENZA_2+2]
                CMP    DH, [BX+2]         ;Confronta il terzo byte
                JNZ    NON_UGUALI         ;Salta se non sono uguali
                INC    AX                 ;Incrementa il numero di ricorrenze
NON_UGUALI:       INC    BX               ;Avanza la scansione di SEQUENZA_1
                LOOP   CICLO_CONFRONTO ;Continua fino a che CX=0
                RET
CONFRONTO           ENDP


INIZIALIZZAZIONE  PROC   NEAR
                ...
                RET
INIZIALIZZAZIONE  ENDP


PRESENTAZIONE       PROC   NEAR
                ...
                RET
PRESENTAZIONE       ENDP


STAMPA_NUMERO       PROC   NEAR
                ...
                RET
STAMPA_NUMERO       ENDP


LEGGE_CHAR          PROC   NEAR
                ...
                RET
LEGGE_CHAR          ENDP

CSEG                ENDS
                END    MAIN
```

CALCOLATORI ELETTRONICI

Prova scritta

17 luglio 1991

Scrivere un programma in assembler che:

- Accetti da tastiera una parola di 16 bit nella sua rappresentazione esadecimale.
- Visualizzi sullo schermo la rappresentazione esadecimale delle 16 configurazioni che si ottengono facendo circolare il dato precedente in un registro di 16 posizioni richiuso su se stesso.

Traccia per la risoluzione

```
            TITLE  -  Prova Scritta del 17 luglio 1991

DSEG        SEGMENT PARA PUBLIC 'DATA'
DSEG        ENDS

STACKM      SEGMENT PARA STACK 'STACK'  ;Viene allocata una zona di
            DB      64 DUP('12345678')  ; memoria per lo Stack: in
STACKM      ENDS                        ; tutto 64*8 bytes.

            ASSUME CS:CSEG,DS:DSEG,SS:STACKM
CSEG        SEGMENT PARA PUBLIC 'CODE'
```

```
;-------------------------------------------------------------------;
;                    Corpo principale del programma                 ;
;-------------------------------------------------------------------;

MAIN                PROC  FAR
                  PUSH  DS                ;Istruzioni da lasciare SEMPRE
                  MOV   AX,00h            ; al principio dei programmi!
                  PUSH  AX                ;
                  CALL  INIZIALIZZAZIONE
                  CALL  PRESENTAZIONE
                  CALL  LEGGI˙PAROLA      ;Legge la parola e la pone in AX
                  MOV   CX, 16            ;Contatore
CICLO˙PRINCIPALE: PUSH  CX                ;Salva CX
                  CALL  STAMPA˙HEX        ;Stampa il contenuto di AX
                  SAL   AL,1              ;Shift a sx di AL
                  RCL   AH,1              ;Rotate a sx di AH (il Carry
                                          ; rientra nel bit meno signif.
                  ADC   AL, 0             ;Il Carry di AH rientra in AL
                  POP   CX                ;Ripristina CX
                  LOOP  CICLO˙PRINCIPALE
                  RET                     ;Ritorno al Sistema Operativo
MAIN                ENDP

LEGGI˙PAROLA          PROC  NEAR
                  CALL  LEGGI˙TASTO       ;Legge da tastiera e pone in AL
                  PUSH  AX                ;Salva AX
                  CALL  LEGGI˙TASTO       ;Legge da tastiera e pone in AL
                  POP   BX                ;Pone in BL il tasto precedente
                  MOV   CL, 4
                  SHL   BL, CL            ;Sposta nei 4 bit piu' signif.
                                          ; il codice del primo tasto
                  OR    BL, AL            ;Pone in BL i due codici
                  PUSH  BX                ;Salve il primo byte (BL)
                  CALL  LEGGI˙TASTO       ;Legge da tastiera e pone in AL
                  PUSH  AX                ;Salva AX
                  CALL  LEGGI˙TASTO       ;Legge da tastiera e pone in AL
                  POP   BX                ;Pone in BL il tasto precedente
                  MOV   CL, 4
                  SHL   BL, CL            ;Sposta nei 4 bit piu' signif.
                                          ; il codice del primo tasto
                  OR    AL, BL            ;Pone in AL i due codici
                  POP   BX                ;Ripristina BX
                  MOV   AH, BL            ;Pone in AX i 4 codici
```

```
                RET
LEGGI_PAROLA        ENDP


STAMPA_HEX          PROC   NEAR
                PUSH   AX                 ;Salva AX
                MOV     AL, AH            ;Prepara per stampare AH
                CALL   STAMPA_HEX_AL      ;Stampa AL in esadecimale
                POP    AX                 ;Ripristina AX
                PUSH   AX                 ;Risalva AX
                CALL   STAMPA_HEX_AL      ;Stampa AL
                CALL   STAMPA_ENTER       ;Sposta il cursore alla riga succ.
                POP    AX                 ;Ripristina AX
                RET
STAMPA_HEX          ENDP


STAMPA_HEX_AL       PROC   NEAR
                PUSH   AX                 ;Salva AX
                MOV     CL,4              ;Porta i bit 7654 di AL in 3210
                SHR    AL,CL              ; ponendo 0 nei bit 7654
                CALL   STAMPA_NUMERO      ;Stampa il numero in AL
                POP    AX                 ;Ripristina AX
                AND    AL,00001111b       ;Mantiene solo i bit 3210 di AL
                CALL   STAMPA_NUMERO      ;Stampa il numero in AL
                RET                       ;Ritorno alla procedura chiamante
STAMPA_HEX_AL       ENDP


STAMPA_NUMERO         PROC   NEAR
                PUSH   AX                 ;Salva AX
                ADD    AL, '0'            ;Addiziona la base dei numeri ASCII
                CMP    AL, '9'+1          ;Se e' minore di 9,
                JC     CIFRA_DEC          ; salta a CIFRA_DEC
                ADD    AL, 'A'-'9'-1      ;Altrimenti aggiusta per le lettere
CIFRA_DEC:        MOV    AH,0Eh           ;Servizio BIOS 'Write Char'
                MOV    BL,0               ;Pagina attiva
                INT    10h
                POP    AX                 ;Ripristina AX
                RET                       ;Ritorno alla procedura chiamante
STAMPA_NUMERO         ENDP
```

```
LEGGI˙TASTO         PROC   NEAR
CICLO˙LETTURA:      MOV    AH,07h            ;Servizio DOS 'Read Keyboard Char
                   INT    21h               ; Without Echo'
                   CMP    AL,'0'            ;Se e' minore di '0'
                   JC     CICLO˙LETTURA     ; allora rilegge il carattere
                   CMP    AL,'9'+1          ;Se e' compreso tra '0' e '9'
                   JC     FINE˙LETTURA      ; allora termina la lettura
                   CMP    AL,'A'            ;Se e' ¿'9' e ¡'A'
                   JC     CICLO˙LETTURA     ; allora rilegge il carattere
                   CMP    AL,'F'+1          ;Se e' maggiore di 'F'
                   JNC    CICLO˙LETTURA     ; allora rilegge il carattere
FINE˙LETTURA:       SUB    AL, '0'           ;Toglie '0'
                   CMP    AL, 10            ;Se e' minore o uguale a 9
                   JC     CIFRA˙DECIMALE    ; termina la lettura
                   SUB    AL, 'A'-'9'-1     ;Altrimenti lo porta nel range
CIFRA˙DECIMALE:     RET
LEGGI˙TASTO         ENDP


STAMPA˙ENTER        PROC   NEAR
                   ...
                   RET
STAMPA˙ENTER        ENDP


INIZIALIZZAZIONE   PROC   NEAR
                   ...
                   RET
INIZIALIZZAZIONE   ENDP


PRESENTAZIONE      PROC   NEAR
                   ...
                   RET
PRESENTAZIONE      ENDP


CSEG                ENDS
                   END    MAIN
```

CALCOLATORI ELETTRONICI

Prova scritta

29 luglio 1991

Scrivere un programma in assembler che:

- Accetti da tastiera numeri positivi e negativi di 4 cifre piu‘ segno.
- Stampi su schermo la lista di numeri inseriti in ordine crescente.

Il programma deve operare nel seguente modo:

1. Presentarsi.
2. Richiedere se si vuole:
    - inserire un numero,
    - stampare la lista ordinata dei numeri già inseriti,
    - uscire dal programma.
3. Ritornare, se del caso, al punto 2.

Traccia per la risoluzione

```
                         TITLE   -   Prova Scritta del 29 luglio 1991

DSEG                SEGMENT PARA PUBLIC 'DATA'
MAX˙NUM˙VALORI       EQU       100
LUNGH˙VALORE         EQU       7
ENTER               EQU       13
LINEFEED            EQU       10
MESS˙INSERIMENTO     DB        ENTER, LINEFEED, 'Inserire il numero: $'
```

```
ISTRUZIONI        DB    ENTER, LINEFEED, LINEFEED
                  DB    'Premere: 1 per inserire un nuovo valore,'
                  DB    ENTER, LINEFEED
                  DB    '        2 per stampare la lista dei numeri,'
                  DB    ENTER, LINEFEED
                  DB    '        3 per uscire dal programma.'
LINEA_VUOTA       DB    ENTER, LINEFEED, '$'
VALORI            DB    MAX_NUM_VALORI DUP('?????', ENTER, LINEFEED)
NUM_VALORI        DB    ?
DSEG              ENDS


STACKM            SEGMENT PARA STACK 'STACK'  ;Viene allocata una zona di
                  DB    64 DUP('12345678')  ; memoria per lo Stack: in
STACKM            ENDS                      ; tutto 64*8 bytes.


                  ASSUME CS:CSEG,DS:DSEG,SS:STACKM
CSEG              SEGMENT PARA PUBLIC 'CODE'

;-------------------------------------------------------------------;
;                Corpo principale del programma                      ;
;-------------------------------------------------------------------;

MAIN              PROC  FAR
                  PUSH  DS              ;Istruzioni da lasciare SEMPRE
                  MOV   AX,00h          ; al principio dei programmi!
                  PUSH  AX              ;
                  CALL  INIZIALIZZAZIONE
                  CALL  PRESENTAZIONE
                  MOV   DX, OFFSET ISTRUZIONI ;Stampa le istruzioni
                  CALL  STAMPA_STRINGA
CICLO_PRINCIPALE: CALL  LEGGI_SCELTA    ;Legge la scelta da tastiera
                                        ; e pone il risultato in AL
                  CMP   AL, '1'         ;Se il tasto non e' '1',
                  JNZ   NO_INSERIMENTO  ; allora salta
                  CALL  INSERIMENTO     ;Procedura di inserimento
                  CALL  ORDINAMENTO     ;Procedura di ordinamento
                  MOV   DX, OFFSET ISTRUZIONI ;Stampa le istruzioni
                  CALL  STAMPA_STRINGA
                  JMP   CICLO_PRINCIPALE
NO_INSERIMENTO:   CMP   AL, '2'         ;Se il tasto non e' '2',
```

```
                  JNZ    USCITA          ; allora esce
                  CALL   STAMPA_LISTA
                  MOV    DX, OFFSET ISTRUZIONI ;Stampa le istruzioni
                  CALL   STAMPA_STRINGA
                  JMP    CICLO_PRINCIPALE ;Ritorno al ciclo principale
USCITA:           RET                    ;Ritorno al Sistema Operativo
MAIN              ENDP


LEGGI_SCELTA      PROC  NEAR
NUOVA_LETTURA:    MOV    AH,08h          ;Servizio DOS 'Read Keyboard Char
                  INT    21h             ; Without Echo'
                  CMP    AL,'1'          ;Controlla il range: se e'
                  JC     NUOVA_LETTURA   ; compreso tra '1' e '3', allora
                  CMP    AL,'3'+1        ; esce, altrimenti ritorna
                  JNC    NUOVA_LETTURA   ; a NUOVA_LETTURA
                  RET
LEGGI_SCELTA      ENDP


INSERIMENTO       PROC  NEAR
                  MOV    BL, [NUM_VALORI]
                  CMP    BL, MAX_NUM_VALORI ;Se il numero di valori e'
                  JZ     FINE_INSERIMENTO ; troppo elevato, esce subito
                  MOV    DX, OFFSET MESS_INSERIMENTO
                  CALL   STAMPA_STRINGA
                  MOV    AL, LUNGH_VALORE
                  MUL    BL
                  ADD    AX, OFFSET VALORI
                  MOV    BX, AX          ;Punta al primo valore vuoto
                  CALL   LEGGI_SEGNO     ;Legge il segno e lo pone in AL
                  MOV    [BX], AL        ;Lo memorizza
                  MOV    CX, 4           ;Numero di cifre
CICLO_INSERIM:    INC    BX              ;Aggiorna il puntatore
                  CALL   LEGGI_CIFRA     ;Legge una cifra numerica
                  MOV    [BX], AL        ;Memorizza la cifra
                  LOOP   CICLO_INSERIM   ;Continua fino a che CX=0
                  INC    BYTE PTR [NUM_VALORI] ;Aggiorna il numero di valori
FINE_INSERIMENTO: RET
INSERIMENTO       ENDP


LEGGI_SEGNO       PROC  NEAR
NUOVO_SEGNO:      MOV    AH,08h          ;Servizio DOS 'Read Keyboard Char
```

```
                INT    21h               ; Without Echo'
                CMP    AL,'+'            ;Controlla se il carattere e' '+'
                JZ     FINE˙SEGNO        ; se lo e' termina.
                CMP    AL,'-'            ;Controlla se il carattere e' '-'
                JNZ    NUOVO˙SEGNO       ; se non lo e', ripete
FINE˙SEGNO:     PUSH   AX                ;Salva il registro AX
                MOV    AH,02h            ;Servizio DOS 'Character Output'
                MOV    DL, AL            ;Carattere da stampare
                INT    21h
                POP    AX                ;Ripristina il contenuto di AX
                RET
LEGGI˙SEGNO     ENDP


LEGGI˙CIFRA     PROC   NEAR
NUOVA˙CIFRA:    MOV    AH,08h            ;Servizio DOS 'Read Keyboard Char
                INT    21h               ; Without Echo'
                CMP    AL,'0'            ;Controlla il range: se e'
                JC     NUOVA˙CIFRA       ; compreso tra '0' e '9', allora
                CMP    AL,'9'+1          ; esce, altrimenti ritorna
                JNC    NUOVA˙CIFRA       ; a NUOVA˙CIFRA
                PUSH   AX                ;Salva il registro AX
                MOV    AH,02h            ;Servizio DOS 'Character Output'
                MOV    DL, AL            ;Carattere da stampare
                INT    21h
                POP    AX                ;Ripristina il contenuto di AX
                RET
LEGGI˙CIFRA     ENDP


ORDINAMENTO     PROC   NEAR
                ...
                RET
ORDINAMENTO     ENDP


STAMPA˙LISTA    PROC   NEAR
                MOV    DX, OFFSET LINEA˙VUOTA
                CALL   STAMPA˙STRINGA    ;Stampa una linea bianca
                MOV    DX, OFFSET VALORI
                MOV    AL, [NUM˙VALORI]
                MOV    BL, LUNGH˙VALORE
                MUL    BL
                MOV    BX, DX
```

```
                ADD     BX, AX          ;Punta al termine della stringa
                MOV     CL, [BX]        ;Salva il contenuto di [BX]
                MOV     BYTE PTR [BX], '$' ;Segnala il termine per la
                CALL    STAMPA_STRINGA  ; procedura di stampa
                MOV     [BX], CL        ;Ripristina il contenuto di [BX]
                RET
STAMPA_LISTA        ENDP


STAMPA_STRINGA      PROC    NEAR
                MOV     AH,09h          ;Servizio DOS 'Print String'
                INT     21h
                RET
STAMPA_STRINGA      ENDP


INIZIALIZZAZIONE  PROC  NEAR
                MOV     AX, SEG DSEG
                MOV     DS, AX
                MOV     [NUM_VALORI], 0 ;Numero di valori inseriti
                RET
INIZIALIZZAZIONE  ENDP


PRESENTAZIONE       PROC    NEAR
                ...
                RET
PRESENTAZIONE       ENDP



CSEG              ENDS
                END     MAIN
```

CALCOLATORI ELETTRONICI

Prova scritta

16 settembre 1991

Scrivere un programma in assembler che:

- Accetti da tastiera parole (da 1 a 10 caratteri).
- Stampi su schermo la lista delle parole inserite in ordine di lunghezza.

Il programma deve operare nel seguente modo:

1. Presentarsi.
2. Richiedere se si vuole:
   - inserire una parola.
   - stampare la lista delle parole gia' inserite.
   - uscire dal programma.

   Al termine dell'inserimento e della stampa il programma deve ripetere le operazioni previste al punto 2.

Traccia per la risoluzione

```
                 TITLE  -  Prova Scritta del 16 settembre 1991


DSEG                 SEGMENT PARA PUBLIC 'DATA'
MAX_NUM_PAROLE       EQU     100          ;Deve essere minore di 256
MAX_LUNGH_PAROLA     EQU     12
LUNGH_PAROLA         EQU     10
ENTER                EQU     13
LINEFEED             EQU     10
```

```
MESS_INSERIMENTO  DB    ENTER, LINEFEED, 'Inserire la parola: $'
ISTRUZIONI        DB    ENTER, LINEFEED, LINEFEED
                  DB    'Premere: 1 per inserire una parola,'
                  DB    ENTER, LINEFEED
                  DB    '         2 per stampare la lista delle parole,'
                  DB    ENTER, LINEFEED
                  DB    '         3 per uscire dal programma.'
LINEA_VUOTA       DB    ENTER, LINEFEED, '$'
PAROLE            DB    MAX_NUM_PAROLE DUP( LUNGH_PAROLA DUP(' '), "
                      ENTER, LINEFEED)
LUNGHEZZE         DB    MAX_NUM_PAROLE DUP('?')
NUM_PAROLE        DB    ?
DSEG              ENDS


STACKM            SEGMENT PARA STACK 'STACK'  ;Viene allocata una zona di
                  DB    64 DUP('12345678')    ; memoria per lo Stack: in
STACKM            ENDS                        ; tutto 64*8 bytes.


                  ASSUME CS:CSEG,DS:DSEG,SS:STACKM
CSEG              SEGMENT PARA PUBLIC 'CODE'

;------------------------------------------------------------------;
;                 Corpo principale del programma                    ;
;------------------------------------------------------------------;

MAIN              PROC  FAR
                  PUSH  DS               ;Istruzioni da lasciare SEMPRE
                  MOV   AX,00h           ; al principio dei programmi!
                  PUSH  AX               ;
                  CALL  INIZIALIZZAZIONE
                  CALL  PRESENTAZIONE
CICLO_PRINCIPALE: CALL  LEGGI_SCELTA     ;Legge la scelta da tastiera
                                         ; e pone il risultato in AL
                  CMP   AL, '1'          ;Se il tasto non e' '1',
                  JNZ   NO_INSERIMENTO   ; allora salta
                  CALL  INSERIMENTO      ;Procedura di inserimento
                  CALL  ORDINAMENTO      ;Procedura di ordinamento
                  MOV   DX, OFFSET ISTRUZIONI ;Stampa le istruzioni
                  CALL  STAMPA_STRINGA
                  JMP   CICLO_PRINCIPALE
```

```
NO_INSERIMENTO:   CMP   AL, '2'           ;Se il tasto non e' '2',
                  JNZ   USCITA            ; allora esce
                  CALL  STAMPA_LISTA
                  MOV   DX, OFFSET ISTRUZIONI ;Stampa le istruzioni
                  CALL  STAMPA_STRINGA
                  JMP   CICLO_PRINCIPALE ;Ritorno al ciclo principale
USCITA:           RET                     ;Ritorno al Sistema Operativo
MAIN              ENDP


LEGGI_SCELTA      PROC  NEAR
NUOVA_LETTURA:    MOV   AH,08h            ;Servizio DOS 'Read Keyboard Char
                  INT   21h               ; Without Echo'
                  CMP   AL,'1'            ;Controlla il range: se e'
                  JC    NUOVA_LETTURA     ; compreso tra '1' e '3', allora
                  CMP   AL,'3'+1          ; esce, altrimenti ritorna
                  JNC   NUOVA_LETTURA     ; a NUOVA_LETTURA
                  RET
LEGGI_SCELTA      ENDP


INSERIMENTO       PROC  NEAR
                  MOV   BL, [NUM_PAROLE]
                  CMP   BL, MAX_NUM_PAROLE ;Se il numero di parole e'
                  JZ    FINE_INSERIMENTO ; troppo elevato, esce subito
                  MOV   DX, OFFSET MESS_INSERIMENTO
                  CALL  STAMPA_STRINGA
                  MOV   AL, MAX_LUNGH_PAROLA
                  MUL   BL
                  ADD   AX, OFFSET PAROLE
                  MOV   BX, AX            ;Punta alla prima parola vuota
                  CALL  LEGGI_PAROLA      ;Legge una parola
                  MOV   BL,[NUM_PAROLE] ;Pone in BL il numero di parole
                  XOR   BH, BH            ;Azzera AH e memorizza la lunghezza
                  MOV   [OFFSET LUNGHEZZE + BX], DL ; della parola inserita
                  INC   BYTE PTR [NUM_PAROLE] ;Aggiorna il numero di parole
FINE_INSERIMENTO: RET
INSERIMENTO       ENDP


LEGGI_PAROLA      PROC  NEAR
                  MOV   CX,LUNGH_PAROLA ;Inizializza il contatore caratteri
NUOVA_LETT:       MOV   AH,08h            ;Servizio DOS 'Read Keyboard Char
                  INT   21h               ; Without Echo'
```

```
                CMP   AL, ENTER        ;Se il tasto e' enter,
                JZ    FINE_LETTURA     ; termina la lettura
                CMP   AL,' '           ;Controlla il range: se non e'
                JC    NUOVA_LETT       ; valido, rilegge un altro tasto
                PUSH  AX               ;Salva il registro AX
                MOV   AH,02h           ;Servizio DOS 'Character Output'
                MOV   DL, AL           ;Carattere da stampare
                INT   21h
                POP   AX               ;Ripristina il contenuto di AX
                MOV   [BX], AL         ;Memorizza il tasto
                INC   BX               ;Aggiorna il puntatore
                LOOP  NUOVA_LETT       ;Reitera fino a che CX=0
FINE_LETTURA:   MOV   DX, LUNGH_PAROLA ;Calcola in DX la lunghezza della
                SUB   DX, CX           ; parola inserita
                RET
LEGGI_PAROLA    ENDP


ORDINAMENTO     PROC  NEAR
                MOV   AL, [NUM_PAROLE]
                DEC   AL               ;Decrementa AL
                JZ    FINE_ORDINAMENTO ;Se AL era 1, termina la procedura
                MOV   CL, MAX_LUNGH_PAROLA
                MUL   CL
                MOV   SI, OFFSET PAROLE ;SI punta all'inizio area parole
                MOV   DI, SI           ;Copia del puntatore
                ADD   DI, AX           ;DI punta all'inizio ultima parola
                XOR   BX, BX           ;Contatore parole
CICLO_ORDINAM:  CALL  CONFRONTO        ;Confronta le parole puntate da SI
                                       ; e DI. Se vanno scambiate ritorna
                                       ; settando il flag Carry
                JNC   NO_SCAMBIO       ;Se no, salata lo scambio
                CALL  SCAMBIO          ;Procedura di scambio parole
NO_SCAMBIO:     INC   BX               ;Incrementa contatore parole
                ADD   SI, MAX_LUNGH_PAROLA
                CMP   SI, DI           ;Se il confronto non e' terminato
                JNZ   CICLO_ORDINAM    ; ripeti la procedura
FINE_ORDINAMENTO: RET
ORDINAMENTO     ENDP


SCAMBIO         PROC  NEAR
                PUSH  SI
                PUSH  DI
```

```
CICLO˙SCAMBIO:    MOV   AL, BYTE PTR [SI] ;Legge il prossimo carattere
                  CMP   AL, ENTER         ;Se al termine della parola
                  JZ    SCAMBIO˙LUNGH     ; termina lo scambio
                  MOV   AH, BYTE PTR [DI]
                  MOV   BYTE PTR [DI], AL ;Procedura di scambio parole
                  MOV   BYTE PTR [SI], AH
                  INC   DI                ;Incrementa il puntatore DI
                  INC   SI                ;Incrementa il puntatore SI
                  JMP   CICLO˙SCAMBIO     ;Ripeti per un altro carattere
SCAMBIO˙LUNGH:    MOV   AL, [OFFSET LUNGHEZZE + BX] ;Prima lunghezzza
                  PUSH  BX                ;Ora scambia le lunghezze
                  MOV   BL, [NUM˙PAROLE]
                  XOR   BH, BH
                  MOV   AH, [OFFSET LUNGHEZZE-1+BX]
                  MOV   [OFFSET LUNGHEZZE-1+BX], AL
                  POP   BX
                  MOV   [OFFSET LUNGHEZZE + BX], AH
                  POP   DI
                  POP   SI
                  RET
SCAMBIO           ENDP


CONFRONTO         PROC  NEAR
                  MOV   AL, [OFFSET LUNGHEZZE + BX] ;Prima lunghezzza
                  PUSH  BX
                  MOV   BL, [NUM˙PAROLE]
                  XOR   BH, BH
                  CMP   AL, [OFFSET LUNGHEZZE-1+BX] ;Seconda lunghezza
                  POP   BX
                  RET
CONFRONTO         ENDP


STAMPA˙LISTA      PROC  NEAR
                  MOV   DX, OFFSET LINEA˙VUOTA
                  CALL  STAMPA˙STRINGA    ;Stampa una linea bianca
                  MOV   DX, OFFSET PAROLE
                  MOV   AL, [NUM˙PAROLE]
                  MOV   BL, MAX˙LUNGH˙PAROLA
                  MUL   BL
                  MOV   BX, DX
                  ADD   BX, AX            ;Punta al termine della stringa
                  MOV   CL, [BX]          ;Salva il contenuto di [BX]
```

```
                MOV    BYTE PTR [BX], '$' ;Segnala il termine per la
                CALL   STAMPA_STRINGA   ; procedura di stampa
                MOV    [BX], CL          ;Ripristina il contenuto di [BX]
                RET
STAMPA_LISTA       ENDP


STAMPA_STRINGA     PROC   NEAR
                MOV    AH,09h            ;Servizio DOS 'Print String'
                INT    21h
                RET
STAMPA_STRINGA     ENDP


INIZIALIZZAZIONE  PROC   NEAR
                MOV    AX, SEG DSEG
                MOV    DS, AX
                MOV    [NUM_PAROLE], 0 ;Numero di parole inserite
                RET
INIZIALIZZAZIONE  ENDP


PRESENTAZIONE      PROC   NEAR
                MOV    DX, OFFSET ISTRUZIONI ;Stampa le istruzioni
                CALL   STAMPA_STRINGA
                RET
PRESENTAZIONE      ENDP



CSEG               ENDS
                END    MAIN
```

CALCOLATORI ELETTRONICI

Prova scritta

21 ottobre 1991

Scrivere un programma in assembler che agisca da semplice guida telefonica:

- Accetti da tastiera il cognome di un utente telefonico (max 15 campi alfanumerici)
- Il numero di telefono (max 12 caratteri numerici).

Il programma deve operare nel seguente modo:

1. Presentarsi.
2. Richiedere se si vuole:
   - inserire un nuovo utente.
   - richiedere il numero di telefono di un utente.
   - uscire dal programma.

   Al termine dell'inserimento e della richiesta il programma deve ripetere le operazioni previste al punto 2.

Traccia per la risoluzione

```
                    TITLE  -  Prova Scritta del 21 ottobre 1991


DSEG                    SEGMENT PARA PUBLIC 'DATA'
MAX_NUM_ENTRATE     EQU     100           ;Deve essere minore di 256
LUNGH_UTENTE        EQU     15
LUNGH_TELEFONO      EQU     12
MAX_LUNGH_ENTRATA EQU     LUNGH_UTENTE + LUNGH_TELEFONO + 1
LUNGH_ENTRATA       EQU     LUNGH_UTENTE + LUNGH_TELEFONO
```

```
ENTER              EQU      13
LINEFEED           EQU      10
INSERIM˙UTENTE       DB       ENTER, LINEFEED, 'Inserire nome utente: $'
INSERIM˙TELEFONO     DB       ENTER, LINEFEED, 'Inserire numero di telefono: $'
MESS˙RICERCA         DB       ENTER, LINEFEED, 'Inserire utente da cercare: $'
NON˙TROVATO          DB       ENTER, LINEFEED, 'Nome non trovato.$'
TROVATO              DB       ENTER, LINEFEED, 'Numero di telefono: $'
ISTRUZIONI           DB       ENTER, LINEFEED, LINEFEED
                   DB       'Premere: 1 per inserire un utente,'
                   DB       ENTER, LINEFEED
                   DB       '         2 per cercare un utente,'
                   DB       ENTER, LINEFEED
                   DB       '         3 per uscire dal programma.'
LINEA˙VUOTA          DB       ENTER, LINEFEED, '$'
ENTRATE              DB       MAX˙NUM˙ENTRATE+1 “
                            DUP( LUNGH˙ENTRATA DUP(' '), '$')
RICERCA˙ENTRATA      DB       LUNGH˙ENTRATA DUP(' '), '$'
NUM˙ENTRATE          DB       ?
DSEG               ENDS


STACKM               SEGMENT PARA STACK 'STACK'  ;Viene allocata una zona di
                   DB       64 DUP('12345678')  ; memoria per lo Stack: in
STACKM               ENDS                        ; tutto 64*8 bytes.


                   ASSUME CS:CSEG,DS:DSEG,SS:STACKM
CSEG                 SEGMENT PARA PUBLIC 'CODE'

;-----------------------------------------------------------------;
;                    Corpo principale del programma               ;
;-----------------------------------------------------------------;

MAIN                 PROC   FAR
                   PUSH   DS                 ;Istruzioni da lasciare SEMPRE
                   MOV    AX,00h             ; al principio dei programmi!
                   PUSH   AX                 ;
                   CALL   INIZIALIZZAZIONE
                   CALL   PRESENTAZIONE
CICLO˙PRINCIPALE: CALL   LEGGI˙SCELTA        ;Legge la scelta da tastiera
                                             ; e pone il risultato in AL
                   CMP    AL, '1'            ;Se il tasto non e' '1',
```

```
                  JNZ    NO_INSERIMENTO    ; allora salta
                  CALL   INSERIMENTO       ;Procedura di inserimento
                  MOV    DX, OFFSET ISTRUZIONI ;Stampa le istruzioni
                  CALL   STAMPA_STRINGA
                  JMP    CICLO_PRINCIPALE
NO_INSERIMENTO:   CMP    AL, '2'           ;Se il tasto non e' '2',
                  JNZ    USCITA            ; allora esce
                  CALL   RICERCA
                  MOV    DX, OFFSET ISTRUZIONI ;Stampa le istruzioni
                  CALL   STAMPA_STRINGA
                  JMP    CICLO_PRINCIPALE ;Ritorno al ciclo principale
USCITA:           RET                      ;Ritorno al Sistema Operativo
MAIN              ENDP


LEGGI_SCELTA      PROC   NEAR
NUOVA_LETTURA:    MOV    AH,08h            ;Servizio DOS 'Read Keyboard Char
                  INT    21h               ; Without Echo'
                  CMP    AL,'1'            ;Controlla il range: se e'
                  JC     NUOVA_LETTURA     ; compreso tra '1' e '3', allora
                  CMP    AL,'3'+1          ; esce, altrimenti ritorna
                  JNC    NUOVA_LETTURA     ; a NUOVA_LETTURA
                  RET
LEGGI_SCELTA      ENDP


INSERIMENTO       PROC   NEAR
                  MOV    BL, [NUM_ENTRATE]
                  CMP    BL, MAX_NUM_ENTRATE ;Se il numero di entrate e'
                  JZ     FINE_INSERIMENTO ; troppo elevato, esce subito
                  MOV    AL, MAX_LUNGH_ENTRATA
                  MUL    BL
                  ADD    AX, OFFSET ENTRATE
                  MOV    BX, AX            ;Punta alla prima parola vuota
                  PUSH   BX                ;Salva il puntatore alla entrata
                  MOV    DX, OFFSET INSERIM_UTENTE
                  CALL   STAMPA_STRINGA
                  CALL   LEGGI_UTENTE      ;Legge un utente
                  POP    BX                ;Ripristina il puntatore all'utente
                  ADD    BX,LUNGH_UTENTE ; e lo fa puntare al telefono
                  MOV    DX, OFFSET INSERIM_TELEFONO
                  CALL   STAMPA_STRINGA
                  CALL   LEGGI_TELEFONO   ;Legge il numero di telefono
                  INC    BYTE PTR [NUM_ENTRATE] ;Aggiorna il numero di parole
```

```
FINE_INSERIMENTO: RET
INSERIMENTO       ENDP


LEGGI_UTENTE      PROC  NEAR
                  MOV   CX,LUNGH_UTENTE ;Inizializza il contatore caratteri
NUOVA_LETT:       MOV   AH,08h         ;Servizio DOS 'Read Keyboard Char
                  INT   21h            ; Without Echo'
                  CMP   AL, ENTER      ;Se il tasto e' enter,
                  JZ    FINE_LETTURA   ; termina la lettura
                  CMP   AL,' '         ;Controlla il range: se non e'
                  JC    NUOVA_LETT     ; valido, rilegge un altro tasto
                  PUSH  AX             ;Salva il registro AX
                  MOV   AH,02h         ;Servizio DOS 'Character Output'
                  MOV   DL, AL         ;Carattere da stampare
                  INT   21h
                  POP   AX             ;Ripristina il contenuto di AX
                  MOV   [BX], AL       ;Memorizza il tasto
                  INC   BX             ;Aggiorna il puntatore
                  LOOP  NUOVA_LETT     ;Reitera fino a che CX=0
FINE_LETTURA:     RET
LEGGI_UTENTE      ENDP


LEGGI_TELEFONO    PROC  NEAR
                  MOV   CX,LUNGH_TELEFONO ;Inizializza contatore caratteri
NUOVA_LETT2:      MOV   AH,08h         ;Servizio DOS 'Read Keyboard Char
                  INT   21h            ; Without Echo'
                  CMP   AL, ENTER      ;Se il tasto e' enter,
                  JZ    FINE_LETTURA2  ; termina la lettura
                  CMP   AL,'0'         ;Controlla il range: se non e'
                  JC    NUOVA_LETT2    ; compreso tra '0' e '9',
                  CMP   AL,'9'+1       ; allora rilegge un altro tasto
                  JNC   NUOVA_LETT2
                  PUSH  AX             ;Salva il registro AX
                  MOV   AH,02h         ;Servizio DOS 'Character Output'
                  MOV   DL, AL         ;Carattere da stampare
                  INT   21h
                  POP   AX             ;Ripristina il contenuto di AX
                  MOV   [BX], AL       ;Memorizza il tasto
                  INC   BX             ;Aggiorna il puntatore
                  LOOP  NUOVA_LETT2    ;Reitera fino a che CX=0
FINE_LETTURA2:    RET
LEGGI_TELEFONO    ENDP
```

```
RICERCA              PROC  NEAR
                    MOV   AL, [NUM˙ENTRATE] ;Se non e' stato inserito nessun
                    CMP   AL,0              ; utente, allora salta al termine
                    JZ    FINE˙RICERCA      ; della ricerca
                    MOV   DX, OFFSET MESS˙RICERCA
                    CALL  STAMPA˙STRINGA    ;Stampa messaggio di ricerca
                    MOV   BX, OFFSET RICERCA˙ENTRATA
                    CALL  LEGGI˙UTENTE      ;Legge un utente
                    MOV   CX, BX
                    SUB   CX, OFFSET RICERCA˙ENTRATA ;Se e' stato premuto solo
                    JZ    FINE˙RICERCA      ; ENTER, termina la ricerca.
                    CLD
                    MOV   BL, [NUM˙ENTRATE]
                    MOV   DI, OFFSET ENTRATE
CICLO˙RICERCA:       PUSH  CX               ;Salva la lunghezza della stringa
                    PUSH  DI
                    MOV   SI, OFFSET RICERCA˙ENTRATA
                    REP   CMPSB             ;Confronta le stringhe di bytes
                    JE    CONFRONTO˙OK
                    POP   DI                ;Ripristina puntatore utenti
                    POP   CX                ;Ripristina lunghezza stringa
                    ADD   DI, MAX˙LUNGH˙ENTRATA
                    DEC   BL                ;Decrementa contatore confronti
                    JNZ   CICLO˙RICERCA     ;Continua la ricerca se BL!=0
                    MOV   DX, OFFSET NON˙TROVATO
                    CALL  STAMPA˙STRINGA    ;Stampa stringa 'Non Trovato'
                    JMP   FINE˙RICERCA      ;Termina la ricerca
CONFRONTO˙OK:         MOV   DX, OFFSET TROVATO
                    CALL  STAMPA˙STRINGA    ;Stampa stringa 'Trovato'
                    POP   DX                ;Punta inizio utente trovato
                    POP   CX
                    ADD   DX,LUNGH˙UTENTE   ;Punta al suo numero telefonico
                    CALL  STAMPA˙STRINGA    ; e lo stampa
FINE˙RICERCA:        RET
RICERCA              ENDP


STAMPA˙STRINGA        PROC  NEAR
                    MOV   AH,09h            ;Servizio DOS 'Print String'
                    INT   21h
```

```
                    RET
STAMPA_STRINGA      ENDP



INIZIALIZZAZIONE  PROC  NEAR
                    MOV     AX, SEG DSEG
                    MOV     DS, AX
                    MOV     ES, AX
                    MOV     [NUM_ENTRATE], 0 ;Numero di utenti inseriti
                    RET
INIZIALIZZAZIONE  ENDP


PRESENTAZIONE       PROC  NEAR
                    MOV     DX, OFFSET ISTRUZIONI ;Stampa le istruzioni
                    CALL    STAMPA_STRINGA
                    RET
PRESENTAZIONE       ENDP



CSEG                 ENDS
                    END   MAIN
```

CALCOLATORI ELETTRONICI

Prova scritta

25 novembre 1991

Scrivere un programma in assembler in grado di gestire un'agenda giornaliera. L'agenda deve memorizzare gli impegni orari di una giornata di lavoro dalle ore 9.00 alle ore 17.00. Il programma deve:

- Presentarsi visualizzando gli impegni nel formato seguente:
  9.00 ....
  10.00 ....
  11.00 ....
  ....
  17.00 ....
- Permettere l'inserimento di nuovi impegni e al termine la visualizzazione degli stessi.
- Permettere la cancellazione di tutti gli impegni.
- Permettere di uscire dal programma.

L'impegno orario e' rappresentato da un stringa alfanumerica (max 50 caratteri).

Traccia per la risoluzione

```
                    TITLE  -  Prova Scritta del 25 novembre 1991

DSEG              SEGMENT PARA PUBLIC 'DATA'
LEN_IMPEGNO         EQU    58                  ;Lunghezza di una stringa
LEN_STRINGA         EQU    50
NUM_IMPEGNI         EQU    9
ENTER               EQU    13
```

```
LINEFEED          EQU     10
AGENDA             DB      ENTER, LINEFEED
                DB      '                    AGENDA GIORNALIERA'
                DB      ENTER, LINEFEED
IMPEGNI          DB       ' 9:00 ', LEN_STRINGA DUP('.'), ENTER, LINEFEED
                DB      '10:00 ', LEN_STRINGA DUP('.'), ENTER, LINEFEED
                DB      '11:00 ', LEN_STRINGA DUP('.'), ENTER, LINEFEED
                DB      '12:00 ', LEN_STRINGA DUP('.'), ENTER, LINEFEED
                DB      '13:00 ', LEN_STRINGA DUP('.'), ENTER, LINEFEED
                DB      '14:00 ', LEN_STRINGA DUP('.'), ENTER, LINEFEED
                DB      '15:00 ', LEN_STRINGA DUP('.'), ENTER, LINEFEED
                DB      '16:00 ', LEN_STRINGA DUP('.'), ENTER, LINEFEED
                DB      '17:00 ', LEN_STRINGA DUP('.'), ENTER, LINEFEED
                DB      '$'                  ;Termine stringa
ISTRUZIONI        DB      ENTER, LINEFEED, ENTER, ENTER, ENTER
                DB      'Premere da "A" a "I" per inserire;'
                DB      ENTER, LINEFEED
                DB      '            "J" per cancellare;',ENTER,LINEFEED
                DB      '            "K" per uscire.', ENTER, LINEFEED
                DB      ENTER, 'Scelta: $'
MESS_INSERIMENTO  DB       ENTER, LINEFEED
                DB      'Stringa da inserire: $'
DSEG             ENDS


STACKM             SEGMENT PARA STACK 'STACK'  ;Viene allocata una zona di
                DB       64 DUP('12345678')  ; memoria per lo Stack: in
STACKM             ENDS                      ; tutto 64*8 bytes.


                ASSUME CS:CSEG,DS:DSEG,SS:STACKM
CSEG              SEGMENT PARA PUBLIC 'CODE'

;------------------------------------------------------------------;
;                  Corpo principale del programma                  ;
;------------------------------------------------------------------;

MAIN              PROC  FAR
                PUSH  DS               ;Istruzioni da lasciare SEMPRE
                MOV   AX,00h           ; al principio dei programmi!
                PUSH  AX               ;
                CALL  INIZIALIZZAZIONE
```

```
                  CALL   PRESENTAZIONE
CICLO_PRINCIPALE: CALL   STAMPA_VIDEATA
                  CALL   LEGGI_SCELTA     ;Legge la scelta da tastiera
                                          ; e pone il risultato in AL
                  CMP    AL, 'J'          ;Se il tasto e' >= 'J', allora
                  JNC    NO_INSERIMENTO   ; allora non si inserisce
                  CALL   INSERIMENTO      ;Procedura di inserimento
                  JP     CICLO_PRINCIPALE
NO_INSERIMENTO:   JNZ    USCITA           ;Se il tasto e' 'K', esci
                  CALL   CANCELLAZIONE
                  JMP    CICLO_PRINCIPALE
USCITA:           RET                     ;Ritorno al Sistema Operativo
MAIN              ENDP


LEGGI_SCELTA      PROC   NEAR
NUOVA_LETTURA:    CALL   LEGGI_CHAR
                  AND    AL, 0DFh         ;Riporta a maiuscolo
                  CMP    AL,'A'           ;Controlla il range: se e'
                  JC     NUOVA_LETTURA    ; compreso tra 'A' e 'K', allora
                  CMP    AL,'L'           ; esce, altrimenti ritorna
                  JNC    NUOVA_LETTURA    ; a NUOVA_LETTURA
                  RET
LEGGI_SCELTA      ENDP


INSERIMENTO       PROC   NEAR
                  PUSH   AX               ;Salva AX
                  MOV    DX, OFFSET MESS_INSERIMENTO
                  CALL   STAMPA_STRINGA   ;Stampa messaggio richiesta inserim.
                  POP    AX               ;Ripristina AX
                  MOV    BX, OFFSET IMPEGNI ;Puntatore alla agenda
                  SUB    AL, 'A'          ;AL = numero della scelta
                  MOV    CL, LEN_IMPEGNO  ;CL = lunghezza di un
                  MUL    CL
                  ADD    BX, AX           ;BX punta all'inizio dell'impegno
                  ADD    BX, 6            ;BX punta alla zona di inserimento
                  MOV    CX, LEN_STRINGA  ;Contatore di max caratteri
CICLO_INSERIM:    CALL   LEGGI_CHAR       ;Legge un carattere e lo pone in AL
                  CMP    AL, ENTER        ;Se si preme ENTER,
                  JZ     FINE_INSERIM     ; si termina l'inserimento
                  MOV    [BX], AL         ;Memorizza il carattere
                  INC    BX               ;Incrementa il puntatore
                  LOOP   CICLO_INSERIM    ;Continua fino a che CX=0
```

```
FINE˙INSERIM:    RET
INSERIMENTO       ENDP


CANCELLAZIONE      PROC   NEAR
                  MOV    BX, OFFSET IMPEGNI ;Inizio zona di memoria
                  MOV    CX, NUM˙IMPEGNI ;Contatore numero di impegni
CICLO˙CANCELL:     ADD    BX, 6             ;Sposta inizio dopo le ore
                  MOV    DX, LEN˙STRINGA ;Contatore lunghezza stringa
CICLO˙CANC˙2:      MOV    BYTE PTR [BX], '.' ;Cancella un carattere
                  INC    BX                ;Incrementa il puntatore
                  DEC    DX                ;Decrementa contatore stringa
                  JNZ    CICLO˙CANC˙2      ;Itera se richiesto
                  ADD    BX, 2             ;Aggiorna il puntatore
                  LOOP   CICLO˙CANCELL     ;Continua fino a che CX=0
                  RET
CANCELLAZIONE      ENDP


STAMPA˙VIDEATA     PROC   NEAR
                  MOV    AX, 0700h         ;Invoca il 'Clear Screen Service'
                  MOV    CX, 0000h         ;Finestra da cancellare: CX e DX
                  MOV    DX, 1850h         ; indicano le coords del carattere
                  MOV    BH, 7             ;Attributo caratteri
                  INT    10h               ; in alto a sx e in basso a dx
                  MOV    AH, 02h           ;Invoca il 'Set Cursor Position'
                  MOV    BH, 00h           ;Pagina video attiva
                  XOR    DX, DX            ;Azzera DX
                  INT    10h
                  MOV    DX, OFFSET AGENDA ;Punta alla stringa
                  CALL   STAMPA˙STRINGA    ; degli impegni e la stampa
                  MOV    DX, OFFSET ISTRUZIONI
                  CALL   STAMPA˙STRINGA    ;Stampa le istruzioni
                  RET
STAMPA˙VIDEATA     ENDP


STAMPA˙STRINGA     PROC   NEAR
                  MOV    AH,09h            ;Servizio DOS 'Print String'
                  INT    21h
                  RET
STAMPA˙STRINGA     ENDP
```

```
LEGGI_CHAR          PROC   NEAR
                    MOV    AH,01h          ;Servizio DOS 'Read Keyboard Char'
                    INT    21h             ;
                    RET
LEGGI_CHAR          ENDP


INIZIALIZZAZIONE    PROC   NEAR
                    MOV    AX, SEG DSEG
                    MOV    DS, AX
                    RET
INIZIALIZZAZIONE    ENDP


PRESENTAZIONE       PROC   NEAR
                    ...
                    RET
PRESENTAZIONE       ENDP



CSEG                ENDS
                    END    MAIN
```

CALCOLATORI ELETTRONICI

Prova scritta

10 febbraio 1992

Scrivere un programma in assembler che effettui il prodotto di due matrici *A* e *B* e dia come risultato la matrice *C*. Il programma deve:

- Accettare da tastiera le due matrici (gli elementi delle matrici *A* e *B*, di dimensione 2x2, sono numeri positivi inferiori a 10).
- Presentare su video la matrice *C* risultato del prodotto.

Scrivere successivamente lo stesso programma nel caso di dimensione 3x3.

Traccia per la risoluzione

```
                  TITLE  -  Prova Scritta del 10 febbraio 1992


DSEG            SEGMENT PARA PUBLIC 'DATA'
DIM_MAT          EQU     3                   ;Risolto nel caso 3x3
MATRICE_A        DB      DIM_MAT * DIM_MAT DUP(?)
MATRICE_B        DB      DIM_MAT * DIM_MAT DUP(?)
MATRICE_C        DB      DIM_MAT * DIM_MAT DUP(?)
DSEG            ENDS


STACKM            SEGMENT PARA STACK 'STACK'  ;Viene allocata una zona di
                DB      64 DUP('12345678')  ; memoria per lo Stack: in
STACKM            ENDS                       ; tutto 64*8 bytes.
```

```
                ASSUME CS:CSEG,DS:DSEG,SS:STACKM
CSEG            SEGMENT PARA PUBLIC 'CODE'

;------------------------------------------------------------------;
;                Corpo principale del programma                    ;
;------------------------------------------------------------------;

MAIN            PROC  FAR
                PUSH  DS               ;Istruzioni da lasciare SEMPRE
                MOV   AX,00h           ; al principio dei programmi!
                PUSH  AX               ;
                CALL  INIZIALIZZAZIONE
                CALL  PRESENTAZIONE
                MOV   BX, OFFSET MATRICE˙A
                CALL  LEGGI˙MATRICE    ;Legge la prima matrice
                MOV   BX, OFFSET MATRICE˙B
                CALL  LEGGI˙MATRICE    ;Legge la seconda matrice
                CALL  MOLTIPLICAZIONE  ;Moltiplica le due matrici
                MOV   BX, OFFSET MATRICE˙C
                CALL  STAMPA˙MATRICE   ;Stampa la matrice risultato
                RET                    ;Ritorno al Sistema Operativo
MAIN            ENDP


MOLTIPLICAZIONE   PROC  NEAR
                MOV   SI, OFFSET MATRICE˙A
                MOV   DI, OFFSET MATRICE˙B
                MOV   BX, OFFSET MATRICE˙C
                MOV   CL, DIM˙MAT      ;Contatore righe in CL
CICLO˙1:        MOV   CH, DIM˙MAT      ;Contatore colonne in CH
CICLO˙2:        MOV   DH, DIM˙MAT      ;Contatore della moltiplicazione
                PUSH  SI
                PUSH  DI
                XOR   DL, DL           ;Azzera il risultato parziale in DL
CICLO˙3:        MOV   AL, [SI]         ;Carica elemento di A
                MUL   BYTE PTR [DI]    ;Moltiplica per elemento di B
                ADD   DL, AL           ;Somma il risultato parziale
                INC   SI               ;Incrementa puntatore matrice A
                ADD   DI, DIM˙MAT      ;Incrementa puntatore matrice B
                DEC   DH               ;Decrementa il contatore del ciclo
                JNZ   CICLO˙3          ; di moltiplicazione
                POP   DI
```

```
                POP    SI
                INC    DI                ;Aggiusta puntatore matrice B
                MOV    DS:[BX], DL       ;Memorizza il rislutato
                INC    BX                ;Incrementa puntatore matrice C
                DEC    CH                ;Decrementa contatore colonne
                JNZ    CICLO˙2
                ADD    SI, DIM˙MAT       ;Aggiusta puntatore matrice A
                SUB    DI, DIM˙MAT       ;Aggiusta puntatore matrice B
                LOOP   CICLO˙1
                RET
MOLTIPLICAZIONE   ENDP


LEGGI˙MATRICE     PROC   NEAR
                MOV    CX, DIM˙MAT*DIM˙MAT ;Numero di entrate
CICLO˙LETTURA:    CALL   LEGGE˙NUMERO      ;Legge il numero e lo pone in AL
                MOV    [BX], AL          ;Memorizza la lettura
                INC    BX                ;Incrementa il puntatore
                LOOP   CICLO˙LETTURA     ;Continua fino a che CX=0
                RET
LEGGI˙MATRICE     ENDP


STAMPA˙MATRICE    PROC   NEAR
                MOV    CX, DIM˙MAT*DIM˙MAT ;Numero di entrate
CICLO˙STAMPA:     MOV    AL, [BX]          ;Legge il numero
                CALL   STAMPA˙NUMERO     ;Stampa il contenuto di AL
                INC    BX                ;Incrementa il puntatore
                LOOP   CICLO˙STAMPA      ;Continua fino a che CX=0
                RET
STAMPA˙MATRICE    ENDP


LEGGE˙NUMERO      PROC   NEAR
NUOVA˙LETTURA:    MOV    AH,07h            ;Servizio DOS 'Read Keyboard Char
                INT    21h               ; Without Echo'
                CMP    AL,'0'            ;Se il tasto premuto e' ¡ '0'
                JC     NUOVA˙LETTURA     ; legge un altro tasto
                CMP    AL,'9'+1          ;Se il tasto e' ¿ '9' ne legge
                JNC    NUOVA˙LETTURA     ; un altro
FINE˙LETTURA:     SUB    AL, '0'           ;Sottrae la base dei numeri in ASCII
                RET
LEGGE˙NUMERO      ENDP
```

```
INIZIALIZZAZIONE  PROC  NEAR
                  MOV   AX, SEG DSEG
                  MOV   DS, AX
                  RET
INIZIALIZZAZIONE  ENDP


STAMPA_NUMERO       PROC  NEAR
                  ...
                  RET
STAMPA_NUMERO       ENDP


PRESENTAZIONE       PROC  NEAR
                  ...
                  RET
PRESENTAZIONE       ENDP



CSEG                ENDS
                  END   MAIN
```

CALCOLATORI ELETTRONICI

Prova scritta

18 febbraio 1992

Scrivere un programma assembler che calcoli i primi 400 valori della sequenza generata con la seguente legge:

$$x_{i+1} = |ax_i + b|_{mod 256}$$

dove a e b sono numeri compresi fra 0 e 255.
Il programma deve:

- Acquisire i valori di a e b;
- Visualizzare la sequenza su 20 righe di 20 numeri con il seguente formato:

$$bbb1bb25b145bb30b100bbb0...$$

Traccia per la risoluzione

```
                    TITLE  -  Prova Scritta del 18 febbraio 1992

DSEG                 SEGMENT PARA PUBLIC 'DATA'
NUM_ITERAZIONI         EQU      400
X0                   EQU      0
A                    DB       ?
B                    DB       ?
MASCHERA               DB       ?
DSEG                 ENDS

STACKM                 SEGMENT PARA STACK 'STACK'  ;Viene allocata una zona di
                    DB       64 DUP('12345678')   ; memoria per lo Stack: in
STACKM                 ENDS                        ; tutto 64*8 bytes.
```

```
                ASSUME CS:CSEG,DS:DSEG,SS:STACKM
CSEG             SEGMENT PARA PUBLIC 'CODE'

;-------------------------------------------------------------;
;                Corpo principale del programma               ;
;-------------------------------------------------------------;

MAIN             PROC   FAR
                PUSH   DS              ;Istruzioni da lasciare SEMPRE
                MOV    AX,00h          ; al principio dei programmi!
                PUSH   AX              ;
                CALL   INIZIALIZZAZIONE
                CALL   PRESENTAZIONE
                CALL   LEGGI_NUMERO      ;Legge un numero e lo pone in AL
                MOV    [A], AL           ;Lo memorizza in A
                CALL   LEGGI_NUMERO      ;Legge un numero e lo pone in AL
                MOV    [B], AL         ;Lo memorizza in B
                MOV    AL, X0           ;Inizializza X0 in AL
                MOV    CX, NUM_ITERAZIONI ;Contatore
CICLO_PRINCIPALE: PUSH   CX              ;Salva CX
                PUSH   AX
                CALL   STAMPA_AX        ;Stampa il numero in AL
                POP    AX
                MUL    BYTE PTR [A]     ; AX = [A]*AL
                ADD    AL, BYTE PTR [B]; AL = ( ([A]*AL) + [B] ) mod 256
                POP    CX               ;Ripristina CX
                LOOP   CICLO_PRINCIPALE
                RET                    ;Ritorno al Sistema Operativo
MAIN             ENDP


STAMPA_AX         PROC   NEAR
                CALL   STAMPA_BIANCO    ;Stampa il primo carattere bianco
                XOR    AH,AH            ;Azzera il registro AH
                MOV    [MASCHERA],0     ;Azzera la maschera
                MOV    CL, 100
                CALL   STAMPA_CARATT
                MOV    CL, 10
                CALL   STAMPA_CARATT
                MOV    [MASCHERA],1
                CALL   STAMPA_NUMERO
                RET                    ;Ritorno alla procedura chiamante
STAMPA_AX         ENDP
```

```
STAMPA_CARATT      PROC   NEAR
                   PUSH   AX
                   IDIV   CL
                   OR     [MASCHERA], AL  ;Non appena AL e' diverso da 0,
                                          ; la MASCHERA diventa 'non 0'.
                   CALL   STAMPA_NUMERO    ; Stampa AL
                   MUL    CL
                   MOV    BL, AL
                   POP    AX
                   SUB    AL, BL
                   RET
STAMPA_CARATT      ENDP


STAMPA_NUMERO      PROC   NEAR
                   PUSH   AX                ;Salva AX
                   MOV    BL, AL            ;Se AL e la MASCHERA sono 0,
                   OR     BL,[MASCHERA]     ; stampa un bianco ed esce.
                   JNZ    STAMPA_NUM
                   CALL   STAMPA_BIANCO
                   JP     FINE_STAMPA
STAMPA_NUM:        ADD    AL,'0'            ;La base da cui partire e' '0'
                   CALL   STAMPA_CHAR       ;Stampa il carattere in AL
FINE_STAMPA:       POP    AX                ;Ripristina AX
                   RET
STAMPA_NUMERO      ENDP


STAMPA_BIANCO      PROC   NEAR
                   PUSH   AX                ;Salva AX
                   MOV    AL,' '            ;Carica in AL il codice di ' '
                   CALL   STAMPA_CHAR       ;Stampa il carattere in AL
                   POP    AX                ;Ripristina AX
                   RET
STAMPA_BIANCO      ENDP


STAMPA_CHAR        PROC   NEAR
                   PUSH   CX                ;Salva CX
                   MOV    AH,0Eh            ;Servizio BIOS 'Write Char'
                   MOV    BL,0              ;Pagina attiva
                   MOV    CX,0001h          ;Stampa un solo carattere
```

```
                INT     10h
                POP     CX              ;Ripristina CX
                RET                     ;Ritorno alla procedura chiamante
STAMPA_CHAR         ENDP


INIZIALIZZAZIONE  PROC   NEAR
                MOV     AX, SEG DSEG
                MOV     DS, AX
                RET
INIZIALIZZAZIONE  ENDP


LEGGI_NUMERO        PROC   NEAR
                ...
                RET
LEGGI_NUMERO        ENDP


PRESENTAZIONE       PROC   NEAR
                ...
                RET
PRESENTAZIONE       ENDP



CSEG                  ENDS
                END    MAIN
```

CALCOLATORI ELETTRONICI

Prova scritta

7 aprile 1992

Scrivere un programma in assembler che, dato un numero compreso fra 0 e 99, calcoli tutte le potenze dello stesso fino a raggiungere il numero massimo contenuto in 32 bit. Il programma deve:

- Accettare da tastiera il numero.
- Presentare su video la serie di potenze significative.

Traccia per la risoluzione

```
                TITLE  -  Prova Scritta del 7 aprile 1992

DSEG            SEGMENT PARA PUBLIC 'DATA'
NUMERO            DW      ?
DSEG            ENDS

STACKM            SEGMENT PARA STACK 'STACK'  ;Viene allocata una zona di
                DB      64 DUP('12345678')  ; memoria per lo Stack: in
STACKM            ENDS                        ; tutto 64*8 bytes.

                ASSUME CS:CSEG,DS:DSEG,SS:STACKM
CSEG            SEGMENT PARA PUBLIC 'CODE'

;-------------------------------------------------------------------;
;                   Corpo principale del programma                  ;
;-------------------------------------------------------------------;
```

```
MAIN              PROC   FAR
                 PUSH   DS                ;Istruzioni da lasciare SEMPRE
                 MOV    AX,00h            ; al principio dei programmi!
                 PUSH   AX                ;
                 CALL   INIZIALIZZAZIONE
                 CALL   PRESENTAZIONE
                 CALL   LEGGI_NUMERO        ;Legge il numero e lo memorizza
                                          ; nel registro AX.
                 MOV    NUMERO, AX          ;Memorizza il numero.
                 MOV    DX, 0               ;Azzera la parte alta dei 32 bit
                                          ; DX:AX per la moltiplicazione.
CICLO_PRINCIPALE: CALL  STAMPA_NUMERO       ;Stampa il numero in DX:AX.
                 CALL   MOLTIPLICAZIONE ;Esegue [NUMERO]*DX:AX -> DX:AX
                                          ; ritornando lo Zero Flag ad 1
                                          ; se deve reiterare.
                 JZ     CICLO_PRINCIPALE
                 RET                        ;Ritorno al Sistema Operativo
MAIN              ENDP


MOLTIPLICAZIONE   PROC   NEAR
                 PUSH   AX                  ;Salva i 16 bit di AX sullo Stack
                 MOV    AX, DX              ;Moltiplica i 16 bit piu' signifi-
                 MUL    NUMERO              ; cativi per il NUMERO -> DX:AX
                 XCHG   DX, CX              ;Pone il risultato in CX:AX
                 XCHG   BX, AX              ;Pone il risultato in CX:BX
                 POP    AX                  ;Ripristina i 16 bit meno signif.
                 MUL    NUMERO              ; e li moltiplica per NUMERO, po-
                                          ; nendo il risultato in DX:AX
                                          ;Il risultato finale andra' in:
                 ADD    DX, BX              ;           DX:AX   +
                                          ;        CX:BX      =
                                          ;   ---------------
                                          ;        CX:DX:AX
                 ADC    CX, 0               ;Se CX e' uguale a zero, pone lo
                                          ; Zero Flag ad uno per reiterare
                 RET
MOLTIPLICAZIONE   ENDP
```

```
INIZIALIZZAZIONE  PROC   NEAR
                  MOV    AX, SEG DSEG
                  MOV    DS, AX
                  RET
INIZIALIZZAZIONE  ENDP

PRESENTAZIONE     PROC   NEAR
                  ...
                  RET
PRESENTAZIONE     ENDP

LEGGI_NUMERO      PROC   NEAR
                  ...
                  RET
LEGGI_NUMERO      ENDP

STAMPA_NUMERO     PROC   NEAR
                  ...
                  RET
STAMPA_NUMERO     ENDP

CSEG              ENDS
                  END    MAIN
```

CALCOLATORI ELETTRONICI

Prova scritta

1 giugno 1992

Scrivere un programma assembler che accetti un numero decimale in base 10 con il seguente formato: zz,zzz e ne calcoli la trasformazione in base 2 con il formato xxxxx,xxxxx. La trasformazione è ovviamente approssimata.
Il programma deve stampare come output:

- La trasformazione in base 2 xxxxx,xxxxx.
- La ritrasformazione in base 10 esatta del numero in base 2 calcolato al punto precedente.

CALCOLATORI ELETTRONICI

Prova scritta

17 luglio 1992

Scrivere un programma in assembler che, dato un numero compreso fra 0 e 999, lo scompomga nei suoi fattori primi. Il programma deve:

- Accettare da tastiera il numero.
- Presentare su video la serie dei suoi fattori primi con la relativa molteplicità.

Esempio: 150 = 2 3 5(2)

Traccia per la risoluzione

```
                TITLE  -  Prova Scritta del 17 luglio 1992

DSEG                SEGMENT PARA PUBLIC 'DATA'
MAX_NUM_FATTORI     EQU     17
DATI                DW      MAX_NUM_FATTORI DUP(?)
DSEG                ENDS


STACKM              SEGMENT PARA STACK 'STACK'  ;Viene allocata una zona di
                DB      64 DUP('12345678')  ; memoria per lo Stack: in
STACKM              ENDS                        ; tutto 64*8 bytes.


                ASSUME CS:CSEG,DS:DSEG,SS:STACKM
CSEG                SEGMENT PARA PUBLIC 'CODE'
```

```
;------------------------------------------------------------------;
;                   Corpo principale del programma                  ;
;------------------------------------------------------------------;

MAIN                PROC   FAR
                    PUSH   DS              ;Istruzioni da lasciare SEMPRE
                    MOV    AX,00h          ; al principio dei programmi!
                    PUSH   AX              ;
                    CALL   INIZIALIZZAZIONE
                    CALL   PRESENTAZIONE
                    CALL   LEGGI'NUMERO    ;Legge il numero e lo pone in AX
                    CALL   SCOMPONI        ;Scompone AX
                    CALL   STAMPA'OUTPUT   ;Stampa il risultato
                    RET                    ;Ritorno al Sistema Operativo
MAIN                ENDP


SCOMPONI            PROC   NEAR            ;Limite:  0 ¡ AX ¡ 65535
                    MOV    BX, OFFSET DATI ;Punta all'inizio zona dati
NUOVA'SCANSIONE:    MOV    CX, 2           ;Primo fattore da controllare
CICLO'SCOMPONI:     XOR    DX, DX          ;Azzera la parte alta del dividendo
                    PUSH   AX              ;Salva il numero
                    DIV    CX              ;Divide per il fattore
                    CMP    DX, 0           ;Se la divisione non da resto nullo,
                    JNZ    PROSSIMO'FATT   ; prova con il prossimo fattore
                    MOV    [BX], CX        ;Memorizza il fattore
                    ADD    BX, 2           ;Aggiorna il puntatore
                    POP    DX              ;Toglie dallo stack un dato non
                                           ; piu' significativo
                    CMP    AX, 1           ;Reitera la procedura solo se
                    JNZ    NUOVA'SCANSIONE ; AX e' diverso da 1
                    JMP    FINE'SCOMPOSIZ
PROSSIMO'FATT:      POP    AX              ;Ripristina il dividendo
                    INC    CX              ;Prova con il prossimo fattore
                    JMP    CICLO'SCOMPONI  ;Reitera la procedura
FINE'SCOMPOSIZ:     MOV    WORD PTR [BX],0 ;Memorizza un marker per identi-
                                           ; ficare la fine dei dati
                    RET
SCOMPONI            ENDP


LEGGI'NUMERO        PROC   NEAR
                    ...                    ;Legge un numero e lo pone in AX
```

```
                RET
LEGGI_NUMERO        ENDP


STAMPA_OUTPUT       PROC   NEAR
                MOV    BX, OFFSET DATI+2 ;Puntatore alla zona dati+2
                MOV    DX, [BX-2]        ;Legge il primo fattore
NUOVA_MOLTEPL:     MOV    CH, 1             ;Pone ad 1 la molteplicita'
CICLO_LETTURA:     MOV    AX, [BX]          ;Legge il fattore
                ADD    BX, 2             ;Punta al prossimo fattore
                CMP    AX, DX            ;Lo confronta con il precedente:
                JNZ    NUOVO_FATT        ; se diversi, va a stampare
                INC    CH                ;Se uguali, inc. la molteplicita'
                JMP    CICLO_LETTURA     ; e reitera la procedura
NUOVO_FATT:        CALL   STAMPA_FATTORE    ;Stampa: ' DX(CH) '
                MOV    DX, AX            ;Pone il nuovo fattore in DX
                CMP    DX, 0             ;Se non e' al termine dei dati,
                JNZ    NUOVA_MOLTEPL     ; reitera la procedura
                RET                      ; altrimenti esce
STAMPA_OUTPUT       ENDP


STAMPA_FATTORE      PROC   NEAR              ;DX contiene il numero da stampare
                ...                      ; mentre CH la molteplicita'
                RET
STAMPA_FATTORE      ENDP


INIZIALIZZAZIONE  PROC   NEAR
                MOV    AX, SEG DSEG
                MOV    DS, AX
                RET
INIZIALIZZAZIONE  ENDP


PRESENTAZIONE       PROC   NEAR
                ...
                RET
PRESENTAZIONE       ENDP



CSEG               ENDS
                END    MAIN
```

CALCOLATORI ELETTRONICI

Prova scritta

17 settembre 1992

Scrivere un programma in assembler che esegua la trasformazione fra numeri in basi diverse. In particolare il programma deve:

1. Accettare da tastiera un numero K che rappresenti la base della numerazione del primo numero.

2. Accettare da tastiera un numero N nella base prima indicata.

3. Accettare da tastiera un numero L che rappresenti la base di numerazione nella quale occorre trasformare il numero N.

Il programma deve fornire il numero specificato ai punti 1) e 2) nella base indicata in 3). Sono da considerare i seguenti limiti:

1. $2 \leq K, L \leq 20$.

2. Il valore del numero rappresentato da N(base$K$) deve essere minore di 65536 (decimale) = 2^{16}; in caso contrario il programma deve segnalare errore.

3. I numeri da usare per K ed L sono:

$$2 \rightarrow 2, \ldots 9 \rightarrow 9, 10 \rightarrow A, \ldots 20 \rightarrow K.$$

4. Gli stessi simboli devono essere usati per rappresentare i numeri.
   Esempio: 23B (base C) = 335 (base A).

E' facoltativo estendere la condizione 2 fino a 2^{32}

Traccia per la risoluzione

```
                TITLE  -  Prova Scritta del 17 settembre 1992

DSEG            SEGMENT PARA PUBLIC 'DATA'
ENTER           EQU     13
LINEFEED        EQU     10
NUMERO          DW      ?
ERRORE          DB      ENTER, LINEFEED, LINEFEED
                DB      'Il numero inserito e'' maggiore di 16 bit$'
INSER˙BASE      DB      ENTER, LINEFEED, LINEFEED
                DB      'Inserire la base: $'
INSER˙NUMERO    DB      ENTER, LINEFEED, LINEFEED
                DB      'Inserire il numero: $'
MESS˙OUTPUT     DB      ENTER, LINEFEED, LINEFEED
                DB      'Il numero convertito e'': $'
DSEG            ENDS


STACKM          SEGMENT PARA STACK 'STACK'  ;Viene allocata una zona di
                DB      64 DUP('12345678')  ; memoria per lo Stack: in
STACKM          ENDS                        ; tutto 64*8 bytes.


                ASSUME CS:CSEG,DS:DSEG,SS:STACKM
CSEG            SEGMENT PARA PUBLIC 'CODE'

;-------------------------------------------------------------------;
;                     Corpo principale del programma                ;
;-------------------------------------------------------------------;

MAIN            PROC  FAR
                PUSH  DS              ;Istruzioni da lasciare SEMPRE
                MOV   AX,00h          ; al principio dei programmi!
                PUSH  AX              ;
                CALL  INIZIALIZZAZIONE
                CALL  PRESENTAZIONE
                CALL  LEGGI˙BASE      ;Legge base; pone risult. in AL e AH
                CALL  LEGGI˙NUMERO    ;Legge il numero nella base
```

```
                                        ; specificata da AL e lo pone in AX
                JC      STAMPA˙ERRORE   ;Se ¿ 16 bit stampa errore ed esce
                MOV     [NUMERO], AX    ;Memorizzo il numero
                CALL    LEGGI˙BASE      ;Legge la base e lo pone in AL
                CALL    STAMPA˙NUMERO   ;Stampa [NUMERO] nella base indicata
                JMP     FINE            ; in AL ed esce
STAMPA˙ERRORE:  MOV     DX, OFFSET ERRORE ;Messaggio di errore
                CALL    STAMPA˙STRINGA  ;Stampa il messaggio
FINE:           RET                     ;Ritorno al Sistema Operativo
MAIN            ENDP


LEGGI˙BASE      PROC    NEAR
                MOV     DX, OFFSET INSER˙BASE
                CALL    STAMPA˙STRINGA  ;Stampa messaggio di inserimento
                MOV     CL, 'K'+1       ;Limite sup. carattere in lettura
NUOVO˙CARATT:   CALL    LEGGI˙CARATT    ;Leggi carattere con limite
                CMP     AL, ENTER       ;Se e' stato premuto ENTER
                JZ      NUOVO˙CARATT    ; leggi un altro carattere
                CMP     AL, '2'         ;Se la base indicata e' ¡2
                JC      NUOVO˙CARATT    ; leggi un altro carattere
                CALL    STAMPA˙CARATT   ;Stampa il carattere letto
                RET
LEGGI˙BASE      ENDP


LEGGI˙CARATT    PROC    NEAR
CICLO˙LETTURA:  MOV     AH,08h          ;Servizio DOS 'Read Keyboard Char
                INT     21h             ; Without Echo'
                CMP     AL, '9'+1       ;Se il tasto e' ¡= '9', allora
                JC      NO˙CONVERSIONE  ; salta la conversione in maiuscolo
                AND     AL, 0DFh        ;Converte in maiuscolo
NO˙CONVERSIONE: CMP     AL, ENTER       ;Se si preme ENTER,
                JZ      FINE˙LETTURA    ; termina la lettura
                CMP     AL,'0'          ;Se e' minore di '0'
                JC      CICLO˙LETTURA   ; allora rilegge il carattere
                CMP     AL, CL          ;Lo confronta con il limite
                JNC     CICLO˙LETTURA   ; posto in CL
                CMP     AL,'9'+1        ;Se e' compreso tra '0' e '9'
                JC      FINE˙LETTURA    ; allora termina la lettura
                CMP     AL,'A'          ;Se e' ¿'9' e ¡'A'
                JC      CICLO˙LETTURA   ; allora rilegge il carattere
                CMP     AL, CL          ;Lo confronta con il limite
                JNC     CICLO˙LETTURA   ; posto in CL
```

```
                CMP   AL,CL              ;Se e' maggiore di CL
                JNC   CICLO_LETTURA      ; allora rilegge il carattere
FINE_LETTURA:     MOV   AH, AL           ;Pone in AH il risultato in forma
                SUB   AH, '0'            ; non ASCII; Sottrae '0'
                CMP   AH, 10             ;Se e' minore o uguale a 9
                JC    CIFRA_DECIMALE     ; termina la lettura
                SUB   AH, 'A'-'9'-1      ;Altrimenti lo porta nel range
CIFRA_DECIMALE:   RET
LEGGI_CARATT      ENDP


STAMPA_CARATT     PROC  NEAR
                PUSH  AX                 ;Salva il contenuto di AX
                MOV   AH, 02             ;Servizio DOS 'Character Output'
                MOV   DL, AL             ;Indica il carattere da stampare
                INT   21h                ;Stampa
                POP   AX                 ;Ripristina AX
                RET
STAMPA_CARATT     ENDP


LEGGI_NUMERO      PROC  NEAR
                PUSH  AX                 ;Salva il contenuto di AX
                MOV   DX, OFFSET INSER_NUMERO
                CALL  STAMPA_STRINGA     ;Stampa messaggio di inserimento
                POP   AX                 ;Ripristina AX
                MOV   CX, AX             ;Limite sup. carattere in lettura
                                         ; in AL e valore della base in AH
                XOR   BX, BX             ;Azzera il risultato parziale
CICLO_NUMERO:     CALL  LEGGI_CARATT     ;Legge un carattere da tastiera
                CALL  STAMPA_CARATT      ;Stampa il carattere letto
                CMP   AL, ENTER          ;Se e' stato premuto ENTER, termina
                JZ    FINE_NUMERO        ; la lettura ed esce senza Carry.
                XCHG  AX, BX             ;Scambia AX e BX per la moltiplicaz.
                MOV   BL, AH             ;Salva la parte alta del risultato
                MUL   CH                 ;AX = CH*AL
                MOV   DX, AX             ;Salva il risultato parziale in DX
                MOV   AL, BL             ;Carica la parte alta del risultato
                MUL   CH                 ;AX = CH*AH
                ADD   DH, AL             ;Pone in DX il risultato finale
                JC    FINE_NUMERO        ;Se maggiore di 16 bit, termina
                CMP   AH, 1              ;Se AH e' diverso da zero, deve
                CMC                      ; uscire con il Carry ad 1
                JC    FINE_NUMERO        ;Se maggiore di 16 bit, termina
```

```
                    ADD    DL, BH          ;Somma BH al ris. parziale DX.
                    ADC    DH, 0           ; propagando il Carry
                    MOV    BX, DX          ;Pone il risultato in BX
                    JNC    CICLO_NUMERO    ;Con overflow, esce dalla routine
FINE_NUMERO:        MOV    AX, BX          ;Pone in AX il risultato
                    RET
LEGGI_NUMERO        ENDP


STAMPA_NUMERO       PROC   NEAR
                    PUSH   AX              ;Salva il contenuto di AX
                    MOV    DX, OFFSET MESS_OUTPUT
                    CALL   STAMPA_STRINGA  ;Stampa messaggio di output
                    POP    AX              ;Ripristina AX
                    ...
                    RET
STAMPA_NUMERO       ENDP


STAMPA_STRINGA      PROC   NEAR
                    MOV    AH,09h          ;Servizio DOS 'Print String'
                    INT    21h
                    RET
STAMPA_STRINGA      ENDP


INIZIALIZZAZIONE    PROC   NEAR
                    MOV    AX, SEG DSEG
                    MOV    DS, AX
                    RET
INIZIALIZZAZIONE    ENDP


PRESENTAZIONE       PROC   NEAR
                    ...
                    RET
PRESENTAZIONE       ENDP



CSEG                ENDS
                    END    MAIN
```

CALCOLATORI ELETTRONICI

Prova scritta

19 novembre 1992

Scrivere un programma in assembler che, dato un numero compreso fra 0 e 64000, ne calcoli la radice quadrata e la radice cubica con approssimazione all'intero inferiore. Il programma deve:

- Accettare da tastiera il numero.
- Presentare su video:
  - la radice quadrata e il quadrato approssimato.
  - la radice cubica e il cubo approssimato.

Esempio:
numero: *1612*
radice quadrata: *40* *1600*
radice cubica: *11* *1331*

CALCOLATORI ELETTRONICI

Prova scritta

8 febbraio 1993

Scrivere un programma in assembler che:

- Accetti da tastiera 10 numeri compresi tra -9999 e 9999.
- Li ordini in modo crescente.
- Li visualizzi sullo schermo.

In una prima fase realizzare il programma indicato considerando solo numeri positivi.

CALCALATORI ELETTRONICI

Prova scritta

6 aprile 1993/1

Scrivere un programma in assembler che muova di una posizione un cursore (qualsiasi) sullo schermo nelle quattro direzioni (nord, sud, est, ovest) utilizzando come comandi quattro caratteri diversi definiti a priori.

Il programma deve inoltre:

- partire con il cursore circa al centro dello schermo,
- quando il carattere raggiumge uno dei bordi impedire che esca dallo schermo.

Nel loop principale del programma e' opportuno inserire un ritardo; considerare la posizione piu' opportuna nel loop per il ritardo stesso.

In una prima fase realizzare il programma indicato non considerando controlli sui bordi dello schermo.

CALCOLATORI ELETTRONICI

Prova scritta

6 aprile 1993/2

Scrivere un programma in assembler che muova automaticamente un cursore (qualsiasi) sullo schermo in una delle quattro direzioni (nord, sud, est, ovest). Utilizzare quattro tasti qualsiasi per fare cambiare la direzione di spostamento del cursore.

Il programma deve inoltre:

- partire con il cursore circa al centro dello schermo,
- considerare lo schermo una struttura toroidale.

Nel loop principale del programma e' opportuno inserire un ritardo; considerare la posizione piu' opportuna nel loop per il ritardo stesso.

In una prima fase realizzare il programma indicato non considerando controlli sui bordi dello schermo.

CALCOLATORI ELETTRONICI

Prova scritta

1 giugno 1993/1

Scrivere un programma in assembler che accetti in ingresso una frase con un massimo di 500 caratteri e che di questa calcoli:

- il numero di parole (si considerano separatori: lo spazio, i segni di punteggiatura, gli apostrofi).
- la lunghezza massima e la lunghezza minima delle parole.
- la lunghezza media delle parole (questo valore deve essere calcolato con una cifra decimale significativa).

CALCOLATORI ELETTRONICI

Prova scritta

21 luglio 1993/1

Scrivere un programma in assembler che accetti in ingresso una frase con un massimo di 500 caratteri e che successivamente:

- chieda all'utilizzatore di indicare un carattere o una sequenza di caratteri (max 4 caratteri).
- risponda indicando tutte le parole nelle quali compare il carattere o la sequenza di caratteri precedentemente indicata.
- si considerino separatori fra parole gli spazi e i segni di punteggiatura: punto, virgola, due punti e punto e virgola.

CALCOLATORI ELETTRONICI

Prova scritta

15 settembre 1993

Scrivere un programma in assembler che:

- Accetti in ingresso un numero a una cifra.
- Visualizzi il numero sullo schermo con un'altezza di circa 5 cm in una schematizzazione a 7 segmenti.

Ripetere il secondo punto considerando:

- Numeri a 2 cifre.
- Numeri con un numero di cifre qualsiasi compreso fra 1 e 4.

CALCOLATORI ELETTRONICI

Prova scritta

5 ottobre 1993

Scrivere un programma in assembler che:

1. Accetti da tastiera una stringa contenente al massimo 80 caratteri.
2. Accetti da tastiera una stringa contenente al massimo 4 caratteri.
3. Determini e visualizzi il numero di ricorrenze della seconda stringa nella prima stringa.

In una seconda fase si consideri la seconda stringa formata di esattamente 4 caratteri, e si determino i numeri di ricorrenze parziali formate da solo 3 caratteri e 2 caratteri nell'ordine stabilito dalla seconda stringa (matching approssimati a 3 o 2 caratteri).

CALCOLATORI ELETTRONICI

Prova scritta

3 novembre 1993

Scrivere un programma in assembler che:

1. Accetti da tastiera una stringa contenente 8 caratteri ASCII.
2. Visualizzi tale stringa in formato esadecimale.
3. Aggiunga un bit di parità ad ogni carattere della stringa ASCII, in modo che ogni carattere sia rappresentato da 9 bit e con un numero pari di 1.
4. Stampi la stringa di bit risultante (72 bit = 8 caratteri da 9 bit) in formato esadecimale.

CALCOLATORI ELETTRONICI

Prova scritta

11 gennaio 1994

Scrivere un programma in assembler che consenta di gestire un'agenda giornaliera di impegni. La giornata è suddivisa in ore dalle 8.00 alle 20.00. Il programma deve consentire di:

- Visualizzare gli impegni sotto forma di tabella. Ad ogni ora viene associata una riga con due campi per specificare l'ora e il tipo di impegno.
- Modificare e/o inserire nuovi impegni per ogni singola ora della ora giornata.

CALCOLATORI ELETTRONICI

Prova scritta

7 febbraio 1994

Scrivere un programma in assembler che accetti in ingresso un numero di n cifre (con n fino ad un massimo di 10) che di questo calcoli:

- tutte le possibili permutazioni.
- i numeri cosi' ottenuti devono poi essere visualizzati contemporaneamente sullo schermo.

Scrivere il programma in modo che funzioni fino a n=5.

Al fine di raggiungere la sufficienza è accettabile la versione del programma per n=4.

CALCOLATORI ELETTRONICI

Prova scritta

28 febbraio 1994

Scrivere un programma in assembler che accetti in ingresso una sequenza di 16 cifre esadecimali. A partire dai caratteri di ingresso:

- formare la sequenza di 8 byte generata dal codice inserito.
- stampare la sequenza di 0 e di 1 cosi' ottenuta.
- data la stringa prima ottenuta visualizzarne una codifica *run length*.

Per codifica *run length* si intende:

- Il primo bit della sequenza entra direttamente nella codifica.
- Segue la lunghezza espressa in decimale del numero di volte che il primo bit è presente nella stringa.
- Segue la lunghezza espressa in decimale del numero di volte che il complemento compare nella stringa e cosi' via.

Esempio:
Sequenza ASCII: FFFF.0000.1010.0000
Sequenza di bit: 1111111111111111.0000000000000000.0001000000010000.0000000000000000
Codifica: 1 16 19 1 7 1 20

CALCOLATORI ELETTRONICI

Prova scritta

13 giugno 1994/1

Scrivere un programma in assembler che:

- Accetti in ingresso un numero intero positivo K ($K < 10000$).
- Calcoli la scomposizione in fattori primi del numero.

CALCOLATORI ELETTRONICI

Prova scritta

13 giugno 1994/2

Scrivere un programma in assembler che:

- Accetti in ingresso un numero intero positivo K ($K < 50000$).
- Accetti un secondo numero positivo n ($n < 200$),
- Verifichi se K e' divisibile per n:
  - Se e' divisibile stampa il risultato.
  - Se non e' divisibile stampa il primo numero superiore e il primo numero inferiore (rispetto a n) che divide K in modo esatto. (Limitarsi ai numeri compresi fra 1 e 200)

CALCOLATORI ELETTRONICI

Prova scritta

18 luglio 1994/1

Scrivere un programma in assembler che:

- Accetti in ingresso 10 parole (max 10 caratteri A...Z).
- Stampi le parole inserite in ordine alfabetico crescente.

In una seconda fase si considerino le parole formate utilizzando tutti i caratteri alfanumerici (A ... Z, 0 ... 9). I numeri seguono le lettere nell'ordinamento.

CALCOLATORI ELETTRONICI

Prova scritta

18 luglio 1994/2

Scrivere un programma in assembler che:

- Accetti in ingresso 10 numeri nel seguente formato $nnnn$Ekk con il seguente significato $nnnn * 10^{kk}$.
- Stampi i numeri inseriti in ordine crescente.

In una seconda fase si consideri la possibilità che il numero possa assumere i seguenti formati: $n.nnn$Ekk oppure $nn.nn$Ekk oppure $nnn.n$Ekk.

CALCOLATORI ELETTRONICI

Prova scritta

5 settembre 1994/1

Scrivere un programma in assembler che:

- Accetti in ingresso un numero di 4 cifre.
- Stampi tutti i numeri (di 4 cifre) diversi tra loro che è possibile formare con le cifre del primo numero. (I numeri vanno stampati in modo che compaiano in una sola schermata).
- Stampi i numeri in ordine crescente (questo punto può essere svolto in una fase successiva).

CALCOLATORI ELETTRONICI

Prova scritta

5 settembre 1994/2

Scrivere un programma in assembler che:

- Accetti in ingresso un numero di 5 cifre.
- Stampi tutti i numeri (di 2 cifre) diversi tra loro che è possibile formare con le cifre del primo numero. (I numeri vanno stampati in modo che compaiano in una sola schermata).
- Stampi i numeri in ordine crescente (questo punto può essere svolto in una fase successiva).

CALCOLATORI ELETTRONICI

Prova scritta

28 Ottobre 1994

Scrivere un programma in Assembly che:

1. Accetti in ingresso una stringa di massimo 70 caratteri;

   La stringa può essere formata solo dai seguenti caratteri: 0,1,2,...,9 e A,B,C,...,Z (numeri e lettere); non possono esserci due caratteri 0,1,2,...,9 adiacenti; la stringa non può terminare con un numero.

2. Conti le ricorrenze dei caratteri A,B,C,...,Z nella stringa; se un carattere viene preceduto da un numero *n*, si consideri come se tale carattere fosse ripetuto *n* volte consecutive.

3. Stampi il carattere (o i caratteri) ripetuto(i) più frequentemente.

In un primo tempo assumere che la stringa in ingresso soddisfi alle richieste del punto 1. Successivamente introdurre il controllo della corretta sintassi e generare errore se le specifiche del punto 1. non sono soddisfatte.

Al termine, ripetere il ciclo con una seconda stringa, ma evidenziare il carattere (o i caratteri) che hanno variato maggiormente la loro frequenza nelle due stringhe.

Esempio:
Sequenza in ingresso: ab4abbbm
Output: a

CALCOLATORI ELETTRONICI

Prova scritta

23 Gennaio 1995

Scrivere un programma in Assembly che:

1. Accetti in ingresso una stringa di massimo 70 caratteri;

   La stringa può essere formata solo dai seguenti caratteri: A,B,S,D, 2,...,9; non possono esserci due numeri adiacenti; la stringa non può terminare con un numero.

2. Cancelli lo schermo, e stampi un cursore (carattere a scelta) al centro dello schermo stesso;

3. Scandisca la stringa e sposti il cursore senza cancellare la posizione precedente secondo le seguenti regole:

   - Il carattere 'A' sposta in Alto il cursore di una posizione;
   - Il carattere 'B' sposta in Basso il cursore di una posizione;
   - Il carattere 'S' sposta a Sinistra il cursore di una posizione;
   - Il carattere 'D' sposta a Destra il cursore di una posizione;
   - Un numero agisce da prefisso di ripetizione per il carattere seguente.

In un primo tempo assumere che la stringa in ingresso soddisfi a tutte richieste e che il cursore non esca mai dallo schermo. Successivamente introdurre il controllo della corretta sintassi e generare errore se le specifiche non sono soddisfatte.

Esempio:
Sequenza in ingresso: as4a2sbbbs
Il cursore si deve muovere: asaaaassbbbs e deve generare un disegno del tipo:

```
 XXX
 X X
 X X
XX X
  XX
   O
```

CALCOLATORI ELETTRONICI

Prova scritta

12 Aprile 1995

Scrivere un programma in Assembly che:

1. Accetti in ingresso una stringa di massimo 80 caratteri;

   La stringa può essere formata solo dai seguenti caratteri: 0,1,2,...,9,+,–,= e deve rispettare le seguenti specifiche:

   - non possono comparire più di 3 numeri adiacenti;
   - i simboli +, –, = non possono essere adiacenti;
   - il simbolo = deve comparire una sola volta e al termine della stringa;

   Tale stringa forma così una espressione algebrica da valutare.

2. cancelli lo schermo, e stampi la stringa inserita;

3. stampi il risultato della espressione algebrica, assumendo che tutti i risultati parziali possano essere memorizzati in una parola da 16 bit.

In un primo tempo assumere che la stringa in ingresso soddisfi le specifiche sopra elencate. Successivamente introdurre il controllo della corretta sintassi e generare errore se le specifiche non sono soddisfatte.

Esempio:
Stringa in ingresso: 4+93–3–0+08=
Uscita: 4+93–3–0+08= 102

CALCOLATORI ELETTRONICI

Prova scritta

16 Giugno 1995

Scrivere un programma in Assembly che:

1. Accetti in ingresso una stringa di massimo 70 caratteri;

   La stringa può essere formata solo dai seguenti caratteri: numeri, lettere da ’A’ ad ’F’, e i seguenti simboli: ’+’, ’*’, ’ˆ ’. La stringa deve terminare con il segno ’=’.

2. calcoli il risultato della espressione logica assumendo che il simbolo ’+’ indichi l’operazione di OR, il simbolo ’*’ quella di AND, e il simbolo ’ˆ ’ quella di EXOR.

3. stampi la stringa inserita e il risultato.

In un primo tempo assumere che la stringa in ingresso abbia il seguente formato:

XXXXOXXXXOXXXX=

dove X indica una cifra esadecimale (range ’0’-’F’) e O indica una operazione logica (range: ’+’, ’*’, ’ˆ ’). Successivamente considerare la stringa formata sempre dalla concatenazione di due operazioni logiche, ma i cui operandi non siano limitati a 4 cifre.
Esempio:

Sequenza in ingresso: 1234+5678+9ABC=
Uscita: 1234+5678+9ABC=DEFC

CALCOLATORI ELETTRONICI

Prova scritta

25 Settembre 1995

Scrivere un programma in Assembly che:

1. Accetti in ingresso una stringa di massimo 60 cifre decimali;
2. Accetti in ingresso una seconda stringa di 4 cifre decimali;
3. Calcoli la funzione di correlazione tra le due stringhe, definita come: per ogni posizione della seconda stringa rispetto alla prima calcolare il numero di caratteri coincidenti.
4. stampi la stringa ottenuta (la cui lunghezza deve essere uguale a quella della stringa in ingresso).

In un primo tempo assumere che la seconda stringa sia formata da 4 caratteri; in seguito estendere il programma al caso in cui la seconda stringa abbia lunghezza variabile (tra 2 e 9 caratteri).
Esempio:

Prima stringa in ingresso: 123412431434
Seconda stringa in ingresso: 1234
Uscita: 400121103000

CALCOLATORI ELETTRONICI

Prova scritta

29 Novembre 1995

Scrivere un programma in Assembly che:

1. Accetti in ingresso due date (giorno/mese/anno) nel formato: `gg/mm/aa`, dove l'intervallo di variabilità è:
   `gg` varia da `01` a `30`
   `mm` varia da `01` a `12`
   `aa` varia da `50` a `99`

   Assumere che i dati inseriti soddisfino le specifiche elencate sopra.
2. calcoli e stampi il numero di giorni intercorsi tra le due date.

Assumere che la seconda data sia posteriore alla prima, che tutti i mesi siano formati da 30 giorni, che un anno sia formato da 30*12=360 giorni, e che non vi siano anni bisestili.

<u>Successivamente</u> considerare l'effettiva lunghezza dei mesi, e gli anni formati da 365 giorni.

Esempio:

```
Prima data:     01 12 94
Seconda data:  13 10 95

Giorni trascorsi:  312
```

CALCOLATORI ELETTRONICI

Prova scritta

19 Gennaio 1996

Scrivere un programma in Assembly che:

1. Accetti in ingresso due orari (ore/minuti/secondi) nel formato: `hh:mm:ss`, dove l'intervallo di variabilità per i minuti e i secondi è:
   `mm` varia da `00` a `59`
   `ss` varia da `00` a `59`
2. calcoli e stampi il numero di secondi intercorsi tra i due orari.

Determinare inoltre l'intervallo di variabilià delle ore (`hh`) in modo che tutti i calcoli possano essere effettuati utilizzando operazioni su 16 bit.

Assumere che il secondo orario sia posteriore al primo.

Successivamente considerare anche il caso in cui il primo orario sia posteriore al secondo (cioè si riferisca al giorno precedente).

Esempio:

```
Primo orario:      10:20:30
Secondo orario:    13:25:05

Secondi trascorsi: 11075
```

CALCOLATORI ELETTRONICI

Prova scritta

19 Gennaio 1996

Scrivere un programma in Assembly che continui ciclicamente a stampare i numeri da 0 a 100 fintanto che non vengono premute opportune sequenze di tasti. In un primo tempo considerare l'unica sequenza di tasti possibile: "STOP" che termina il programma.

Successivamente considerare anche la sequenza "PAUSA" che ferma il ciclo fintanto che non viene premuto un tasto qualsiasi.

Al fine di controllare lo stato della tastiera utilizzare i servizi Interrupt numero 16h (servizi 0 e 1).

CALCOLATORI ELETTRONICI

Prova scritta

27 Febbraio 1996

Scrivere un programma in Assembly che, introducendo un giorno e un mese dell'anno in corso (1996), determini il corrispondente giorno della settimana. Si ricorda che il 1 Gennaio 1996 è stato un lunedi.

Estendere in seguito il programma al calcolo del giorno della settimana di un qualunque giorno nel decennio 1990/1999. Si ricorda che il 1 Gennaio 1990 è stato un lunedi e che gli anni 1992 e 1996 sono anni bisestili.

Utilizzare un formato a piacere per l'introduzione dei dati e per la visualizzazione del risultato.

Esempio per il primo caso:

```
Giorno:   27
Mese:      2

Il giorno della settimana e':  Martedi
```

Esempio per il secondo caso:

```
Giorno:   27
Mese:      2
Anno:      1998

Il giorno della settimana e':  Venerdi
```

CALCOLATORI ELETTRONICI

Prova scritta

25 giugno 1996

Scrivere un programma in Assembly che accetti in ingresso una stringa di massimo 20 caratteri e che continui ciclicamente a stamparla, facendola ruotare sullo schermo (matrice 80×25).

Più precisamente la stringa deve essere stampata sullo schermo prima alla posizione (0, 0), poi alla (0, 1), poi alla (0, 2) fintanto che non si raggiunge la posizione (0, 80 – <lunghezza_della_stringa>); nella successiva stampa della stringa tutti i caratteri devono essere stampati consecutivamente, eccetto l'ultimo, che deve essere stampato sotto il penultimo.

```
+-----------------------------+
—abc                          —
—                             —
—                             —
—                             —
+-----------------------------+

+-----------------------------+
— abc                         —
—                             —
—                             —
—                             —
+-----------------------------+

+-----------------------------+
—  abc                        —
—                             —
—                             —
—                             —
+-----------------------------+
              ...
+-----------------------------+
—                           ab—
—                            c—
—                             —
—                             —
+-----------------------------+

+-----------------------------+
—                            a—
—                            b—
—                            c—
—                             —
+-----------------------------+
```

Il programma deve terminare dopo un giro completo della stringa.

<u>Successivamente</u> considerare la possibilità di uscire dal programma con la pressione del tasto ‘SPAZIO’ anche se la stringa non ha terminato il ciclo; al fine di controllare lo stato della tastiera utilizzare l’Interrupt numero 16h (servizi 0 e 1).

<u>Successivamente</u> consentire al programma di eseguire più di un ciclo: al termine di ogni ciclo la stringa verrà raddoppiata.

```
+------------------------------+
—abcabc                        —
—                              —
—                              —
—                              —
+------------------------------+
```

Esempio di risoluzione

```
dati        SEGMENT
coords      dw    ?
deltariga   db    ?
deltacol    db    ?
strlen      dw    16
stringa     db    ' PROVA˙DI˙ESAME '
dati        ENDS

pila        SEGMENT STACK 'STACK'
             DB 1000 DUP (?)
pila        ENDS


ASSUME   DS:dati,SS:pila,CS:codice

codice  SEGMENT PARA PUBLIC 'CODE'
main      proc far
          xor   ax, ax
          push ds
          push ax
          call init
          mov   cx, 80*2+20*2
scloop: mov   dx, [coords]
          mov   bh, [deltariga]
          mov   bl, [deltacol]
          mov   si, 0
lenloop:mov   al, stringa[si]
          call printchar
```

```
          call  inc˙coords
          inc   si
          cmp   si, [strlen]
          jnz  lenloop
          mov   [deltariga], bh
          mov   [deltacol], bl
          mov   dx, [coords]
          call  inc˙coords
          mov   [coords], dx
          call  ritardo
          loop  scloop
          ret
main      endp

ritardo proc near
          push  cx
          mov  cx, 30000
ciclo:   loop  ciclo
          pop   cx
ritardo endp

inc˙coords proc near
          add  dh, [deltariga]
          add  dl, [deltacol]
          cmp  dl, 80
          jnz  no1ang
          dec  dl
          inc  dh
          mov  [deltariga], 1
          mov  [deltacol], 0
          jmp  fine
no1ang:
          cmp  dh, 21
          jnz  no2ang
          dec  dl
          dec  dh
          mov  [deltariga], 0
          mov  [deltacol], 0FFh
          jmp  fine
no2ang:
          cmp  dl, 0FFh
          jnz  no3ang
          inc  dl
          dec  dh
          mov  [deltariga], 0FFh
          mov  [deltacol], 0
          jmp  fine
no3ang:
          cmp  dh, 0FFh
          jnz  fine
          inc  dl
          inc  dh
          mov  [deltariga], 0
```

```
          mov [deltacol], 1
fine:     ret
inc_coords endp

printchar proc near
          push bx
          push ax
          mov bh, 0
          mov ah, 2
          int 10h
          pop ax
          push ax
          mov ah, 0Eh
          mov bl, 0
          int 10h
          pop ax
          pop bx
          ret
printchar endp

init      proc near
          mov  ax, dati
          mov  ds, ax
          mov  [coords], 0
          mov  [deltariga], 0
          mov  [deltacol], 1
          ret
init      endp

codice    ENDS
          END main
```

CALCOLATORI ELETTRONICI

Prova scritta

16 luglio 1996

Scrivere un programma in Assembly che

- accetti in ingresso una stringa di massimo 50 caratteri;
- accetti in ingresso una seconda stringa numerica formata dallo stesso numero di caratteri della precedente.
- cancelli lo schermo e stampi la prima stringa in alto a sinistra (posizione 0,0).

Il programma deve poi spostare ogni lettera della stringa appena stampata in basso di un numero di posizioni dato dal numero della seconda stringa associato alla lettera della prima stringa.

Esempio:

```
Prima stringa:   ABCDE
Seconda stringa: 12143

Output:    ABCDE
           A C
            B
              E
             D
```

Successivamente il programma deve entrare in un ciclo in cui le lettere della prima stringa rimbalzano tra le due posizioni indicate precedentemente.

In un primo tempo supporre che i dati inseriti soddisfino alle specifiche; in un secondo tempo effettuare il controllo degli errori.

CALCOLATORI ELETTRONICI

Prova scritta

10 Settembre 1996

Scrivere un programma in Assembly che

- accetti in ingresso una stringa di 10 caratteri formata esclusivamente dai caratteri 'A', 'B', 'S', 'D';
- stampi la stringa al centro dello schermo;
- entri in un ciclo in cui ad ogni iterazione vengono mossi tutti i caratteri della stringa nel seguente modo:
  - i caratteri 'A' vengono spostati di una posizione verso l'Alto;
  - i caratteri 'B' vengono spostati di una posizione verso il Basso;
  - i caratteri 'S' vengono spostati di una posizione verso Sinistra;
  - i caratteri 'D' vengono spostati di una posizione verso Destra;

  Qualora la posizione in cui un carattere deve essere mosso risulti occupata, il carattere rimarrà fermo. In più, quando un carattere raggiunge il bordo dello schermo, si ferma.
- il ciclo termina quando tutti i caratteri sono fermi.

Esempio:

```
.........................
.........................
........ABSDAAS..........
.........................
.........................

.........................
........A...AA...........
.........S..DS...........
.........B...............
.........................

........A...AA...........
.........................
........S...DS...........
.........................
.........B...............

........A...AA...........
.........................
.......S....DS...........
.........................
.........B...............
```

........A...AA...........
....
......S.....DS...........
....
.........B...............

Ultima iterazione:
........A...AA...........
....
S..... .....DS...........
....
.........B...............

CALCOLATORI ELETTRONICI

Prova scritta

1 Ottobre 1996

Scrivere un programma in Assembly che:

- accetti in ingresso una stringa composta da almeno 10 caratteri e massimo 70; la stringa deve contenere 2 lettere maiuscole mentre le rimanenti devono essere minuscole.
- entri in un ciclo la cui funzione è di stampare la stringa inserita con le seguenti modifiche:
  - le due lettere maiuscole si spostano all'interno della stringa di una posizione;
  - nel caso la posizione successiva sia già occupata dall'altra lettera maiuscola o se si è arrivati al termine della stringa, la lettera modificherà la propria direzione.
- ad ogni iterazione il programma deve acquisire un carattere da tastiera: nel caso in cui il tasto sia "SPAZIO", il programma termina, altrimenti viene rieseguito il ciclo.

<u>Successivamente</u> introdurre il controllo su;;a correttezza del formato della stringa inserita dall'utente.

Esempio:

```
aBcdefGhijkl
abCdefgHijkl
abcDefghIjkl
abcdEfghiJkl
abcdeFghijKl
abcdefGhijkL
abcdefgHijKl
abcdefghIJkl
abcdefgHijKl
abcdefGhijkL
abcdeFghijKl
abcdEfghiJkl
abcDefghIjkl
abCdefgHijkl
```

CALCOLATORI ELETTRONICI

Prova scritta

19 Novembre 1996

Scrivere un programma in Assembly formato da un ciclo principale in cui un carattere viene mosso di una posizione per ogni iterazione. Il carattere partirà dall'angolo alto a sinistra dello schermo e si muoverà verso destra; una volta arrivato al bordo destro, scenderà di una posizione e continuerà a muoversi verso sinistra. Una volta arrivato al bordo sinistro, scenderà di una posizione e riprenderà secondo l'andamento descritto precedentemente. Quando il carattere raggiunge l'utlima posizione dello schermo (angolo in basso a destra), riprenderà dalla posizione iniziale.

Inserire all'interno del ciclo due procedure che permettano all'utente di variare la velocità di movimento del carattere attraverso la pressione di due diversi tasti. **Attenzione:** il carattere DEVE muoversi di una posizione ad ogni iterazione, indipendentemente dalla pressione dei tasti sopra menzionati!

Al fine di controllare lo stato della tastiera si consiglia di utilizzare l'Interrupt numero 16h (servizi 0 e 1).

Successivamente introdurre altre due procedure che permettano all'utente di

- riportare il carattere alla posizione iniziale.
- fermare il movimento del carattere fino alla pressione di un altro tasto;

CALCOLATORI ELETTRONICI

Prova scritta

28 Gennaio 1997

Scrivere un programma in Assembly che

- accetti in ingresso una stringa di massimo 80 caratteri;
- pulisca lo schermo;
- stampi la stringa inserita lettera per lettera a partire dal centro dello schermo spostandosi ad ogni carattere di una posizione verso destra; ogni volta che viene stampata la lettera 'b', la stampa della stringa prosegue verso il basso dello schermo; ogni volta che viene stampata la lettera 'a', la stampa della stringa prosegue verso l'alto dello schermo; ogni volta che viene stampata la lettera 'd', la stampa della stringa prosegue verso la destra dello schermo; ogni volta che viene stampata la lettera 's', la stampa della stringa prosegue verso la sinistra dello schermo.

Successivamente:

- introdurre il controllo che la stampa non esca dallo spazio dello schermo; in tal caso segnalare errore.
- modificare il programma in modo che venga stampato un asterisco nel caso in cui la posizione corrente sia già occupata da un altro carattere.

```
Inserire una stringa:  hujaikosusjhurbfuhedplpp

            bruhjsus
            f      o
            u      k
            h      i
            e   huja
            dplpp
```

CALCOLATORI ELETTRONICI

Prova scritta

28 Febbraio 1997

Scrivere un programma in Assembly che:

- accetti in ingresso una stringa composta da massimo 70 caratteri; la stringa può contenere solo lettere maiuscole e minuscole;
- entri in un ciclo in cui ad ogni iterazioine venga stampata una nuova stringa ottenuta dalla precedente eliminando le lettere (anche multiple) con codice ASCII minore.
- il ciclo termini quando la stringa risulta vuota.

<u>Successivamente</u> modificare il programma in modo che ad ogni iterazione vengano stampate due stringhe: la prima secondo quanto scritto sopra, mentre la seconda contenente le lettere eliminate dalla prima.

Esempio:

```
Inserire una stringa:   FFAFcAAMFFFEEEACf
iterazione 1:            FFFcMFFFEEECf        AAAA
iterazione 2:            FFFMFFFEEEf          AAAAcC
iterazione 3:            FFFMFFFf             AAAAcCEEE
iterazione 4:            M                    AAAAcCEEEFFFFFFf
iterazione 5:                                 AAAAcCEEEFFFFFFfM
```

CALCOLATORI ELETTRONICI

Prova scritta

2 Aprile 1997

Scrivere un programma in Assembly che:

- accetti in ingresso una stringa di massimo 70 caratteri composta da lettere maiuscole, minuscole e spazi; lo spazio, che puø' anche comparire più volte consecutivamente, è carattere separatore tra parole diverse;
- accetti in ingresso una lettera maiuscola;
- stampi tutte le parole della prima stringa che contengono la lettera di cintrollo specificata dalla seconda immissione su righe diverse consecutivamente, in modo da allineare sulla stessa colonna (centrata nello schermo) la lettera di controllo.

Nel caso si avessero più ricorrenzze della lettera di controllo in una stessa parola, tale parole viene considerata e stampata più volte.

<u>Successivamente</u> eliminare l'inserimento della lettera di controllo e creare 5 colonne, una per ogni vocale.

Esempio:

```
Inserire una stringa: Compito di Calcolatori Elettronici
Inserire una lettera: T

                                 *
                            Compito
                          Calcolatori
                              Elettronici
                             Elettronici

Inserire una stringa: Compito di Calcolatori Elettronici

       *               *                        *             *
      Calcolatori      Elettronici          Compito          Compito
 Calcolatori         Elettronici               di       Compito
                                      Calcolatori         Calcolatori
                                        Elettronici   Calcolatori
                                      Elettronici       Elettronici
```

CALCOLATORI ELETTRONICI

Prova scritta

7 Luglio 1997

Scrivere un programma in Assembly che:

- accetti in ingresso 10 numeri n_i, $i = 1 \ldots 10$, compresi tra 0 e 1000;
- rappresenti i numeri inseriti sotto forma di istogramma (a tal fine si scelga un modo per la loro approssimazione).

In un primo tempo assumere che il range di variabilità dell'istogramma sia tra 0 e 1000; successivamente determinare il range di variabilità utilizzando il valore massimo assunto dai dieci numeri, cioè tra 0 e $\max(n_i)$.

Esempio:

```
Inserire 10 numeri: 100, 200, 300, 100, 55, 45, 540, 230, 0, 200

                  XXX
                  XXX
                  XXX
                  XXX
                  XXX
      XXX          XXX
      XXX          XXX XXX
   XXX XXX          XXX XXX    XXX
   XXX XXX          XXX XXX    XXX
XXX XXX XXX XXX XXX    XXX XXX    XXX
XXX XXX XXX XXX XXX XXX XXX XXX    XXX
```

CALCOLATORI ELETTRONICI

Prova scritta

10 Settembre 1997

Scrivere un programma in Assembly che:

- accetti in ingresso 2 stringhe alfanumeriche (che comprendono solo le lettere dell'alfabeto maiuscole e minuscole, e i numeri) senza spazi e con almeno un carattere in comune;
- stampi la prima stringa sullo schermo in direzione orizzontale da sinistra a destra;
- stampi la seconda stringa verticalmente dall'alto al basso in modo che le due stringhe si incrocino in corrispondenza del primo carattere che hanno in comune, che verrà sostituito da un asterisco.

In un primo tempo assumere che i dati in ingresso siano conformi alle specifiche. Successivamente introdurre il controllo sugli errori di inserimento.

Esempio:

```
Prima  stringa:    Albero
Seconda stringa:   REGALO

Output:
                      R
                   Alb*ro
                      G
                      A
                      L
                      O
```

CALCOLATORI ELETTRONICI

Prova scritta

12 Settembre 1997

Scrivere un programma in Assembly che permetta di inserire una stringa di lettere minuscole, maiuscole e numeri, di massimo 40 caratteri. Si supponga che la stringa contenga almeno 2 vocali. La prima e l'ultima vocale dividono la stringa in 3 parti.

Il programma deve pulire lo schermo e stampare la stringa iniziale in posizione centrale. Successivamente deve stampare su righe successive le tre stringhe in modo che la seconda stringa (la parte centrale della stringa iniziale) rimanga nella posizione originale, mentre le prima e la terza si spostino di un carattere verso sinistra e verso destra rispettivamente fino a raggiungere il bordo dello schermo. Quando entrambe le stringhe mobili hanno raggiunto il bordo dello schermo il programma termina.

<u>Successivamente</u>

- introdurre i controlli sulla corretteza della stringa in ingresso e
- fare in modo che l'insieme delle lettere che dividono la stringa iniziale sia definibile da utente.

Esempio:

```
Inserire la stringa:                zxcvb3abcdefghijklmn
Inserire le lettere per dividerla:  aeidfou

+--------------------------------+
|       zxcvb3abcdefghijklmn     |
|      zxcvb3 abcdefghi jklmn    |
|     zxcvb3  abcdefghi  jklmn   |
|    zxcvb3   abcdefghi   jklmn  |
|   zxcvb3    abcdefghi    jklmn |
|  zxcvb3     abcdefghi     jklmn|
| zxcvb3      abcdefghi     jklmn|
|zxcvb3       abcdefghi     jklmn|
+--------------------------------+
```

CALCOLATORI ELETTRONICI

Prova scritta

2 Dicembre 1997

Scrivere un programma in Assembly che permetta di inserire una stringa formata solo da lettere maiuscole e minuscole e che la visualizzi sullo schermo. Il programma deve poi entrare in un ciclo da cui può uscire solo alla pressione del tasto 'Spazio'. All'interno del ciclo viene controllata la tastiera; ogni volta che viene premuto il tasto corrispondente ad una lettera, dalla stringa iniziale vengono eliminate tutte le ricorrenze (maiuscole e minuscole) della lettera digitata; la nuova stringa viene sovrascritta alla precedente (alle stesse coordinate della precedente), eliminando gli eventuali caratteri rimasti dalla stringa precedente.

Successivamente modificare il programma in modo che la pressione di tasti corrispondenti a lettere minuscole abbiano l'effetto di eliminare le corrispondenti ricorrenze nella stringa iniziale (come sopra), mentre la pressione di tasti corrispondenti a lettere maiuscole abbiano l'effetto opposto, ovvero di reinserire le lettere eliminate in precedenza.

Esempio:

```
Inserire una stringa:   abcAAbCCaBBc

abcAAbCCaBBcdE

(dopo la pressione di 'a')
bcbCCBBcdE

(dopo la pressione di 'b')
cCCcdE

(dopo la pressione di 'c')
dE

(dopo la pressione di 'B')
bbBBdE

(dopo la pressione di 'A')
abAAbaBBdE

(dopo la pressione di 'C')
abcAAbCCaBBcdE
```

CALCOLATORI ELETTRONICI

Prova scritta

27 Gennaio 1998

Scrivere un programma in Assembly che permetta di inserire tre stringhe a, b, e c di caratteri alfanumerici (lettere e numeri) e le visualizzi sullo schermo. Il programma deve stampare la stringa a, carattere per carattere, partendo dalla metà del bordo sinistro dello schermo verso destra; ogni volta che si incontra una ricorrenza di un carattere della stringa b, la stampa procede verso l'alto di un carattere. Parallelamente ogni volta che si incontra una ricorrenza di un carattere della stringa c, la stampa procede verso il basso di un carattere.

<u>Successivamente</u> modificare il programma in modo che richieda l'immissione di una quarta stringa d; ogni volta che si incontra una ricorrenza di un carattere di quest'ultima stringa, la stampa della stringa a viene riportata nella posizione originale.

In un primo tempo si assuma che gli insiemi dei caratteri delle stringhe b, c e d abbiano intersezione nulla; <u>successivamente</u> introdurre tale controllo e, in caso di intersezione non nulla, mostrare le lettere in comune.

Esempio:

```
Inserire la prima stringa:      zxcvb3aqcdefghijklmn
Inserire la seconda stringa:    vb
Inserire la terza stringa:      hl
Inserire la quarta stringa:     f

     3aqcdef
    b
zxcv       gh
             ijkl
                 mn
```