ANNO III   NUMERO 8

```
###     ### ###   #####   ########     #####
##### ##### ### ###   ### ###    ### ###   ###
### ### ### ### ###       ###    ### ###   ###
###     ### ### ###       ########   ###  design
###     ### ### ###       ### ###    ###   ###
###     ### ### ###   ### ###  ##### ###   ###
###     ### ###   #####   ###   #####  #####
```

BOLLETTINO DEL CLUB UTENTI MICRO DESIGN

AGOSTO 1983

PARLIAMO ANCORA DI LINGUAGGI

Cari amici, in seguito alla pubblicazione dell'elenco dei linguaggi che avevamo adattato al nostro micro (bollettino di APRILE 1983) siamo stati subissati da richieste di spiegazioni e di maggiori informazioni.

Ci siamo cosi' resi conto che una ragguardevole frazione dei soci del club non e' ancora molto familiare con la terminologia corrente nel mondo del software e dura una certa fatica a distinguere fra termini cosi' poco consueti.

In questo bollettino pertanto cercheremo di chiarire la terminologia relativa al tipo dei linguaggi, scusandoci con i piu' esperti che troveranno questo esercizio di nomenclatura piuttosto noioso.

Siamo pero' certi che anche costoro apprezzeranno il prossimo bollettino che illustrera' piu' diffusamente le caratteristiche dei linguaggi da noi adattati.

Cominceremo dai linguaggi piu' vicini al linguaggio del micro per spostarci poi a quelli piu' vicini al linguaggio umano, il nome del linguaggio sara' trascritto in italiano nei titoli e portera' vicino, fra parentesi, l'equivalente inglese.

ASSEMBLATORE (ASSEMBLER)

Esempio ASM.COM

Viene cosi' definito un linguaggio che traduce le istruzioni da un codice mnemonico (facile da ricordare) al codice macchina, con una corrispondenza uno a uno, cioe' ad ogni istruzione in codice mnemonico corrisponde una sola istruzione in linguaggio macchina.

Poiche' le istruzioni in linguaggio macchina (il binario usato dal micro) sono semplici, ne consegue che anche le istruzioni dell'assembler sono semplici e poco potenti, oltre che di difficile uso. Questa ultima caratteristica e' dovuta al fatto che una qualsiasi operazione, anche in apparenza semplice come una moltiplicazione, dovra' essere scomposta in una sequenza di operazioni elementari corrispondenti alle istruzioni del linguaggio macchina, per poter essere programmata in assembler.

Il vantaggio del linguaggio assembler rispetto alla scrittura in linguaggio macchina consiste nel fatto che e' piu' facile ricordare un codice di lettere che richiama alla memoria una operazione (es. ADD per l'operazione di addizione) che un codice binario od esadecimale, come le istruzioni macchina.

Il vantaggio dell'assemblatore rispetto ai linguaggi piu' potenti consiste nella possibilita' di effettuare tutte le operazioni possibili (anche se talvolta occorre scomporle in una lunga sequenza di operazioni elementari). Con i linguaggi piu' potenti, invece, alcune operazioni non sono realizzabili se non sono state previste da chi ha progettato il linguaggio (provate a leggere una porta di ingresso con il BASIC !!).

Un linguaggio assemblatore di solito non interpreta o traduce le istruzioni man mano che gli vengono passate dall'operatore, ma prevede che l'insieme delle istruzioni che compongono il programma sia gia' stato memorizzato in un opportuno tipo di supporto (disco, nastro perforato, nastro magnetico). Esso effettua la traduzione delle istruzioni e rimemorizza il programma, ora in linguaggio macchina, su un altro o sullo stesso supporto. Alla fine delle operazioni si ottiene un programma che potra' essere direttamente eseguito e non necessita di alcun altro ausilio.

Quanto sopra significa che assemblatore e programma assemblato non occupano mai contemporaneamente la memoria. Infatti mentre lavora l'assemblatore il programma viene memorizzato sul supporto man mano che viene tradotto e non occupa pertanto spazio di memoria. D'altra parte il programma tradotto puo' essere successivamente caricato in memoria **al posto** dell'assemblatore, perche' non ha bisogno di quest'ultimo per lavorare.

MACRO-ASSEMBLATORE (MACRO-ASSEMBLER)

Es: MAC.COM, ACT.COM

E' un lnguaggio che consente, oltre alle normali funzioni, di definire delle MACROISTRUZIONI, cioe' un insieme di istruzioni semplici che concorrono a formare una istruzione piu' complessa (detta appunto macroistruzione), che viene identificata con un nome definito dal programmatore.

Ogni volta che all'interno del programma necessita usare l'istruzione complessa sara' sufficiente scrivere il nome della macroistruzione ed il macro assemblatore provvedera' ad inserire al posto di questa l'insieme di istruzioni semplici che la compongono.

Le macroistruzioni contenute in un programma devono essere comunque sempre fornite al macroassemblatore insieme con il programma (cioe' ogni tipo di macroistruzione deve essere definita mediante la definizione dell'insieme di istruzioni che la compongono e l'assegnazione di un nome).

Il vantaggio di questo tipo di linguaggio rispetto ad un normale assemblatore e' che il programmatore puo' crearsi una volta per tutte un insieme di potenti istruzioni, scelte fra le operazioni che usa piu' di sovente, e scrivere i propri programmi utilizzando queste potenti istruzioni, avendo cura solo di far precedere il programma dalla definizione di queste macroistruzioni. In altre parole, con un po' di pazienza e di esperienza, il programmatore puo' in effetti crearsi un linguaggio molto potente.

ASSEMBLATORE IN RILOCABILE (RELOCABLE ASSEMBLER)

Es: M80.COM

I programmi fatti per un normale aasemblatore contengono oltre alle istruzioni, anche l'indirizzo della cella di memoria a partire dalla quale il programma dovra' essere caricato e non sara' possibile far girare il programma se questo sara' stato caricato a pertire da una locazione diversa.

Negli assemblatori in rilocabile e' invece possibile definire se un programma, od un segmento di programma, devono partire da una locazione di memoria determinata (come negli assemblatori normali) oppure se la locazione di partenza potra' essere fornita successivamente. In questo ultimo caso il segmento od il programma vengono detti "rilocabili".

Questa opzione e' molto comoda perche' consente di preparare una "LIBRERIA" di programmi rilocabili, gia' tradotti, che eseguono funzioni particolari, ad esempio la lettura di un file da cassetta, la stampa di un file, ecc., e di poterli di volta in volta "cucire assieme", mescolandoli come piu' torna comodo e facendoli partire dalla locazione di memoria piu' opportuna, formando programmi complessi che non dovranno ogni volta essere ripassati attraverso l'assemblatore.

CONCATENATORE (LINKER)

Es: L80.COM

E' il programma che serve a "cucire" fra loro i programmi rilocabili prodotti da un assemblatore-rilocatore. Riceve dall'operatore una lista dei programmi che devono essere cuciti insieme, l'indirizzo di partenza del programma che ne dovra' risultare, eventuali indirizzi di partenza di segmenti di programma (se diversi da quelli che risulterebbero dalla messa in sequenza dei segmenti), l'ordine di messa in sequenza dei vari programmi o segmenti di programma che dovranno costituire il programma finale.

Il programma concatenatore si occupa di ricalcolare tutti gli indirizzi dei salti e delle chiamate di subroutines, cambiati in seguito alla assegnazione di nuovi indirizzi di partenza, di calcolare gli indirizzi dei salti e chiamate fatti da un programma a istruzioni e subroutines di un altro programma ed in genere di mettere a posto tutte le cose in modo che la massa eterogenea di programmi fornitigli si comporti come un unco programma ben coordinato.

Alcuni concatenatori, appartenenti ad una famiglia di linguaggi prodotti da una unica ditta, sono in grado di cucire fra loro i programmi generati sia dai compilatori (vedi seguito) che dagli assemblatori appartenenti alla famiglia. In tale modo il programma risultante puo' essere generato tramite il compilatore, per le parti relative ai calcoli e alla presentazione dei dati su video o stampante, e tramite l'assembaltore per le parti relative alla gestione diretta delle porte di ingresso/uscita o di periferiche particolari, in modo da sfruttare al meglio le caratteristiche dei singoli linguaggi.

COMPILATORE (COMPILER, TRUE COMPILER, FULL COMPILER)

Es: BASCOM.COM, F80.COM,

Ha un funzionamento simile a quello dell'assemblatore, ma sfrutta istruzioni piu' potenti e complesse, che vengono tradotte come un insieme di istruzioni macchina, perdendo quindi la corrispondenza uno a uno dell'assemblatore. E' un linguaggio piu' vicino all'uomo che alla macchina, nel senso che le sue istruzioni sono simili a quelle con cui siamo usi a trattare (moltiplicazione, divisione, se questo... e' vero, allora..., altrimenti..., ecc.). E' particolarmente adatto a programmi in cui ci siano da eseguire calcoli, da prendere decisioni, da gestire formati complessi nella accettazione e nella presentazione dei dati. Gestisce facilmente le operazioni in decimale (invece che in binario come l'assemblatore).

Come l'assemblatore, lavora su programmi registrati in precedenza su un supporto (in questo caso quasi esclusivamente il disco) e genera il programma tradotto, ancora sul supporto e non in memoria. Questo significa che il programma prodotto non avra' bisogno della presenza in memoria del compilatore per poter girare, esattamente come per l'assemblatore.

A differenza degli assemblatori, i compilatori occupano molta memoria (olre 24 K nella maggior parte dei casi) ed inoltre i programmi da loro prodotti sono tanto piu' efficienti, in termini di limitatezza di occupazione di memoria e di velocita' di esecuzione, quanto maggiore e' la quantita' di memoria a disposizione del compilatore.

Sul mercato si trovano oggi diversi tipi di compilatori, ognuno dei quali adattato meglio a certi tipi di calcoli; i piu' comuni tipi di compilatori sono:

BASIC - Di uso generale, facile da capire ed usare
FORTRAN - Orientato al calcolo scientifico
COBOL - Orientato alle applicazioni commerciali
PASCAL - Orientato alle applicazioni scientifiche, ma di uso abbastanza generale

INTERPRETE (INTERPRETER)

Es: BASIC.COM

Ha le stesse caratteristiche del compilatore, salvo il fatto che le istruzioni gli possono essere fornite sia da un supporto, che immediatamente dall'operatore tramite la console. A differenza del compilatore, pero', le istruzioni non vengono tradotte e rimemorizzate sul supporto in forma direttamente eseguibile, ma vengono interpretate ed eseguite sul momento.

Quanto sopra significa che in memoria devono sempre risiedere **contemporaneamente** il programma interprete ed il programma che deve essere eseguito, con conseguente notevole occupazione di memoria e limitazione alle dimensioni del programma utente, soprattutto in presenza di interpreti di grosse dimensioni.

Un altro difetto degli interpreti e' la limitata velocita', dovuta al fatto che ogni istruzione deve essere interpretata al momento e quindi il tempo necessario a questa operazione di interpretazione si somma al tempo di esecuzione dell'istruzione stessa.

I vantaggi degli interpreti, rispetto ai compilatori riesedono nella possibilita' di provare con estrema facilita' il programma prodotto, poiche' di solito gli interpreti sono dotati di potenti mezzi di test, che lavorano direttamente sulle istruzioni del linguaggio e non hanno bisogno di fare riferimento alle istruzioni macchina equivalenti. Inoltre la loro semplicita' di struttura li rende piu' facilmente realizzabili e quindi piu' facilmente reperibili sul mercato.

SEMICOMPILATORE (SEMICOMPILER, COMPILER)

Es: CB80.COM

E' una via di mezzo tra un vero compilatore ed un intrerprete. Come un compilatore lavora su programmi memorizzati su un supporto tramite un programma editore (vedi seguito), elabora le istruzioni e produce un programma che viene a sua volta memorizzato sul supporto. Il programma prodotto non e' pero' scritto in linguaggio macchina, ma in un linguaggio che e' una via intermedia tra quello umano e quello macchina, detto "meta-codice".

In fase di esecuzione il meta-codice viene interpretato da un interprete semplificato e di ridotte dimensioni detto "supporto run-time", che deve essere presente in memoria contemporaneamente al meta-codice.

I semicompilatori sfruttano in parte i vantaggi dei compilatori e quelli degli interpreti, infatti sono di piu' facile realizzazione dei compilatori, ma possono girare su ridotte quantita' di memoria, visto che il meta-codice e' molto piu' compatto del programma originario e che il supporto run-time e' di solito molto piu' piccolo dell'interprete (un semicompilatore da 24 K ha di solito un supporto run-time di non piu' di 12 K).

Con i semicompilatori abbiamo esaurito i tipi piu' comuni di linguaggi, e passiamo ora ad esaminare un'altra grossa classe di programmi, detta in italiano **PROGRAMMI DI UTILITA'** ed in inglese **UTILITIES.**

EDITORE (EDITOR)

Es: ED.COM, WM.COM

E' un programma che consente di scrivere con facilita' i programmi. Esso infatti riceve i caratteri dalla tastiera di console e li presenta sul video, ma contemporaneamente li immagazzina in memoria, in un'area detta spazio di lavoro, o "workspace".

La caratteristica principale dell'editore e' pero' quella di facilitare le correzioni. Per fare questo esso riconosce un certo numero di caratteri speciali, di solito ottenuti premendo contemporaneamente il tasto "CONTROL" ed un carattere. Questi caratteri speciali consentono di spostare il puntatore avanti ed indietro lungo una riga, alla riga precedente o a quella successiva, all'inizio o alla fine del testo. Altri caratteri consentono di cancellare un certo numero di lettere del testo prima o dopo il puntatore, di inserirne altre in mezzo, di sostituire parole o frasi in tutto il testo con altre piu' appropriate.

Al termine delle operazioni il programma editore puo' memorizzare il testo su di un supporto (disco, nastro), da cui esso potra' poi essere riletto per essere utilizzato, ad esempio da un programma assemblatore, oppure per essere successivamente rimodificato dal programma editore.

Gli editori sono in prevalenza utilizzati dai compilatori e dagli assemblatori, mentre di solito gli interpreti contengono gia' al proprio interno alcune funzioni degli editori.

ELABORATORE DI TESTI (WORD PROCESSOR)

Es: WS.COM, POW.COM

E' un programma editore molto evoluto, aggiunge alle normali funzioni dell'editore quelle dedicate alla scrittura di testi, quali l'impaginazione automatica (allineamento delle parole al margine destro e a quello sinistro, definizione della lunghezza delle righe, posizionamento dei margini destro, sinistro, superiore ed inferiore, selezione del numero di righe a pagina, ecc.) la numerazione automatica delle pagine, la centratura dei titoli.

Oltre alle funzioni viste sopra di solito accetta anche delle direttive di stampa, quali la passata multipla di stampa di certe frasi (grassetto) o la sottolineatura di certe altre, l'inserzione di intestazioni in cima od in fondo alla pagina, ecc.

Gli elaboratori di testi piu' evoluti consentono inoltre funzioni molto potenti quali la cancellazione o lo spostamento di blocchi di testo, la ricopiatura di parti di testo in posizioni diverse dell'elaborato, la scrittura e lettura di blocchi di testo su e da disco.

Alcuni programmi, piu' semplici, detti post-elaboratori di testo, sfruttano un testo scritto tramite un normale programma editore, in cui sono inseriti dei caratteri speciali che il postelaboratore interpreta e fa eseguire in fase di stampa (ad esempio la centratura dei titoli, la numerazione di pagina, la conversione automatica di caratteri in maiuscolo o minuscolo, ecc.)

I normali word-processors invece fungono sia da editori che da elaboratori ed hanno il vantaggio di mostrare gia' sul video il risultato finale dell'elaborazione, prima della stampa.

ANALISTA (DEBUGGER)
Es: DDT.COM, ZDT.COM

E' un programma utilizzato per controllare il corretto funzionamento di altri programmi, ed eventualmente per correggerne gli errori.

Lavora su programmi in linguaggio macchina e quindi e' adatto a funzionare con programmi prodotti da assemblatori e, con un po' piu' di difficolta', da compilatori.

Fra le funzioni principali di uno "analista" ci sono:
- Il caricamento di programmi da disco
- La visualizzazione di aree di memoria
- La visualizzazione e sostituzione del contenuto di celle di memoria
- La visualizzazione e modifica dei registri interni del micro
- Il comando di partenza di un programma, con l'inserzione di "BREAKPOINT", cioe' indirizzi in cui l'esecuzione del programma si deve fermare per consentire all'operatore di andare ad esaminare come si sono modificati la memoria od i registri interni del micro.
- La "tracciatura" (trace) di un programma, cioe' la sua esecuzione istruzione per istruzione, con la visualizzazione del contenuto dei registri del micro dopo l'esecuzione dell'istruzione.

Il motivo per cui gli analisti lavorano male sui programmi prodotti dai compilatori e' dovuta al fatto che nei compilatori si perde la corrispondenza uno a uno fra le istruzioni del linguaggio e le istruzioni macchina e l'operatore solitamente non sa in quali istruzioni macchina e' stata tradotta l'istruzione fornita al compilatore.

Molto spesso gli analisti forniscono una limitata funzione di assemblaggio e disassemblaggio (vedi seguito) immediato delle istruzioni.

ANALISTA SIMBOLICO (SYMBOLIC DEBUGGER)
Es: ZSID.COM

E' un programma che presenta in piu', rispetto ad un normale debugger, l'opzione di poter trattare dati ed indirizzi non in forma esadecimale, ma con il simbolo che era stato utilizzato nel programma in assembler.

Ad esempio se nel programma era stato utilizzato il simbolo INIZIO per indicare l'indirizzo di partenza di una routine, si potra' dare il comando di partenza del programma da INIZIO, senza curarsi dell'effettivo indirizzo di partenza della routine.

I debuggers simbolici sono in genere dotati dell'assemblatore e disassemblatore rudimentali visti in precedenza.

DISASSEMBLATORE (DISASSEMBLER)
Es: DISINTEL.COM, DISZILOG.COM

E' un programma che consente di riottenere il programma scritto in assembler da uno scritto in linguaggio macchina e caricato in memoria.

E' un programma di difficile utilizzo perche' tende a decodificare come istruzioni anche eventuali aree di dati presenti in memoria fra una parte del programma e la successiva. Ha quindi una certa facilita' a perdere il passo, cioe' a interpretare come codice operativo l'operando di una istruzione e viceversa, a causa di eventuali tabelle di dati interposte fra le istruzioni del programma.

L'utilizzo di un disassemblatore deve essere fatto pertanto da esperti che siano in grado di riconoscere eventuali incongruenze nel codice disassemblato e segnalare al disassemblatore la presenza di probabili dati fra le istruzioni.

Per questo motivo i disasseblatori sono quasi del tutto inutili per ricostruire il codice prodotto da un compilatore, che mescola dati ed istruzioni con una certa frequenza ed inoltre usa istruzioni che utilizzano molto lo STACK e le manipola con una logica molto adatta alla traduzione automatica del codice, ma molto diversa dalla logica umana.

Concludiamo questa chiacchierata scusandoci con coloro che abbiamo annoiato e rimandandovi al prossimo bollettino per una spiegazione piu' dettagliata delle caratteristiche principali dei linguaggi elencati nel bollettino di APRILE 1983.

TASTO DI "CAPITAL LOCK" SULLA TASTIERA LX 387

Abbiamo ricevuto dal Sig. **Giovanni PORCELLI** di LATINA una interessante modifica da apportare alla scheda di tastiera LX 387 per realizzare la funzione di "CAPITAL LOCK" ad hardware, non piu' a software come illustrato nel manuale della scheda video MICRO design CVP 001. Questa soluzione costa solo il prezzo di un deviatore ed un minimo di lavoro iniziale e gode del notevole vantaggio di poter inserire o disinserire l'opzione in qualsiasi momento, durante qualsiasi programma.

Il Sig. Porcelli ci ha inviato un piacevole articoletto e delle ottime fotografie a colori. Purtroppo le foto a colori non hanno sufficiente contrasto per poter essere riprodotte con il nostro sistema di stampa ed inoltre ragioni di spazio ci costringono a riassumere il contenuto dell'articolo. Ci scusiamo di questo con il Sig. Porcelli e lo ringraziamo per la gradita collaborazione.

Ecco la modifica: Tagliare con una lama 2 o 3 mm di pista nei pressi del piedino 10 dell'integrato AY-5-2376 o KR 2376 (la pista collega il piedino 10 al piedino 18 del connettore D).

Preparare circa 25 cm di piattina a 3 fili colorati.

Saldare i tre capi rispettivamente al piedino 8 del 2376 (che attualmente risulta libero), sulla pista interrotta (lato piedino 10) e sempre sulla pista (lato piedino 18 del conn. D).

Gli estremi opposti della piattina andranno saldati ad un piccolo deviatore a pulsante od a levetta, avendo cura che il filo proveniente dal piedino 18 del conn. D sia connesso al centrale del deviatore.

Con un po' piu' lavoro, riuscendo a reperire un tasto uguale a quelli della tastiera, ma del tipo a deviatore, sara' possibile montare il CAPITAL LOCK vicino al tasto di SHIFT LOCK.

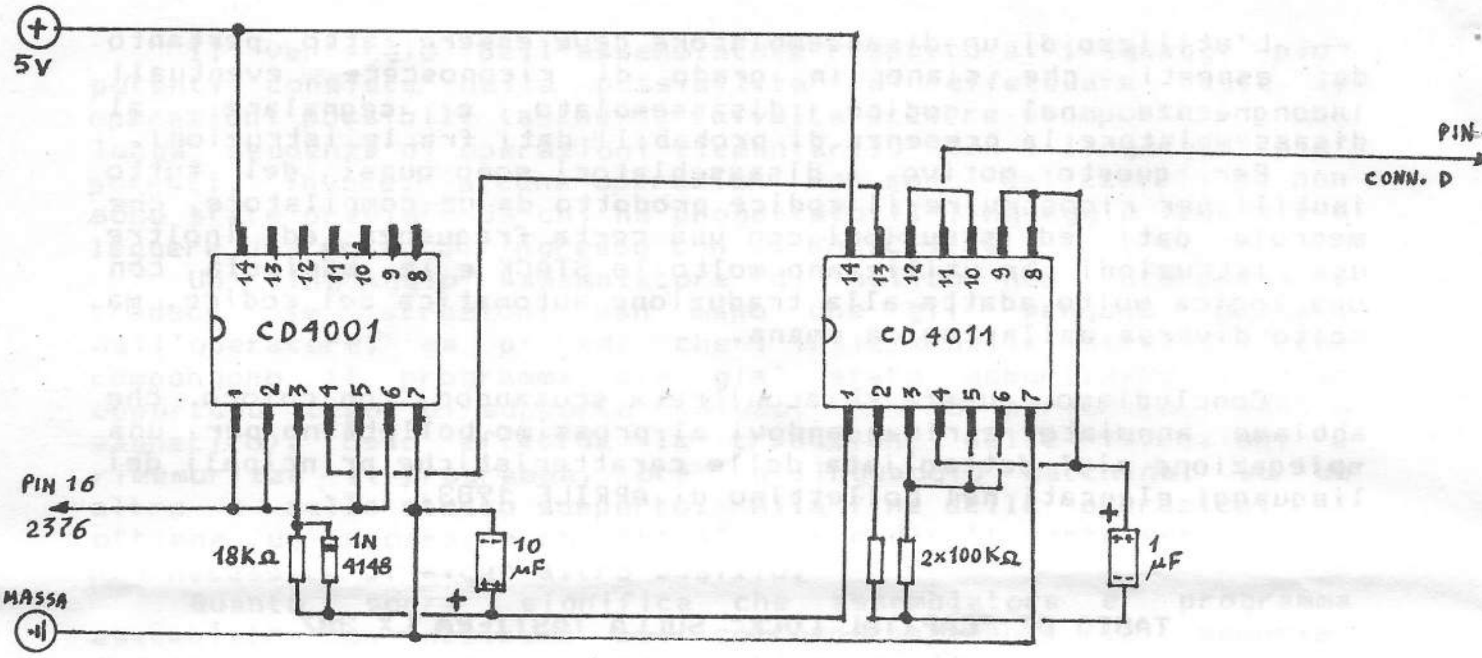
AUTOREPEAT

Il Sig. PORCELLI ci ha anche gentilmente inviato una modifica dei valori dei componenti del circuito dell'autorepeat, pubblicato nel numero di APRILE 1983 ed inviato dal Sig. RISOLDI. Questi nuovi valori, secondo il parere del Sig. Porcelli, rendono il funzionamento piu' pronto. L'elenco e' accompagnato da uno schema pratico della modifica, che pubblichiamo nell'ultima pagina.

Ecco le modifiche di valore dei componenti:

Il condensatore da	0,047 uF	diventa	1 uF - 10 V
" "	15 uF	"	10 uF - 10 V
La resistenza da	100 Kohm	"	18 Kohm

(in parallelo al diodo 1N4148)



MITTENTE

MICRO design
Via Rostan 1
16155 Genova

S T A M P E

```
**********************************************************************
*        Tagliando di iscrizione al club utenti                      *
*                                                                    *
* Cognome...............................Nome.........................*
*                                                                    *
* Via................................................................*
*                                                                    *
* C.A.P.................Citta'.......................................*
*                                                                    *
* Provincia...........................Telefono....../................*
*                                                                    *
* Configurazione micro...............................................*
*                                                                    *
* ...................................................................*
**********************************************************************
```