PREMESSA

A coronamento del “corso” sulla programmazione con linguaggio Assembler dei microprocessori serie ST6, non poteva mancare la raccolta in un unico cd-rom di tutti gli articoli pubblicati nel corso degli anni sull’argomento.

E quando diciamo tutti, intendiamo proprio tutti: dai due programmatori in kit, ai circuiti di prova, dalla spiegazione teorica delle istruzioni del linguaggio Assembler, alla loro applicazione pratica in elettronica, dagli accorgimenti per utilizzare al meglio le istruzioni e la memoria dei micro, all’uso del software emulatore per testare i programmi.

L’intento didattico accompagna tutti gli articoli, anche quelli che, a prima vista, sono di carattere più propriamente pratico: gli stessi programmi-sorgente, che trovate in questo stesso cd-rom in una directory dedicata, servono soprattutto per capire come si deve scrivere un’istruzione per ottenere una determinata funzione.
Leggendo i commenti accanto ad ogni riga di programma, non solo vi impadronirete della materia, ma potrete addirittura utilizzare blocchi di istruzioni trasferendoli nei vostri programmi.

Inoltre, con i software emulatori che vi proponiamo diventa facilissimo controllare le istruzioni via via che vengono eseguite. E’ così possibile capire dove e perché si genera l’errore e come fare per correggerlo.
Per questo motivo, ci siamo premurati di mettere a vostra disposizione, sempre in questo cd-rom, l’ultima versione del software emulatore SimST62, che avete imparato a conoscere, ad usare e ad apprezzare nei nostri articoli.

In appendice trovate il kit di una lampada ad ultravioletti per cancellare i microprocessori con memoria Eprom e un inedito sulla funzione Timer dei microprocessori ST6, che tiene conto del fatto che in alcuni tipi di micro è possibile attivare alcune modalità di funzionamento particolari e molto interessanti.

Non poteva mancare l’indice analitico dei kit e degli argomenti teorici, che vi rimanda immediatamente agli articoli in cui l’argomento scelto è trattato.

la Direzione Editoriale

Bologna, Gennaio 2003

Nota: poiché negli articoli si fa spesso riferimento agli argomenti trattati specificando la rivista in cui sono apparsi, nel sommario abbiamo riportato, oltre al titolo dell’articolo, anche il numero di rivista in cui è stato pubblicato, per facilitarne il ritrovamento all’interno del cd-rom.

PROGRAMMATORE per microprocessori serie ST6 172
Programmatore LX.1170 per gli ST62/10-15-20-25
CIRCUITO TEST per microprocessore ST6E10 172
Scheda test LX.1171 per provare gli ST6
IMPARARE a programmare i MICROPROCESSORI ST6 174
Istruzioni – Variabili – Registri
IMPARARE a programmare i microprocessori ST6 175
Watchdog – Porte – Interrupt – A/D converter – Timer
BUS per TESTARE i micro ST6 179
Bus LX.1202-1203 per testare i micro ST62/10-15-20-25
SCHEDA TEST per ST6 179
Schede test LX.1204-1205 per provare gli ST6
NOTA per il programmatore LX.1170 per micro ST6 179
Consigli per migliorare il programmatore LX.1170
SCHEDA con 4 TRIAC per microprocessori ST6 180
Scheda LX.1206: pilotare 4 diodi triac con un ST6
SCHEDA con DISPLAY LCD pilotata con un ST6 181
Scheda LX.1207: pilotare un display numerico LCD con un ST6
UNA SCHEDA per pilotare un DISPLAY alfanumerico 182
Scheda LX.1208/N: pilotare un display alfanumerico LCD con un ST6
SOFTWARE emulatore per TESTARE i micro ST6 184
Software simulatore DSE622
SOFTWARE simulatore per TESTARE i micro ST6 185
Formato e Opcode delle istruzioni – Carry flag e Z flag – Correzione degli errori con il DSE622
Windows 95 e ST6 185
Se i programmi in DOS per ST6 non girano sotto Windows 95
PER PROGRAMMARE correttamente i micro ST6 189
Cicli macchina – Reset – Watchdog – Gestione ottimale delle porte – Espressioni
NUOVO software SIMULATORE per micro ST6 190
Nuovo simulatore software per micro ST62/10-15-20-25
SOFTWARE emulatore per TESTARE i micro ST6 190
Le direttive .w_on, .ifc, .block
LE DIRETTIVE dell'assembler ST6 191
Le direttive .ascii, .asciz, .def
PROGRAMMATORE per MICRO ST62/60-65 192
Programmatore LX.1325 per micro ST62/60-65
BUS per TESTARE le funzioni PWM e EEPROM 192
Bus LX.1329 per testare i micro ST62/60-65 – Programmi di esempio per PWM e EEPROM
LE DIRETTIVE dell'assembler ST6 193
Le direttive .byte, .equ, .set
OPZIONI del compilatore Assembler 194
Opzioni del compilatore Assembler
Le memorie RAM–EEPROM 195
Tipi di registri – Memoria RAM e RAM aggiuntiva – Lettura e scrittura della memoria EEPROM
Software SIMULATORE per micro ST6 197
Nuovo simulatore software per micro ST62/60-65
LA funzione SPI per lo scambio DATI 198
La funzione SPI per lo scambio seriale dei dati
CIRCUITI test per la SPI 198
Schede test LX.1380-1381-1382 per la funzione SPI
COME PROGRAMMARE i nuovi MICRO serie ST6/C 202
Interfaccia LX.1430 per gli ST6 serie C – Option Byte della serie C
COME UTILIZZARE la DIRETTIVA .MACRO 203
La direttiva .macro
Per PROGRAMMARE i nuovi MICRO serie ST6/C 204
Le funzioni attivabili tramite l'Option Byte della serie C
LA DIRETTIVA .IFC dell'ASSEMBLER per ST6 205
La direttiva .ifc
IL programma LINKER per i microprocessori ST6 206
Il programma Linker – I formati .hex e .obj – Le direttive .pp_on, .extern, .window, .windowend
APPENDICE A: QUALCOSA in più sul TIMER .....
APPENDICE B: Lampada per cancellare le Eprom 174
INDICE ANALITICO .....

Molti **Istituti Tecnici** e non pochi **softwaristi** e **progettisti** ci chiedono con sempre maggiore insistenza di spiegare in modo molto semplice come si programmano i microprocessori **ST62**, ritenendo che se ci prendiamo questo impegno lo adempiremo come è nostra consuetudine nel migliore dei modi.
Per accontentarvi iniziamo subito dicendo che i microprocessori della famiglia **ST62** sono reperibili in due diverse versioni:
quelli siglati **ST62/E** e quelli siglati **ST62/T**.
La lettera **E** posta dopo la sigla **ST62** indica che il microprocessore si può **cancellare** e **riprogrammare** per almeno un **centinaio** di volte.
I microprocessori **ST62/E** si riconoscono facilmente perché al centro del loro corpo è presente una piccola **finestra** (vedi fig.1) che permette di **cancellare** la **EPROM** interna tramite una lampada a raggi ultravioletti.
La lettera **T**, posta dopo la sigla **ST62**, indica che i dati **memorizzati** all'interno del microprocessore **non** si **possono** più cancellare e quindi nemmeno riscrivere.
Gli **ST62/T** si riconoscono facilmente perché sono **sprovvisti** della **finestra** per la cancellazione (vedi fig.1).
Solitamente i microprocessori **ST62/E** vengono usati per le prime prove, perché in presenza di un eventuale **errore** nei programmi è sempre possibile **cancellare** e riscrive il software.
Solo quando si ha la conferma che il programma funziona regolarmente, si preferisce utilizzare i microprocessori della serie **ST62/T**, perché oltre ad essere meno costosi, non è più possibile manometterli.

Nelle **Tabelle N.1** e **N.2** riportiamo le principali caratteristiche di queste due serie di microprocessori. Tenete presente che nei microprocessori da **2 K** di **memoria** è possibile inserire circa **900 - 990 righe** di programma ed in quelli da **4 K** circa **1.800 - 2.000 righe** di programma.

Per completare i dati riportati nelle due tabelle, precisiamo che il numero a due cifre riportato dopo la sigla, ad esempio **ST62E.10 - 15 - 20 - 25**, ha un preciso significato.

La prima cifra indica la memoria disponibile:
- se la prima cifra è un **1** (vedi **1**0-**1**5) sono disponibili **2 K** di **memoria**,
- se la prima cifra è un **2** (vedi **2**0-**2**5) sono disponibili **4 K** di **memoria**.

La seconda cifra indica i piedini disponibili per i segnali d'ingresso e d'uscita:
- se la seconda cifra è uno **0** (1**0**-2**0**) sono disponibili **12 piedini**,
- se la seconda cifra è un **5** (1**5**-2**5**) sono disponibili **20 piedini**.

PROGRAMMATORE per

TABELLA N.1 micro NON CANCELLABILI

Sigla Micro	memoria utile	Ram utile	zoccolo piedini	piedini utili per i segnali
ST62T.10	2 K	64 byte	20 pin	12
ST62T.15	2 K	64 byte	28 pin	20
ST62T.20	4 K	64 byte	20 pin	12
ST62T.25	4 K	64 byte	28 pin	20

TABELLA N.2 micro CANCELLABILI

Sigla Micro	memoria utile	Ram utile	zoccolo piedini	piedini utili per i segnali
ST62E.10	2 K	64 byte	20 pin	12
ST62E.15	2 K	64 byte	28 pin	20
ST62E.20	4 K	64 byte	20 pin	12
ST62E.25	4 K	64 byte	28 pin	20

microprocessori serie ST6

Si parla spesso dei vantaggi che offrono i microprocessori ST62 senza però spiegare quello che interessa maggiormente, cioè come fare per programmarli e quale programmatore utilizzare. Al contrario noi vi spiegheremo come costruirvi un valido programmatore ed anche come si deve procedere per programmare questi microprocessori.

Se osservate la zoccolatura di questi microprocessori (vedi fig.2-3), potete leggere a fianco di ogni piedino una sigla, e poiché non sempre viene precisato il loro esatto significato, sarà utile spiegarlo.

Vcc - Piedino di alimentazione **positiva**. Su questo piedino va applicata una tensione continua **stabilizzata** di **5 volt**.

TIMER - Applicando su questo piedino un **livello logico 1**, la frequenza del **quarzo** (vedi piedini **3-4**) divisa **x12** potrà giungere sullo **stadio contatore**. Da questo piedino è possibile prelevare un segnale ad onda quadra, la cui frequenza può essere stabilita con le istruzioni del programma.

OSC./In-Out - Sui piedini **3-4** viene applicato un **quarzo** necessario per avere la frequenza di **clock** che serve per far funzionare il microprocessore.

NMI - Questo piedino va sempre tenuto a **livello logico 1**. Applicando a questo piedino un impulso negativo, si informa la **CPU** di interrompere il programma che sta eseguendo e di passare automaticamente ad eseguire una seconda e diversa **subroutine** (sottoprogramma).

Vpp - Questo piedino serve per la programmazione. Durante la fase di programmazione questo piedino, che normalmente si trova a **5 volt**, riceve dal computer una tensione di **12,5 volt**. Quando il microprocessore già programmato viene inserito nella sua scheda di utilizzazione, si deve sempre tenere questo piedino a **livello logico 0**, per evitare di danneggiare i dati in memoria.

RESET - Questo piedino, che si trova sempre a **livello logico 1**, resetta il microprocessore ogni volta che viene cortocircuitato a **massa**. Quando si u-

tilizza un microprocessore già programmato, su tale piedino occorre sempre collegare una resistenza al **positivo** ed un condensatore verso **massa**, in modo da avere un **reset** automatico ogni volta che si alimenta il microprocessore.

PA - PB - PC - Sono le **porte** che la **CPU** può utilizzare singolarmente come **ingressi** oppure come **uscite** tramite programma. Se le utilizzate come **uscite**, per non danneggiarle è consigliabile non collegare dei circuiti che assorbano più di **5 milliAmpere**. Per pilotare dei circuiti che assorbono più di **5 mA**, è necessario interporre tra il microprocessore ed il carico esterno dei **transistor** oppure un **integrato** tipo **SN.74244** o **74HC244** o **74LS244**.

GND - Piedino di alimentazione da collegare a **massa**.

SCHEMA ELETTRICO del PROGRAMMATORE

L'intero circuito programmatore visibile in fig.5 è molto semplice perché richiede solo 3 transistor, due **NPN** ed un **PNP**, due integrati stabilizzatori di tensione **uA.78L05** (vedi IC2-IC3), un integrato digitale **C/Mos** tipo **SN.74HC14** contenente sei inverter a trigger di Schmitt (vedi IC1) ed infine uno zoccolo **textool** a 28 piedini.
Su questo zoccolo andrà infilato il microprocessore **ST62** che si vuole programmare.
Tutte le tensioni necessarie al microprocessore **ST62** vengono prelevate dal secondario del trasformatore **T1**.
I **15 volt** alternati, raddrizzati dal ponte **RS1**, forniscono una tensione **continua** di circa **20 - 21 volt** che raggiunge l'Emettitore del transistor **PNP** siglato **TR2**.
Come si vede nel disegno dello schema elettrico, la Base di questo transistor risulta collegata, tramite la resistenza **R3**, al Collettore del transistor **NPN** siglato **TR1**.
Quando questo transistor riceve dai piedini **2-1** del **Connettore** collegato al computer la necessaria tensione di polarizzazione, porta in conduzione il transistor **TR2** ed in questo modo la tensione positiva di **20 - 21 volt** può raggiungere gli ingressi dei due integrati stabilizzatori siglati **IC2 - IC3**.
L'integrato **IC2** provvede a fornire sulla sua uscita una tensione stabilizzata di **5 volt** per alimentare l'integrato **IC1** ed i piedini **1-5** dell'**ST62** a 28 piedini o il solo piedino **1** dell'**ST62** a 20 piedini.
L'integrato **IC3** provvede a fornire una tensione stabilizzata, sempre di **5 volt**, sul piedino **10** dell'**ST62** a 28 piedini o sul piedino **6** dell'**ST62** a 20 piedini.
Quando tramite computer si desidera **memorizzare** un programma all'interno dell'**ST62**, il piedino **3** del **Connettore**, che normalmente si trova a **livello logico 1**, si commuta sul **livello logico 0** e così la Base del transistor **NPN** siglato **TR3** toglie il **cortocircuito** sul diodo zener **DZ1** da **7,5 volt**.
In questo modo la tensione sull'uscita dell'integrato stabilizzatore **IC3** sale dai **5 volt** iniziali a **12,5 volt** (5 + 7,5 = 12,5).
Da questo istante i dati in **scrittura** giungono dal computer sui terminali **4-6-5-7** del **Connettore** e, prima di raggiungere il **microprocessore ST62**, vengono **squadrati** dai quattro **inverter** siglati **IC1/E - IC1/A - IC1/B - IC1/F**.
Le resistenze **R7 - R5 - R6 - R8**, che abbiamo posto in serie agli ingressi di questi **inverter**, servono per proteggerli nell'eventualità che il **CONN.1** venga per **errore** collegato sulla presa **Seriale** del computer anziché su quella **Parallela**.
Poiché non l'abbiamo ancora precisato, vi segnaliamo fin da ora che il **CONN.1** va inserito nella **PRESA PARALLELA** del computer (presa **LPT1**), alla quale è normalmente collegata la **stampante**.
A **memorizzazione** completata, il computer riporta a **livello logico 1** il piedino **3** del **CONN.1** polarizzando così la Base del transistor **TR3**, che portandosi in conduzione, **cortocircuita** a massa il diodo zener **DZ1**.
Quando il diodo **zener** risulta cortocircuitato, sull'uscita dell'integrato stabilizzatore **IC3** la tensione scende da **12,5** a soli **5 volt** ed in queste condi-

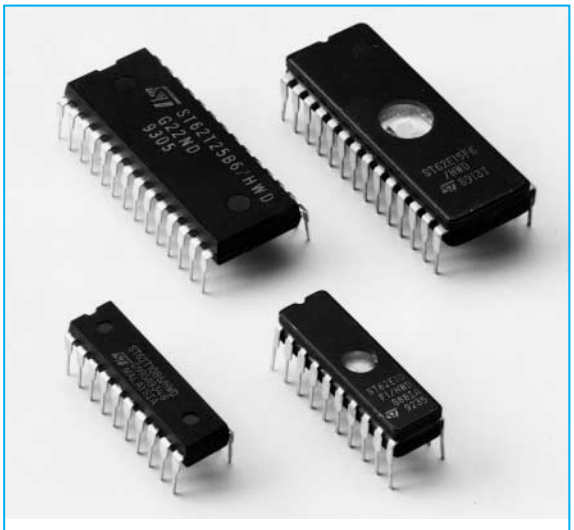
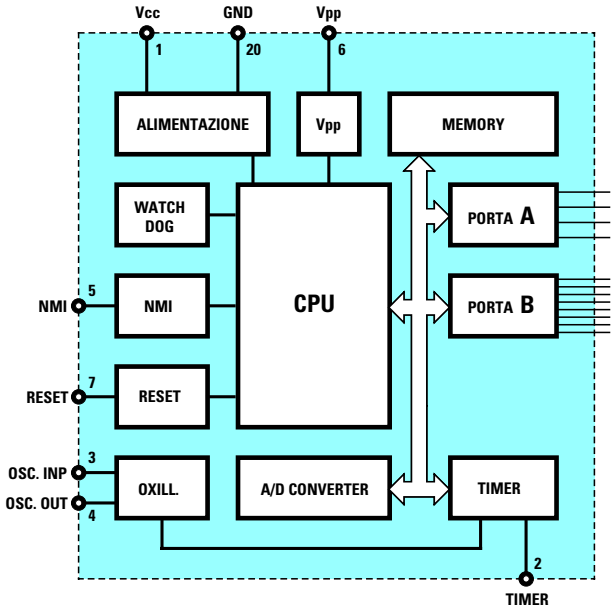
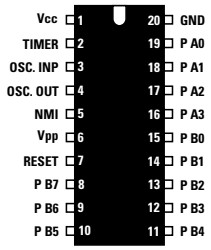


Fig.1 I microprocessori della serie ST62/T sprovvisti di finestra NON sono cancellabili, mentre i microprocessori della serie ST62/E disponendo di una piccola finestra SONO cancellabili. Il numero posto dopo la sigla T o E indica i Kilobyte di memoria e i piedini utili per i segnali di entrata e di uscita (vedi Tabelle 1-2).





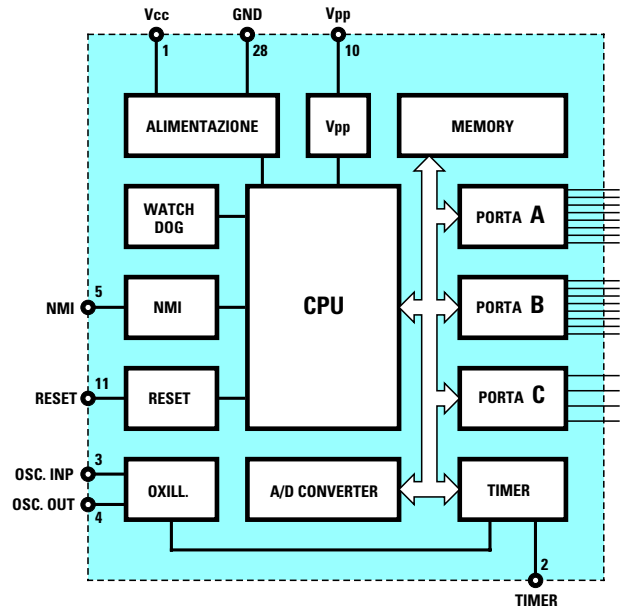
ST 62 E 10

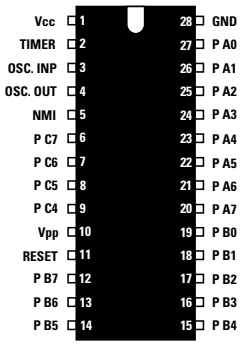
ST 62 T 10

ST 62 E 20

ST 62 T 20

Fig.2 Tutti i microprocessori siglati ST62/E10 e ST62/T10 hanno 2K di memoria utile, mentre quelli siglati ST62/E20 e ST62/T20 hanno 4K di memoria utile. Questi microprocessori a 20 piedini dispongono di 12 porte di entrata o di uscita. La porta A dispone di 4 entrate/uscite (PA1-PA2 ecc.), mentre la porta B di 8 entrate/uscite (PB1-PB2 ecc.)





ST 62 E 15

ST 62 T 15

ST 62 E 25

ST 62 T 25

Fig.3 Tutti i microprocessori siglati ST62/E15 e ST62/T15 hanno 2K di memoria utile, mentre quelli siglati ST62/E25 e ST62/T25 hanno 4K di memoria utile. Questi microprocessori a 28 piedini dispongono di 28 porte di entrata o di uscita. Le porte A-B dispongono di 8 entrate/uscite (vedi PA1, PB1), mentre la porta C di 4 entrate/uscite (vedi PC1).

zioni nessun dato può più essere trascritto nella memoria del microprocessore.
I due inverter **IC1/C - IC1/D**, collegati in **parallelo** ed inseriti in senso inverso rispetto agli altri quattro inverter, vengono utilizzati dal computer per **leggere** i dati dall'**ST62**.
Grazie a questa **uscita** il computer può **rileggere** il programma caricato sul microprocessore e verificare che non vi siano **errori** nella trascrizione dei dati.
In presenza di un **errore** è possibile cancellare il microprocessore e ricopiare nella sua memoria i dati corretti, a patto che l'integrato inserito nel **textool** sia del tipo **ST62/E**.
Nello schema pratico visibile in fig.7 abbiamo raffigurato lo zoccolo **textool** per i microprocessori con **28** piedini e non per i microprocessori con **20 piedini**, ma come vi spiegheremo più avanti, lo stesso zoccolo viene utilizzato per entrambi i microprocessori.
A questo punto possiamo passare alla descrizione della realizzazione pratica e subito dopo vi spiegheremo come procedere per la **memorizzazione** dei **programmi-test** che troverete nel dischetto floppy fornito assieme al kit.
Sono inoltre in preparazione degli articoli teorico-pratici per insegnarvi a scrivere alcuni dei **programmi** che possono svolgere i microprocessori della serie **ST62**.
Vi chiediamo però di concederci un po' di tempo, perché oltre a testare i programmi, vogliamo ricercare tutte le possibili soluzioni per renderli comprensibili a tutti.

REALIZZAZIONE PRATICA

La realizzazione pratica è così semplice che in brevissimo tempo avrete già disponibile il vostro programmatore montato e funzionante.
Sul circuito stampato a fori metallizzati siglato **LX.1170**, dovete montare tutti i componenti richiesti disponendoli come visibile in fig.7.
Potete iniziare inserendo e stagnando i piedini degli zoccoli per l'integrato **IC1** e per il **textool**.
Quest'ultimo deve essere inserito nello stampato rivolgendo la leva di bloccaggio verso il basso, come appare chiaramente visibile nello schema pratico di fig.7.
Dopo questi due componenti potete inserire i due diodi: la fascia **bianca** presente sul corpo plastico del diodo siglato **DS1** va rivolta verso la resistenza R3, mentre la fascia **nera** presente sul corpo in vetro del diodo **zener** siglato **DZ1** va rivolta verso l'alto.
Proseguendo nel montaggio inserite tutte le **resistenze**, i **condensatori** poliestere e l'**elettrolitico**

ELENCO COMPONENTI LX.1170

R1 = 10.000 ohm 1/4 watt
R2 = 47.000 ohm 1/4 watt
R3 = 4.700 ohm 1/4 watt
R4 = 10.000 ohm 1/4 watt
R5 = 220 ohm 1/4 watt
R6 = 220 ohm 1/4 watt
R7 = 220 ohm 1/4 watt
R8 = 220 ohm 1/4 watt
***R9 = 1.500 ohm 1/4 watt**
C1 = 22 mF elettr. 25 volt
C2 = 100.000 pF poliestere
C3 = 100.000 pF poliestere
C4 = 100.000 pF poliestere
C5 = 100.000 pF poliestere
C6 = 100.000 pF poliestere
***C7 = 1.000 mF elettr. 35 volt**
DS1 = diodo EM.513 o 1N.4007
***RS1 = ponte raddriz. 100 V. 1 A.**
DZ1 = zener 7,5 volt
***DL1 = diodo led**
TR1 = NPN tipo BC.547
TR2 = PNP tipo BC.327
TR3 = NPN tipo BC.547
IC1 = C/Mos tipo 74HC14
IC2 = uA.78L05
IC3 = uA.78L05
***F1 = fusibile autoripr. 145 mA**
***T1 = trasformatore 3 watt (TN00.01)**
sec. 15 volt 0,2 Ampere
***S1 = interruttore**
CONN.1 = connettore 25 poli

Nota = I componenti contraddistinti dall'asterisco andranno montati sul circuito stampato siglato LX.1170/B.

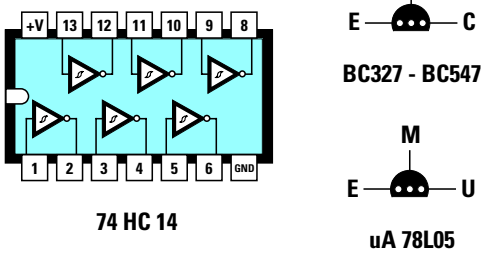


Fig.4 Connessioni dell'SN.74HC14 viste da sopra e dei transistor NPN e PNP e dell'integrato stabilizzatore uA.78L05 viste da sotto.

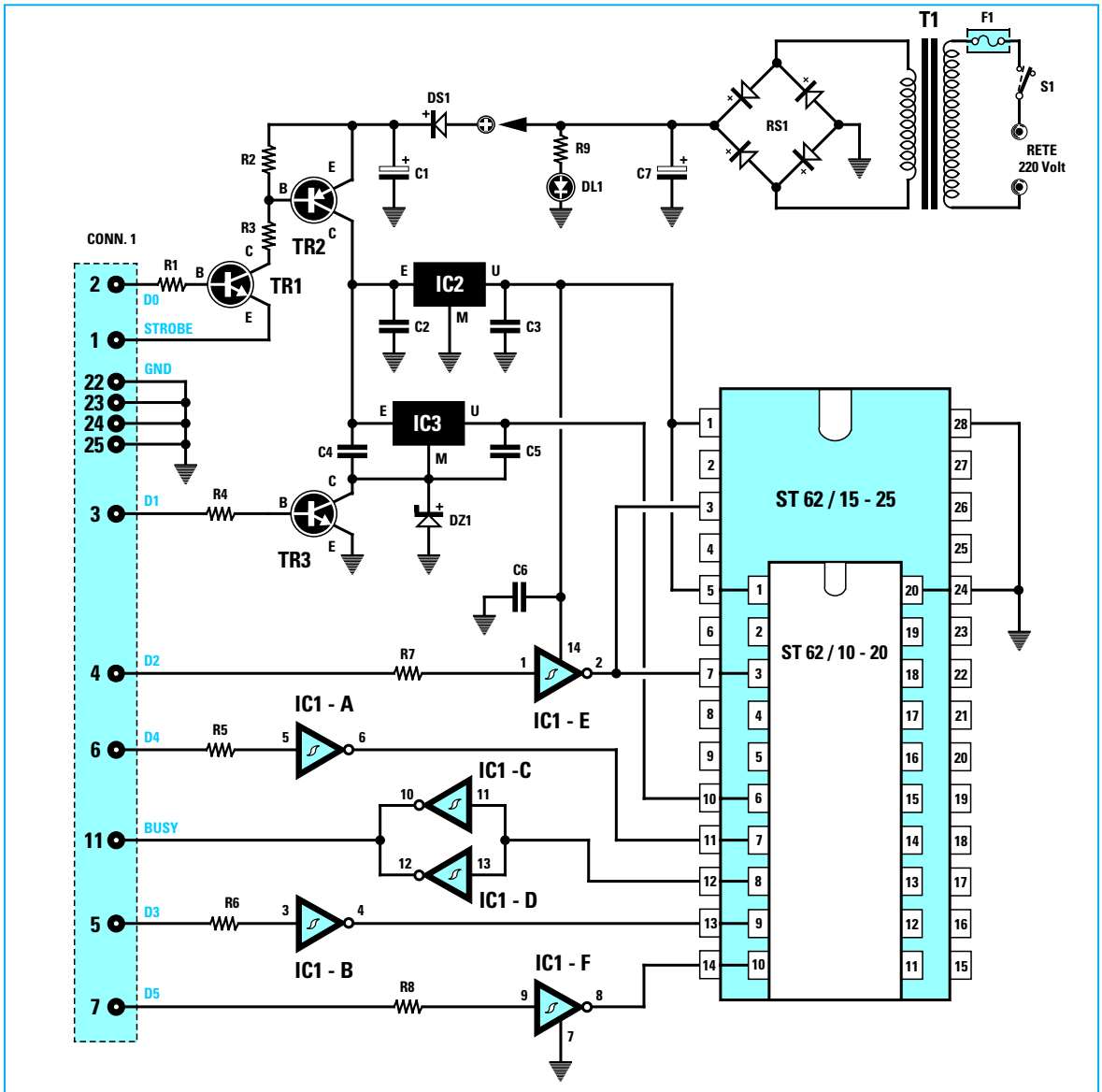


Fig.5 Schema elettrico del programmatore per micro ST62. Il CONN.1 a 25 poli posto sulla destra andrà collegato con un cavetto seriale alla porta PARALLELA del computer, cioè dove ora risulta collegata la STAMPANTE. Dopo aver sfilato il connettore della stampante, dovrete innestare il connettore proveniente da questo PROGRAMMATORE.

C1, che come visibile nello schema pratico di fig.7, deve essere collocato in posizione **orizzontale**.

A questo punto potete inserire i tre **transistor** ed i due integrati **stabilizzatori** e poiché questi ultimi hanno le stesse dimensioni dei transistor, dovete controllare attentamente la loro sigla prima di saldarli sullo stampato.

Come potete vedere nello schema pratico di fig.7, la parte **piatta** dei due **78L05** (IC2 - IC3) va rivolta verso destra e così dicasi per il transistor **BC.547** siglato **TR1**.Gli altri due transistor, siglati **TR2** (un **BC.327**) e **TR3** (un **BC.547**), vanno inseriti rivolgendo la parte **piatta** del loro corpo verso il basso e controllando con molta attenzione le loro sigle, in quanto uno è un **PNP** e l'altro un **NPN**.

Per completare il montaggio non vi resta che inserire sulla parte alta dello stampato il connettore **maschio** d'uscita ed infilare nel suo zoccolo l'integrato **74HC14**, rivolgendo la sua tacca di riferimento verso destra.

Lo stadio di alimentazione verrà montato sul circuito stampato siglato **LX.1170/B**, e poiché questo è un monofaccia, in fig.8 potete osservare le sue dimensioni a grandezza naturale.

Su questo stampato potete inserire come primo

Fig.6 In questa foto potete vedere come si presenta questo programmatore dopo aver montato tutti i suoi componenti. Si noti sulla parte inferiore del circuito stampato lo zoccolo “textool”, che vi permetterà di inserire tutti i microprocessori da programmare senza sforzare i loro piedini.

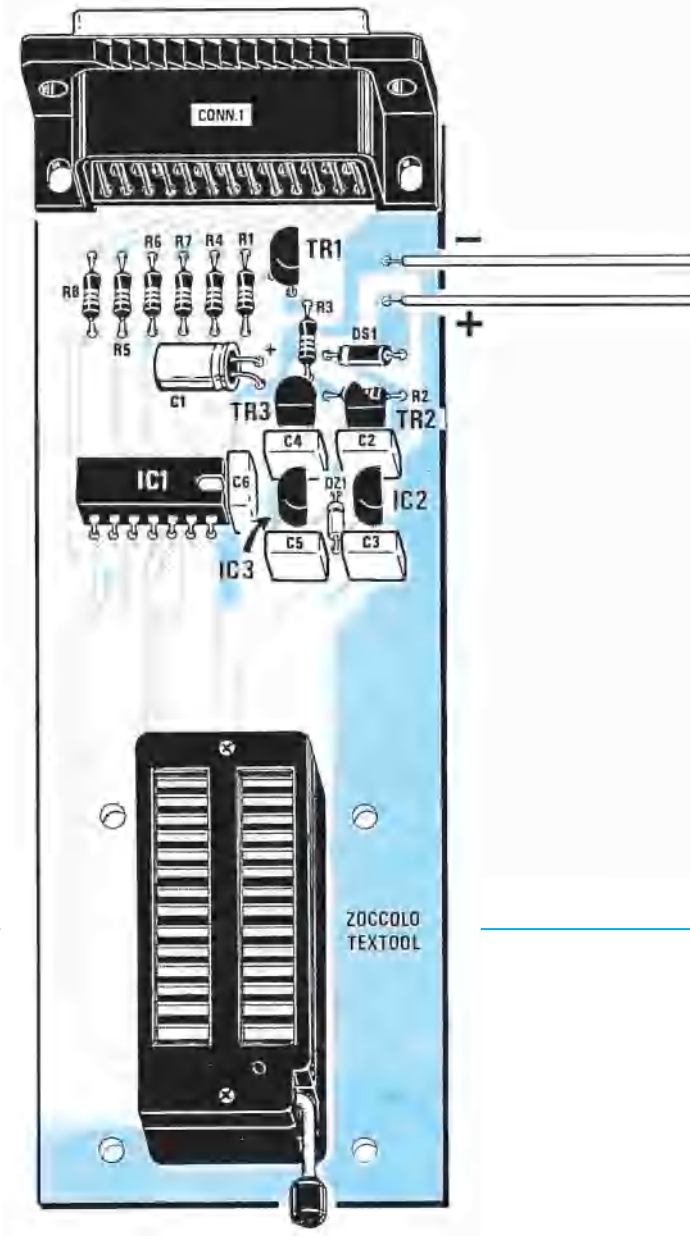


Fig.7 Schema pratico di montaggio dello stadio siglato LX.1170 e, a destra, del suo alimentatore siglato LX.1170/B. Facciamo presente che il CONN.1 può avere una forma diversa da come l’abbiamo disegnata. Se sul connettore fossero presenti due “torrette” (vedi foto), occorrerà toglierle.

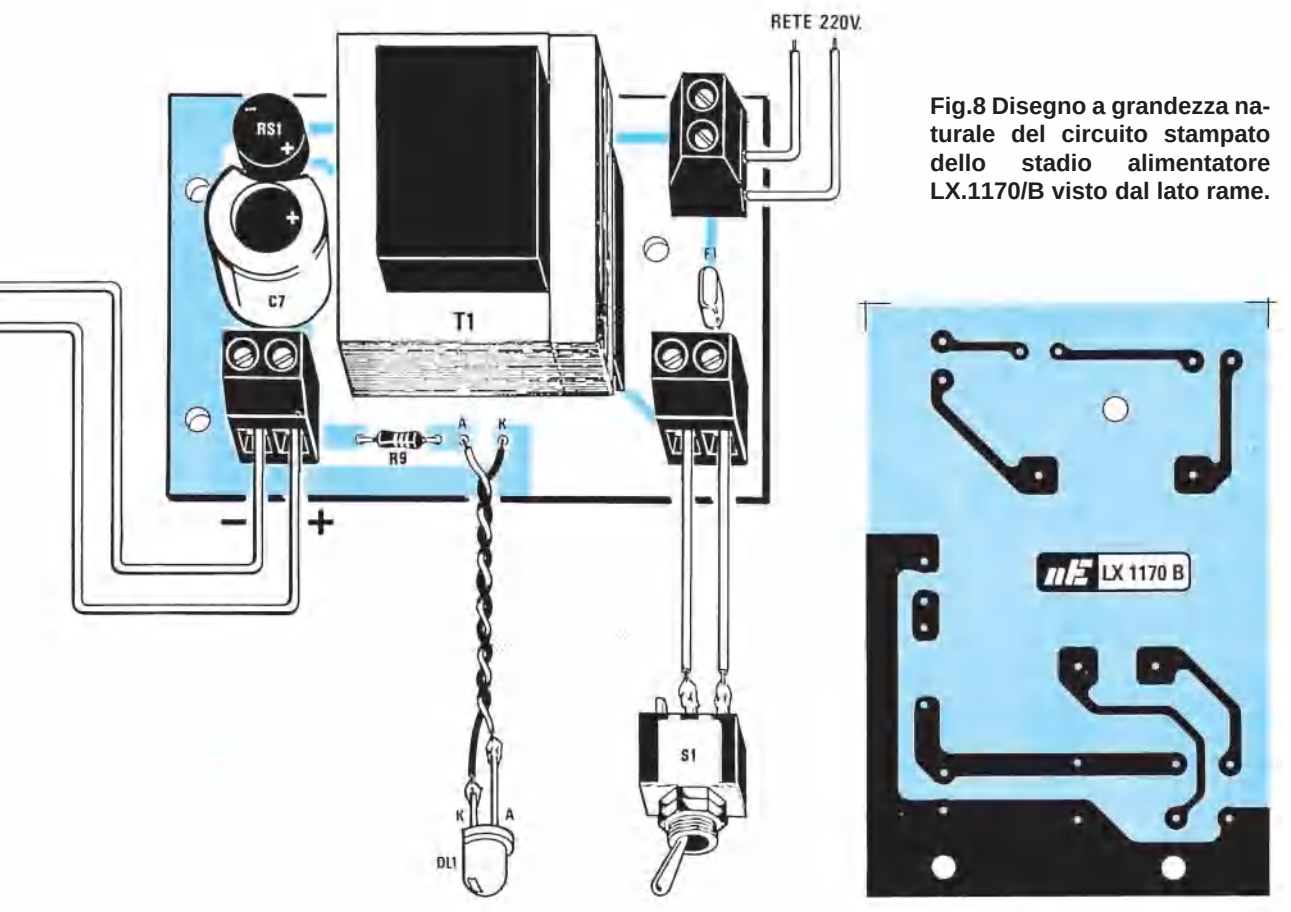


Fig.8 Disegno a grandezza naturale del circuito stampato dello stadio alimentatore LX.1170/B visto dal lato rame.

componente il trasformatore di alimentazione, i cui piedini risultano già predisposti per entrare solo nel loro giusto verso.
Quindi proseguite e completate il montaggio anche di questo stampato inserendo il **ponte raddrizzatore**, il condensatore elettrolitico **C7** rispettando la polarità dei due terminali, la resistenza **R9**, che serve ad alimentare il diodo **led**, ed il **fusibile** autoripristinante siglato **F1**.

MONTAGGIO NEL MOBILE

L'interfaccia verrà fissata dentro un piccolo mobile plastico tipo consolle (vedi fig.11).
Come prima operazione fissate sul mobile il suo **pannello frontale** utilizzando delle viti del diametro di **2 mm** o delle piccole viti autofilettanti.
Su tale pannello fissate con quattro viti lo stampato **LX.1170**, ma prima di eseguire questa operazione dovete stagnare sui due terminali di alimentazione uno spezzone di filo **rosso** per il positivo ed uno di filo **nero** per il **negativo**.
Sul piano del mobile fissate lo stampato dell'alimentatore utilizzando i distanziatori plastici con base **autoadesiva** che trovate nel kit.
Sul piccolo pannello della consolle va invece fissato il **portaled** e l'interruttore di rete **S1**.

A questo punto dovete effettuare i pochi collegamenti richiesti per portare la tensione di alimentazione all'interfaccia **LX.1170**, al diodo **led** ed all'interruttore di rete (vedi figg.7-8).

COME COLLEGARLO al COMPUTER

Dopo aver montato il programmatore siglato **LX.1170** dovete collegarlo alla **presa** della porta **parallela** del computer, cioè a quella che ora utilizzate per la **stampante**. Questa porta si distingue da quella **seriale** perché è **femmina**.
Per questo collegamento non potete usare il connettore che sfilerete dalla stampante, perché questo **non può** innestarsi nel connettore **maschio** presente sull'uscita del programmatore.
Per collegare il programmatore al computer potete usare un qualsiasi **cavo seriale** provvisto ad una

estremità di un connettore **maschio** che va innestato nel **computer**, e dall'altra di un connettore **femmina** che va innestato nel **programmatore**.

IL COMPUTER da USARE

Per programmare gli **ST62** bisogna disporre di un qualsiasi personal computer **IBM compatibile**, non importa se europeo o se costruito ad Hong-Kong o a Taiwan.
A tutti coloro che ci chiedono perché presentiamo programmi per soli IBM compatibili rispondiamo che la maggior parte dei programmi reperibili funzionano sotto **DOS** e poiché questo è il sistema operativo usato su tutti i computer **IBM compatibili** non è possibile adattare i programmi scritti per **DOS** per i computer tipo **APPLE - AMIGA - AMSTRAD** ecc.
Questa scelta non è nostra, ma delle Case di software che avendo constatato che i computer **IBM compatibili** sono i più diffusi in Europa - America - Asia, si sono orientate a realizzare solo programmi per **DOS**.
In questo modo le Case di software vendono un numero maggiore di programmi, quindi riducono i costi di **copyright** ed in più hanno la certezza che questi programmi funzioneranno su qualsiasi modello e marca di computer, perché usati sul **sistema operativo** più diffuso.
Ritornando al computer **IBM compatibile**, non importa di quale marca o tipo e neanche se il modello è vecchio o nuovo, deve soltanto essere dotato di

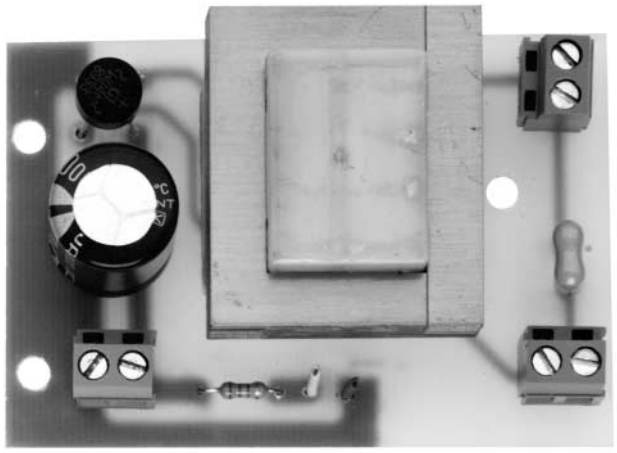


Fig.9 Foto dello stadio di alimentazione che una volta montato dovrete fissare sul coperchio del mobile con tre distanziatori plastici autoadesivi (vedi fig.10).

Fig.10 Lo stampato del programmatore siglato LX.1170 andrà fissato sul pannello del mobile con quattro viti più dado. Sul pannello inclinato dello stesso mobile fisserete il portaled e l'interruttore di accensione.

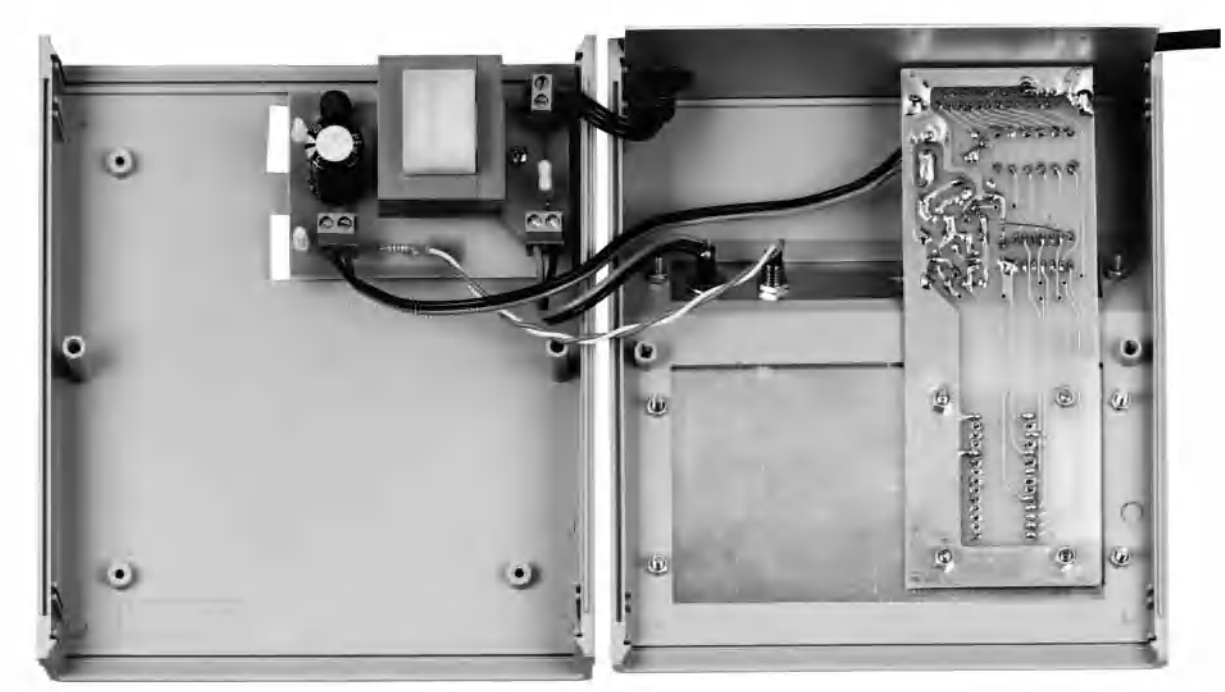


Fig.11 Il mobile scelto per questo programmatore completo delle sue mascherine già forate e serigrafate fornisce al progetto un aspetto molto professionale. Quello che più apprezzerete di questo progetto è la facilità con cui riuscirete, con il dischetto da noi fornito, a programmare qualsiasi tipo di microprocessore ST62.

una **scheda grafica** che rientri nel tipo **CGA - EGA - VGA - SuperVGA**.

INSTALLAZIONE del PROGRAMMA

Con il kit riceverete il **dischetto floppy** fornito dalla **SGS Thomson**, indispensabile per poter programmare tutti i microprocessori della serie **ST62**.
In questo dischetto abbiamo inserito dei **programmi** che vi permetteranno di semplificare tutte le operazioni necessarie per **scrivere** un programma, per **modificarlo** e poi **assemblarlo** ed ovviamente per trasferirlo all'interno della **memoria** di un microprocessore **ST62**.
Il programma vi indicherà inoltre se avete commesso degli **errori**, se avete inserito un **ST62** bruciato, se la memoria del microprocessore è **vergine** o già occupata da un altro programma.

Per iniziare a prendere confidenza con i microprocessori ed imparare a trasferire su questi un programma presente nell'**Hard-Disk**, abbiamo **aggiunto** nello stesso dischetto tre **semplici programmi**, che una volta trasferiti all'interno di un **ST62** vi permetteranno di verificare se avete eseguito correttamente tutte le operazioni di trasferimento dati.
Per copiare nell'Hard-Disk quanto è contenuto in questo **dischetto** dovete eseguire soltanto poche semplici istruzioni.
Quando, dopo aver acceso il computer, sul monitor appare la scritta **C:\>**, inserite il dischetto nell'unità floppy poi digitate:

C:\>A: poi Enter
A:\>installa poi Enter

Nota: Usate solo queste istruzioni e non altre, come ad esempio il **COPY** del **DOS** o le istruzioni dei

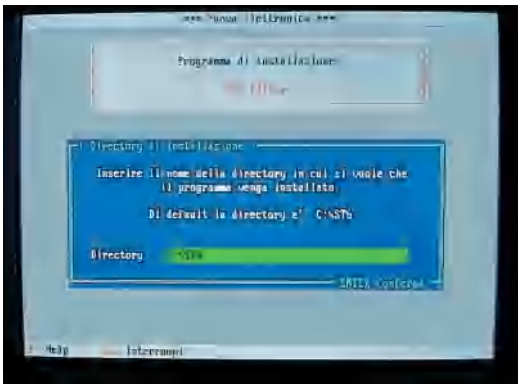


Fig.12 Per trasferire nell'Hard-Disk i programmi contenuti nel dischetto dovete digitare A:\>INSTALLA poi premere Enter. Tutti i programmi verranno memorizzati nella directory C:\ST6.

Fig.13 Poiché i programmi nel dischetto risultano compattati, durante l'operazione di scompattazione apparirà sul monitor l'intero elenco dei files. Il programma occupa 1 Mega circa di memoria.

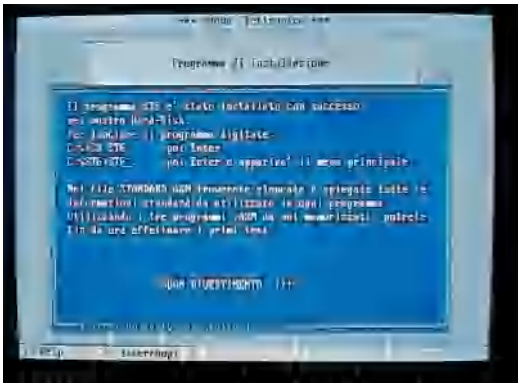


Fig.14 Scompattati tutti i programmi con successo, il computer ve lo segnalerà facendo apparire sul monitor questa scritta. Per uscire da questa finestra pigiate un tasto qualsiasi.

programmi tipo **PCSHELL - PCTOOLS - NORTON Commander**, perché il programma non funzionerebbe.

Con le due semplici istruzioni trascritte sopra, create **automaticamente** una **directory** chiamata **ST6**, nella quale vengono memorizzati tutti i files contenuti nel dischetto.
Durante l'operazione di scompattazione appare sul monitor l'elenco dei **files** (vedi fig.13).
Quando il programma è interamente memorizzato, appare un messaggio a conferma che l'installazione è stata **completata**.
Il programma **scompattato** occupa circa 1 Megabyte di memoria.
Se non premete nessun tasto, dopo qualche minuto compare la scritta:

C:\ST6>

Se volete uscire dalla **directory ST6** sarà sufficiente digitare:

**C:\ST6>CD ** poi Enter

e comparirà così sul monitor **C:\>**.

Una volta installato il programma nell'Hard-Disk potete mettere da parte il dischetto **floppy**, perché non vi servirà più.

COME si RICHIAMA il PROGRAMMA

Tutte le volte che volete richiamare il programma **ST6**, quando sul monitor appare **C:\>** dovete digitare queste semplici istruzioni:

C:\>CD ST6 poi Enter
C:\ST6>ST6 poi Enter

Se dovesse comparire una **directory** diversa da **C:**, ad esempio:

C:\JVFAX>

dovete digitare:

**C:\JVFAX>CD ** poi Enter
C:\>CD ST6 poi Enter
C:\ST6>ST6 poi Enter

Sul monitor comparirà così il **menu principale** (vedi fig.15).

Nota: Le scritte colorate in **azzurro** appaiono direttamente sul monitor, quelle senza colore dovrete digitarle dalla tastiera.

Fig.15 Richiamando il programma con C:\>ST6 Enter, C:\ST6>ST6 Enter, vedrete apparire sul monitor questo "menu". Se premete il tasto funzione F3 apparirà la finestra di fig.16.

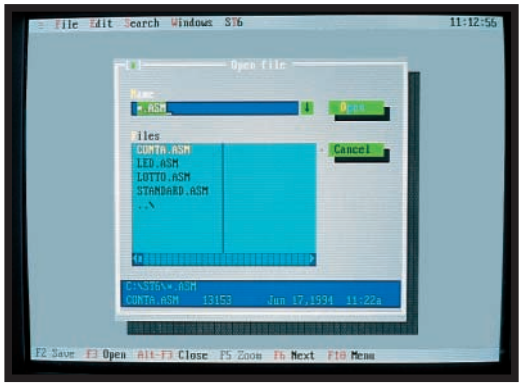


Fig.16 Premendo F3, appariranno in questa finestra i programmi "test" da noi inseriti, cioè Conta - Led - Lotto che potrete trasferire, come spiegato nell'articolo, su un microprocessore ST6 vergine.

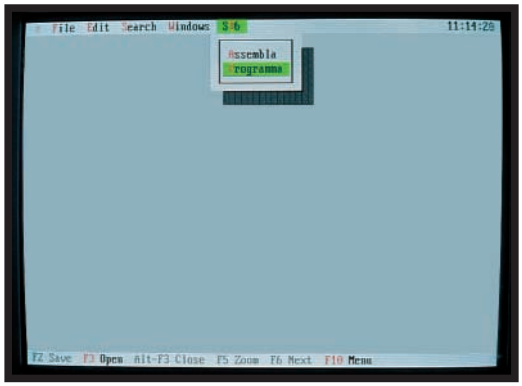


Fig.17 Se portate il cursore sulla scritta ST6 e premete Enter o pigiate i tasti Alt+T, apparirà questa finestra che vi permetterà di programmare l'ST6 inserito nello zoccolo textool del programmatore.

Fig.18 Premendo il tasto P = Programma dopo pochi secondi apparirà sul monitor del computer il software della SGS scritto in lingua inglese. Per continuare pigiate un qualsiasi tasto.

Fig.19 Sullo schermo apparirà una lista con tutti i tipi di ST6 che potete programmare e che sono circa 20. Per selezionare la sigla del vostro microprocessore usate i tasti freccia su e giù.

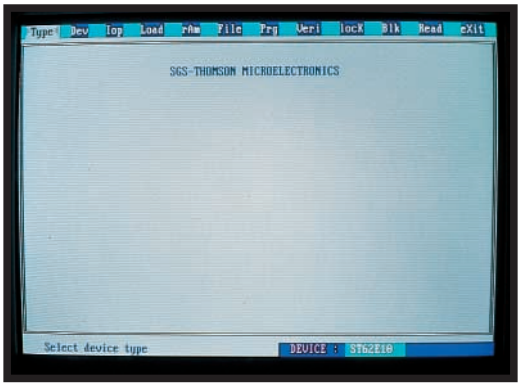


Fig.20 Poiché dovete programmare un ST62E10 portate il cursore su questa sigla poi pigiate Enter. Sullo schermo apparirà questa finestra con in basso l'indicazione dell'ST62E10.

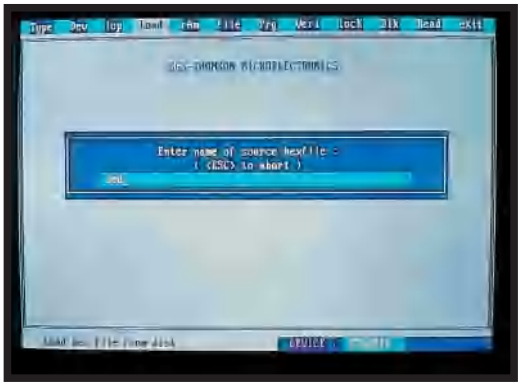


Fig.21 Dalla finestra di fig.20 premete il tasto L = Load e apparirà questa finestra. Qui dovete scrivere il nome del programma che volete trasferire dall'Hard-Disk al microprocessore ST62E10.

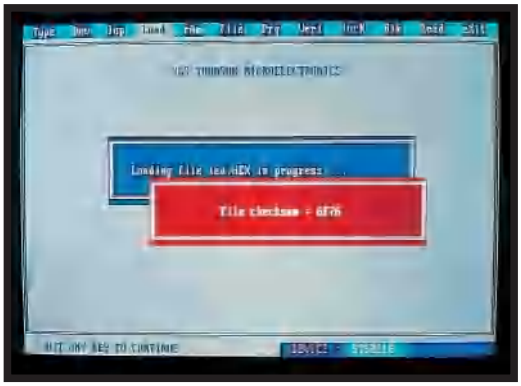


Fig.22 Dopo aver pigiato Enter apparirà la scritta "File checksum" per avvisarvi che il computer ha selezionato il programma, ma non l'ha ancora trasferito sul micro vergine. Per continuare premete un tasto.

Fig.23 Pigiando un qualsiasi tasto apparirà la finestra di fig.20. Per programmare l'ST62E10 che avete inserito nello zoccolo textool del programmatore pigiate il tasto P = Prg e di seguito il tasto N.

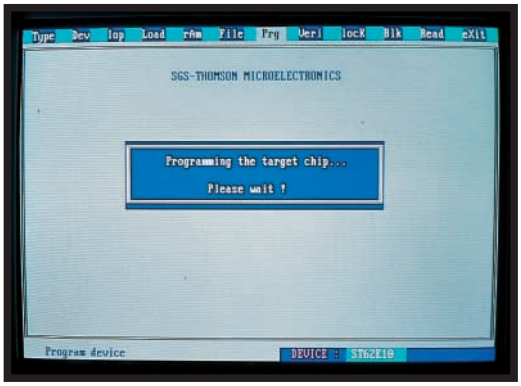


Fig.24 Quando compare questa scritta, non toccate più nessun tasto, perché il computer dopo aver verificato che l'ST62E10 è vergine, provvede a programmarlo impiegando circa 9-15 secondi.

Fig.25 Completata la programmazione, sullo schermo apparirà questa scritta. A questo punto pigiate un qualsiasi tasto e così ritornerete al menu di fig.20. Per uscire basterà premere X.

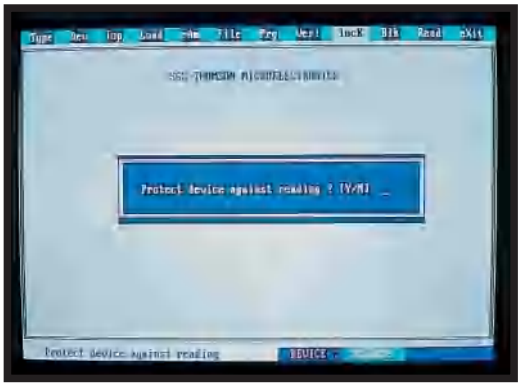


Fig.26 Quando sul monitor appare il menu di fig.20, se volete proteggere il micro dalla lettura dovete premere il tasto K = locK poi Y. L'ST62E10 anche se "protetto" si può cancellare.

A questo punto molti penseranno di aver già risolto tutti i loro problemi, ma poiché non è nostra abitudine illudere nessuno, vogliamo subito precisare che se non conoscete l'architettura di un **microprocessore** e non avete ancora una seppure minima conoscenza generale di come scrivere un programma, saranno necessari dai **3** ai **6 mesi** di pratica per poter diventare **autosufficienti**.
Per questo motivo abbiamo inserito nel dischetto **tre** semplici programmi che oltre a servirvi per effettuare le prime prove pratiche di trasferimento di **dati** verso le **memorie** del **microprocessore**, potranno esservi utili per capire come si imposta un **programma** per **ST62**. Vi spiegheremo infatti anche come richiamare e visualizzare tutte le istruzioni dei vari programmi.

CARICARE un PROGRAMMA

Per trasferire all'interno della **memoria vergine** di un microprocessore **ST62** uno dei tre programmi che noi abbiamo scritto, bisogna innanzitutto inserire il microprocessore nello zoccolo **textool** e **bloccarlo** spostando verso il basso la levetta.
Nel kit del programmatore troverete un **ST62E10** che ha una memoria **EPROM** utile di **2 Kbyte**.
Ovviamente potete caricare uno dei nostri programmi anche su un **ST62E25** da **4 Kbyte** di memoria **EPROM**, che però oltre ad essere più costoso, non può essere utilizzato sulla scheda sperimentale **LX.1171**, pubblicata su questa rivista, perché ha 28 piedini.
Poiché L'**ST62E10** ha soltanto **20 piedini**, dovete collocarlo nello zoccolo come visibile in fig.27, cioè in **basso** e rivolgendo la **tacca** di riferimento verso l'**alto**.
Eseguita questa operazione potete richiamare il programma (vedi paragrafo **Come si richiama il Programma**).
Quando sul monitor del vostro computer appare il menu di fig.15, per proseguire dovete conoscere il **nome** del file da trasferire e per questo dovete semplicemente premere:

F3

Sullo schermo apparirà una nuova finestra con l'elenco dei programmi presenti in memoria (vedi fig. 16). I programmi scritti da noi hanno questi nomi:

CONTA.ASM
LED.ASM
LOTTO.ASM

Nota: Oltre a questi tre files ne troverete un quarto chiamato **STANDARD.ASM**, che a differenza degli altri, **non contiene** un programma da caricare nel microprocessore. In questo file abbiamo voluto inserire tutte le istruzioni **standard** che occorre richiamare in **ogni** programma e che vi risulteranno utilissime nel prossimo articolo, dedicato alle istruzioni dei programmi per **ST62**.

Di questi files ne dovete scegliere **uno solo**, perché all'interno di un microprocessore potete inserire **un solo** programma alla volta.
Ammesso di aver scelto il primo, cioè **LED.ASM**, dovete ricordare il solo nome **LED** tralasciando l'estensione **.ASM**, che **non** vi serve durante la programmazione del microprocessore.
L'estensione **.ASM** è l'abbreviazione della parola **Assembler**.
A questo punto potete uscire da questa **finestra** premendo il tasto **Escape** e vedrete riapparire la pagina del **menu** principale (vedi fig.15).

Tenendo premuto il tasto **ALT** dovete premere il tasto **T = ST6** ed apparirà una finestra con in alto la scritta **Assembla - Programma** (vedi fig.17).

Premete ora il tasto **P = Programma**, e dopo alcuni secondi comparirà l'intestazione del software di programmazione della **SGS** in lingua inglese (vedi fig.18).

Per continuare dovete pigiare un **qualsiasi** tasto e così comparirà la finestra di fig.19.

Premendo i tasti **frecce** giù o su, potete **visualizzare** e **selezionare** tutti i tipi di **microprocessori ST62** che questa interfaccia è in grado di programmare.
Poiché dovete programmare un **ST62E10**, andate con il **cursore** sulla riga in cui appare questa scritta e pigiate **Enter**.

Sul monitor comparirà la pagina di fig.20 ed in basso a destra vedrete la sigla del tipo di microprocessore selezionato, che nel nostro caso è:
DEVICE: ST62E10.

Pigiate il tasto **L = Load** e nella maschera che appare scrivete il nome del file che volete **memorizzare** all'interno dell'**ST62E10**.

Poiché per questo esempio abbiamo scelto il file **LED**, scrivete questo nome nella riga (vedi fig.21) poi premete **Enter**.

Dopo pochi secondi comparirà una seconda finestra rossa (vedi fig.22) con scritto **File checksum =** un **numero esadecimale** di controllo.
Poiché questo numero non vi serve, pigiate un **qualsiasi** tasto.

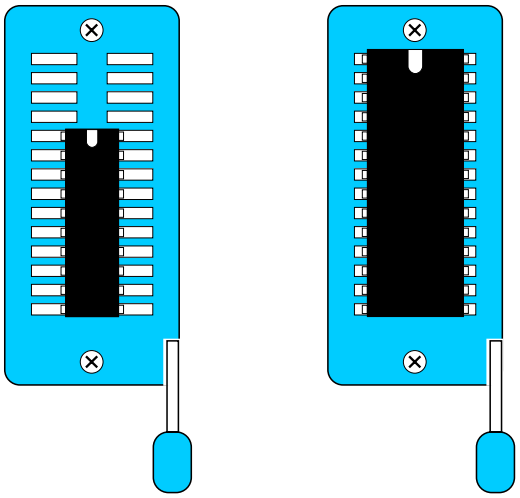


Fig.27 Tutti gli ST62 a 20 piedini andranno inseriti nello zoccolo textool in basso rivolgendo verso l'alto la tacca di riferimento. Anche gli ST62 a 28 piedini andranno inseriti nello zoccolo textool rivolgendo verso l'alto la tacca di riferimento.

Apparirà così la finestra **bianca** visibile in fig.20 e a questo punto dovete solo pigiare il tasto **P = Prg** e sul monitor vedrete la pagina visibile in fig.23.

Ora pigiate il tasto **N** in modo che il computer inizi a **controllare** il microprocessore inserito nello zoccolo **textool**.

Nota: **Non pigiate** mai il tasto **Y** e se per **sbaglio** lo premete, annullate questo comando pigiando il tasto **Escape**, quindi premete ancora il tasto **P** e di seguito **N**.

Dopo aver premuto **N** sul monitor apparirà questa scritta:

Verifying the target chip ... Please Wait
Verifica chip da programmare ... attendi

Se tutto risulta regolare, dopo pochi secondi apparirà sul monitor la finestra di fig.24 con la scritta:

Programming the target chip ... Please wait!
Programmazione in corso ... attendi!

L'operazione di scrittura dei dati dal computer verso le **memorie** del **microprocessore ST62** richiede circa **9 - 15 secondi**.

A programmazione completata sul monitor appare questa scritta (vedi fig.25):

The device is successfully programmed
Microprocessore programmato con successo

Poiché l'operazione di **caricamento dati** nell'**ST6** è completata, potete già estrarre l'**ST62** dallo zoccolo **textool** per inserirlo nel circuito siglato **LX.1171** (vedi articolo su questa rivista a pag.56).

Per uscire dal programma premete un tasto **qualsiasi** e di seguito il tasto **X**. Ritornerete così al menu principale di fig.15.

GLI ERRORI che possono COMPARIRE

Può succedere che per disattenzione premiate il tasto sbagliato o che il microprocessore che inserite nello zoccolo **textool** sia difettoso.
In questi casi sarà il programma a segnalarvi con alcuni messaggi in **inglese** l'anomalia o l'errore commesso cosicché possiate correggerlo.

Target Chip not presente or defective
L'integrato non c'è o è difettoso

Questo messaggio appare ogni volta che vi dimenticate di **inserire** il microprocessore nello zoccolo **textool** oppure quando il microprocessore che avete inserito è **bruciato**.
Non sempre però il microprocessore è fuori uso, perché questo identico messaggio appare anche quando:

- avete inserito il microprocessore nello zoccolo **textool** rivolgendo la **tacca** di riferimento verso il **basso** anziché verso l'**alto**, come visibile in fig.27.

- non avete innestato bene i **connettori** nel computer o nell'interfaccia **LX.1170**.

- vi siete dimenticati di accendere l'**interfaccia** del programmatore.

Device already programmed
Continue Programming? Y/N
L'integrato è già programmato
vuoi continuare? Si/No

Questo messaggio compare quando nello zoccolo **textool** avete inserito un microprocessore **ST62** che risulta **già programmato**.
In questo caso dovete premere il tasto **N** per ritornare così alla finestra di fig.20.
A questo punto potete togliere dallo zoccolo **textool** il microprocessore per **cancellarlo** (vedi paragrafo **Per cancellare un ST62/E**) e quindi riprogrammarlo oppure inserire nello zoccolo un **ST62** vergine e ripetere tutte le operazioni per la programmazione.Vi chiederete allora a cosa serve il comando **Y**, che conferma al programma di proseguire nella programmazione.
Se premete il tasto **Y** lasciando nello zoccolo **textool** l'**ST62** già programmato, non accadrà nulla, cioè il programma presente al suo interno **non si cancellerà** ed il nuovo **non** sarà mai **memorizzato** nella sua memoria.
Poiché nessuno ha mai chiarito quando è possibile usare il comando **Y**, cercheremo di spiegarvelo noi utilizzando degli esempi.
Se durante la fase di programmazione, quando all'interno della memoria del microprocessore è già stato trasferito un **50%** di dati, venisse improvvisamente a mancare la corrente di rete, voi vi trovereste con un microprocessore **programmato** per **metà** che risulterebbe inutilizzabile.
Una volta ritornata la corrente, il **computer** leggendo all'interno dell'**ST62** anche solo una parte di programma, lo considererà **già programmato**, ma se in questo caso premerete il tasto **Y**, il computer trasferirà nella memoria dell'**ST62** il restante **50%** di programma mancante.
Sempre **durante** la fase di programmazione, se si alzasse inavvertitamente la **levetta** dello zoccolo **textool**, i piedini dell'integrato non sarebbero più a contatto e quindi non entrerebbe più alcun dato nel microprocessore.
Poiché qualche dato può già essere entrato nell'**ST62**, ripetendo tutte le operazioni di trasferimento il computer si accorgerà che nelle memorie è già presente un programma e subito lo segnalerà.
Anche in questo caso premendo il tasto **Y**, il computer **completerà** l'inserimento dei dati che in precedenza non erano stati **memorizzati**.

Program result: Device fail at address xxx
Trovato un errore all'indirizzo xxx

Dove **xxx** è un numero esadecimale.
Questo messaggio appare ogniqualvolta il computer non riesce a trasferire correttamente i dati nella **memoria** del microprocessore.
Normalmente questo si verifica quando il microprocessore **ST62** è già stato riprogrammato più di un **centinaio** di volte.
Se questo messaggio compare spesso, è consigliabile sostituire il microprocessore.

Per CARICARE un altro PROGRAMMA

Se volete riutilizzare un microprocessore già programmato per trasferire nella sua memoria un **diverso** programma, dovete prima di tutto **cancellare** i dati al suo interno, dopodiché potete ripetere tutte le operazione già descritte.
Proseguendo nel nostro esempio, se dopo aver **memorizzato** il programma **LED** volete provare le funzioni del programma **CONTA** ed in seguito quelle del programma **LOTTO**, solo dopo aver **cancellato** il microprocessore potrete trasferire dal computer i dati contenuti in uno di questi **files**.

Fig.28 Per cancellare i microprocessori della serie ST62/E e tutti i tipi con EPROM, occorre esporre la loro finestra alla luce emessa da una lampada ultravioletta da 2.300-2.700 Angstrom. Poiché queste lampade non sono facilmente reperibili, abbiamo provveduto ad ordinarne un certo numero ed appena ci perverranno (è prevista una consegna entro settembre) vi presenteremo un completo progetto provvisto di temporizzatore.

Per PROTEGGERE un ST62

Dopo aver constatato che il microprocessore **programmato** funziona correttamente e siete certi che non volete più apportare modifiche al **programma**, ed inoltre non avete più alcuna necessità di rileggere i dati memorizzati al suo interno, vi conviene **proteggerlo**.
Un microprocessore **protetto** tipo **ST62/E** si può **cancellare** per renderlo idoneo a ricevere altri programmi.
Per proteggere un microprocessore, sia del tipo **ST62/T** che del tipo **ST62/E**, lo si deve lasciare inserito nello zoccolo **textool** e procedere come ora vi spiegheremo.

Quando sul monitor appare il menu principale (vedi fig.15), tornate nel menu di programmazione premendo **Alt+T** e di seguito **P** e apparirà la fig.19.
Selezionata la sigla del microprocessore che avete inserito nello zoccolo textool, quando appare il menu di fig.20 premete il tasto **K = Lock** e così apparirà sul monitor la finestra di fig.26.
Per **proteggerlo** sarà sufficiente premere il tasto **Y**, se **non** lo volete proteggere premete il tasto **N**.

Per CANCELLARE un ST62/E

Tutti i microprocessori della serie **ST62/E**, cioè quelli provvisti di una piccola finestra (vedi fig.1), una volta **programmati** si possono **cancellare** e poi nuovamente riprogrammare per utilizzarli con un diverso programma.
Per cancellare questi microprocessori occorre una **lampada ultravioletta** che lavori su una lunghezza d'onda compresa fra i **2.300** e i **2.700 Angstrom**.
Sotto questa lampada va collocato il microprocessore tenendo la sua finestra ad una distanza di circa **2 centimetri**.
A questa distanza per cancellare un microprocessore occorrono dai **15** ai **20 minuti**, sempre che la **finestra** risulti pulita.
Se sopra tale finestra c'è della sporcizia, ad esempio rimangono dei residui di collante dopo aver rimosso un'etichetta autoadesiva, dovrete prima pulirla con un batuffolo di cotone imbevuto di alcool o di acetone.
Poiché la lunghezza del bulbo di una lampada ultravioletta è di circa **30 cm**, potete cancellare contemporaneamente più **ST62/E** disponendoli uno di fianco all'altro (vedi fig.28).

NOTE per la LAMPADA UV

Se vi dimenticate il microprocessore sotto la lampada a **raggi ultravioletti** per una tempo superiore ai **50 minuti** non sono garantite più di **70 - 80 cancellazioni**.
Se volete usare un solo microprocessore per effettuare tantissime prove di **memorizzazione** e **cancellazione**, potete collegare la lampada ad uno dei tanti temporizzatori o timer per lampade da 220 volt pubblicati sulla nostra rivista (ad esempio il Kit **LX.1068** pubblicato sulla rivista **N.153**), che potrete regolare per una accensione massima di **10 minuti** circa.

A lampada **accesa** non fissate **ASSOLUTAMENTE** la luce **viola** che emette, perché **nuoce** gravemente agli occhi.
Per evitare questo inconveniente si potrà mettere sopra la lampada un panno o una scatola di cartone.

CONCLUSIONE

Su questo stesso numero troverete un semplice progetto che oltre a permettervi di controllare se il microprocessore programmato con uno dei tre programmi da noi inseriti nel dischetto, cioè **LED - CONTA - LOTTO**, funziona correttamente, vi consentirà di fare un po' di pratica sulla **cancellazione** di un **ST62/E** e sulla **riprogrammazione**.
In questo articolo vi insegneremo anche ad apportare delle **semplici** varianti sul programma, mentre nei prossimi articoli vi spiegheremo tutto il **set** di **istruzioni** per i microprocessori **ST62**, perché solo conoscendo il significato di queste istruzioni potrete un domani realizzare programmi personalizzati per far svolgere agli **ST62** tutte le funzioni a voi necessarie.

COSTO DI REALIZZAZIONE

Costo di realizzazione dello stadio LX.1170 (vedi figg.6-7) completo di circuito stampato, zoccolo Textool, connettore d'uscita, transistor, integrati con INSERITO un microprocessore ST62/E10, un dischetto floppy contenenti i programmi richiesti, ed il CAVO seriale completo di connettori, ESCLUSI il mobile e lo stadio di alimentazione ..... € 49,10

Costo di realizzazione dello stadio di alimentazione LX.1170/B (vedi fig.8) completo di cordone di alimentazione ..... € 11,60

Il mobile MO.1170 completo delle due mascherine forate e serigrafate ..... € 16,01

Costo del solo stampato LX.1170 ..... € 5,42
Costi del solo stampato LX.1170/B ..... € 1,55

CIRCUITO TEST per

Dopo aver imparato come memorizzare un programma all'interno di un microprocessore ST62E10, e aver constatato di persona che non è poi così difficile come viene invece descritto in altre parti, sarete assaliti dalla curiosità di "testarlo" e per questo vi occorre soltanto il semplice circuito che ora vi presentiamo.

Vogliamo subito precisare che questo circuito serve per testare i programmi che avete imparato a trasferire nel microprocessore **ST62E10** fornito nel kit del programmatore.
Lo stesso circuito può essere utilizzato anche per i programmi che vorrete scrivere, a patto che configuriate le porte come le abbiamo configurate noi, diversamente non potrete sfruttarlo.

In questo circuito di prova, che potete vedere in fig.1, vi sono due integrati, ma quello che abbiamo siglato **IC1** è in pratica il **microprocessore ST62E10** che dovete prima programmare, mentre l'integrato **IC2**, che trovate inserito nel kit, è un **74LS244** utilizzato come **buffer** di corrente.
Infatti dovete tenere presente che sulle uscite dell'**ST62E10** non è possibile applicare dei carichi che assorbano più di **5 mA**, e poiché questo circuito viene utilizzato per accendere dei **diodi led** e dei **display** che assorbono una corrente maggiore, abbiamo dovuto adoperare l'integrato **74LS244**, che è in grado di sopportare carichi fino ad un massimo di **20 mA**.
Sempre guardando lo schema elettrico, sui piedini **3-4** dell'**ST62E10** trovate collegato un quarzo da **8 MHz**, che serve al microprocessore per generare la frequenza di **clock** necessaria per il suo funzionamento.
La frequenza di questo **quarzo** non è critica, quindi potrete utilizzare anche quarzi di frequenza inferiore, ad esempio **7 - 6 - 4 MHz**, tenendo comunque presente che più si scende di frequenza, **più lenta** risulta la velocità di esecuzione del programma.
Non utilizzate quarzi con una frequenza maggiore di **8 MHz**, perché il microprocessore non riuscirà a generare la necessaria frequenza di clock.
Dei tre pulsanti presenti nel circuito, quelli siglati **P1 - P2** svolgono le funzioni rese disponibili dal programma, mentre **P3** serve sempre e solo come comando di **reset**.

Nel **CONN.1** dovete inserire la **scheda** con i **diodi Led**, se avete memorizzato nell'**ST62E10** il programma **LED** o la **scheda** con i due **Display** se avete memorizzato nell'**ST62E10** il programma **CONTA** o **LOTTO**.
Per alimentare questa scheda occorre una tensione **stabilizzata** di **5 volt 200 milliAmpere** circa.

REALIZZAZIONE PRATICA

Sul circuito stampato siglato **LX.1171** dovete montare tutti i componenti disponendoli come visibile in fig.12.
Lo schema è così semplice che non ha certo bisogno di particolari consigli, comunque una volta stagnati tutti i terminali degli zoccoli e del connettore è consigliabile controllare con una lente d'ingrandimento che non vi sia una goccia di stagno tra due piedini che provochi un **corto**.
Come visibile nel disegno dello schema pratico conviene collocare il quarzo in posizione **orizzontale**, saldando il suo corpo sul circuito stampato con una goccia di stagno.
Nello zoccolo **IC2** (quello posto in alto verso il **CONN.1**) inserite l'integrato **74LS244** rivolgendo la tacca di riferimento verso il condensatore **C2**.
Nello zoccolo **IC1** inserite dopo averlo **programmato** il microprocessore **ST62E10**, rivolgendo la tacca di riferimento verso il condensatore **C1**.
Completato il montaggio di questo stampato potete prendere lo stampato siglato **LX.1171/B** e su questo saldare il **connettore maschio**, tutte le resistenze dalla **R3** alla **R10** ed i diodi **led**, come visibile in fig.14.
Quando inserite i diodi led nel circuito stampato dovete rivolgere il terminale **più corto** (terminale **K**) verso le resistenze.
L'ultimo stampato, quello siglato **LX.1171/D**, provvisto di due **display** vi servirà per testare i programmi **CONTA** e **LOTTO**.

microprocessore ST62E10

Come visibile in fig.15 su questo stampato fissate le resistenze da **R1** ad **R8**, che sono da **220 ohm**, e le due resist,enze **R9 - R10** che sono invece da **4.700 ohm**, poi il connettore maschio, i due transistor **TR1 - TR2** rivolgendo la parte piatta del loro corpo verso destra e per ultimi montate i due display, rivolgendo il lato con i **punti decimali** verso il basso.

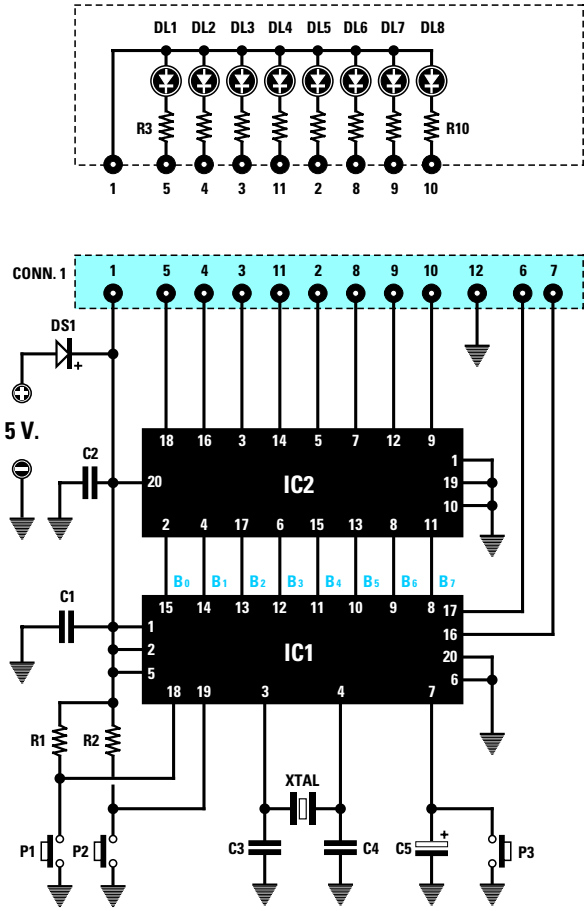
IMPORTANTE

Se nel microprocessore **ST62E10** avete memorizzato i dati del **programma LED**, dovrete inserire nel **CONN.1** della scheda **LX.1171** la scheda con gli **8 diodi led**, se avete memorizzato i dati del **programma CONTA** o del **programma LOTTO**, dovrete inserire la scheda con i **2 display**.
Se per errore scambiate le schede, **non causerete** nessun danno né all'integrato **IC2** né al microprocessore **IC1**, quindi basterà inserire la scheda giusta per vedere il circuito **funzionare**.
Se il circuito **non funziona**, potreste esservi sbagliati nel **memorizzare** il programma nelle memorie del microprocessore.
In questo caso dovrete esporlo sotto una luce ultravioletta per **cancellare** i dati dalla sua memoria, quindi dovrete **riprogrammarlo**.

Questa stessa operazione va effettuata se dopo aver **trasferito** il programma **LED** volete sostituirlo con il programma **CONTA** o con il programma **LOTTO**.

COLLAUDO MICROPROCESSORE nel CIRCUITO TEST

Dopo aver realizzato il circuito test siglato **LX.1171** potete collaudare il microprocessore che avete imparato a programmare con uno dei tre semplici programmi **LED - CONTA - LOTTO**, come vi abbiamo spiegato nell'articolo precedente.
Per prima cosa dovete inserire nello zoccolo a **20 piedini** del circuito test **LX.1171** il microprocessore appena programmato, rivolgendo la sua tacca di riferimento verso il condensatore **C1** (vedi fig.12).
Dopo questa operazione, se avete programmato il microprocessore con il programma **LED** dovete innestare nel connettore **femmina** del circuito test **LX.1171** il circuito applicativo a diodi led siglato **LX.1171/B**.
Se invece avete programmato il microprocessore con uno qualsiasi dei due programmi **CONTA** o **LOTTO**, dovete innestare nel connettore **femmina**



ELENCO COMPONENTI LX.1171 (test St6)

R1 = 10.000 ohm 1/4 watt
R2 = 10.000 ohm 1/4 watt
***R3-R10 = 100 ohm 1/4 watt**
C1 = 100.000 pF poliestere
C2 = 100.000 pF poliestere
C3 = 22 pF a disco
C4 = 22 pF a disco
C5 = 1 mF elettr. 63 volt
DS1 = diodo EM.513 o 1N.4007
***DL1-DL8 = diodi led**
XTAL = quarzo 8 MHz
IC1 = vedi nota
IC2 = TTL tipo 74LS244
P1-P3 = pulsanti
CONN.1 = connettore 12 poli

Nota = I componenti contraddistinti dall'asterisco andranno montati sul circuito stampato siglato LX.1171/B. L'integrato siglato IC1 è il microprocessore incluso nel Kit del porgrammatore LX.1170, che potrete inserire nel circuito dopo averlo programmato.

Fig.1 Schema elettrico del circuito che dovrete realizzare per poter verificare se il programma LED (vedi pagg.37-38) è stato correttamente memorizzato all'interno del micro ST62E10. Per alimentare questo circuito occorre una tensione esterna stabilizzata di 5 volt.

del circuito test il circuito applicativo con i due **display** siglato **LX.1171/D**.
A questo punto potete passare al collaudo vero e proprio del programma caricato nelle memorie del microprocessore.

COLLAUDO PROGRAMMA LED

Dopo aver programmato il microprocessore con il programma **Led** ed averlo inserito nel circuito stampato siglato **LX.1171**, dovete innestare il connettore **maschio** del circuito a **diodi led** nel connettore **femmina** presente sul circuito test, poi dovete alimentare quest'ultimo con una tensione di **5 volt** stabilizzati.

Il programma **LED** vi dà la possibilità di far lampeggiare gli **8 diodi** led presenti nel circuito con **5** diverse **modalità**, che potete selezionare pigiando ripetutamente il pulsante **P1**.

1° Lampeggio

I led si accendono in sequenza uno alla volta da **sinistra** verso **destra**. Il ciclo continua all'infinito.

2° Lampeggio

I led si accendono due alla volta dall'**esterno** verso l'interno (prima **DL1** e **DL8**, poi **DL2** e **DL7** ecc.), fino ai due led centrali (**DL4** e **DL5**), poi i led si accendono sempre due alla volta, ma in senso inverso, cioè dall'interno verso l'esterno. Il ciclo continua all'infinito.

3° Lampeggio
Si accende tutta la fila di led, iniziando dal primo a **sinistra** e proseguendo verso **destra**. Quando sono tutti accesi si spengono tutti insieme. Il ciclo riprende all'infinito.

4° Lampeggio
Lampeggiano uno alla volta prima i led **pari** poi i **dispari**, poi si spengono e si accendono tutti insieme. Il ciclo continua all'infinito.

5° Lampeggio
I led si accendono prima tutti insieme, poi si spengono tutti insieme. Il ciclo si ripete all'infinito.

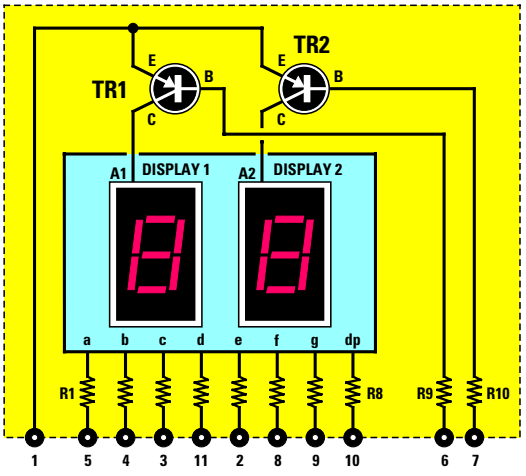
Non appena il circuito viene alimentato, il microprocessore esegue il **1° motivo**. Premendo ripetutamente il pulsante **P1** vengono eseguiti uno di seguito all'altro il **2° - 3° - 4° - 5° motivo**. Se mentre è in corso il **5°** premete nuovamente **P1**, il microprocessore eseguirà di nuovo il **1° lampeggio**.
L'intervallo fra un'accensione dei diodi led e l'altra è di circa **1/2 secondo**, ma è possibile **diminuire** questo tempo premendo ripetutamente **P2**.
La massima velocità di lampeggio consentita dal programma viene raggiunta dopo aver premuto questo pulsante per **8 volte**.
Premendolo ancora una volta, il lampeggio riprenderà con la stessa velocità iniziale.
Premendo in qualunque momento il pulsante **P3** (**RESET**), il microprocessore tornerà ad eseguire il programma da capo, cioè ripartirà dal **primo** lampeggio come se aveste alimentato solo in quel momento il circuito.

Se volete passare al collaudo di uno degli altri due programmi **CONTA - LOTTO**, dovete togliere l'alimentazione al circuito ed estrarre il circuito a led **LX.1171/B**. Ovviamente dovete pure estrarre il microprocessore, e dopo averlo **cancellato**, dovete **riprogrammarlo** con uno degli altri due programmi.

COLLAUDO PROGRAMMA CONTA

Dopo aver programmato il microprocessore con il programma **Conta** ed averlo inserito nel circuito stampato siglato **LX.1171**, dovete innestare il connettore **maschio** del circuito a **display** nel connettore **femmina** presente sul circuito test, poi dovete alimentare quest'ultimo con una tensione di **5 volt** stabilizzati.

Dopo aver alimentato il circuito vedrete comparire sui display il numero **00**, che aumenterà di una unità ogni **1/2 secondo**.
Pertanto ogni 5 decimi di secondo leggerete **01 - 02 - 03 - ecc.** fino a **99**, dopodiché il conteggio ripartirà sempre in avanti da **00**.
Per ottenere un conteggio all'**indietro**, potete premere in qualunque istante il pulsante **P2**.
Supponendo di premere **P2** quando sui display compare ad esempio il numero **74**, vedrete apparire, sempre ad intervalli di **1/2 secondo**, i numeri



ELENCO COMPONENTI LX.1171/D

R1-R8 = 220 ohm 1/4 watt
R9 = 4.700 ohm 1/4 watt
R10 = 4.700 ohm 1/4 watt
TR1 = PNP tipo BC327
TR2 = PNP tipo BC327
DISPLAY1-2 = display Anodo comune tipo HP.5082 o 7731

Fig.2 Se all'interno del micro ST62E10 avete memorizzato il programma CONTA o LOTTO, per poterlo controllare dovrete realizzare questo circuito elettrico. La scheda dei diodi led o dei display andrà inserita nel connettore dell'LX.1171 (vedi figg.19-20).

73 - 72 - 71 - ecc., fino a **00**, dopodiché il conteggio riprenderà da **99** per tornare a **00** e così all'infinito.
Premendo in qualunque momento il tasto **P1** il conteggio proseguirà di nuovo in **avanti** e così pure premendo in qualsiasi momento il pulsante **P2** il conteggio riprenderà all'**indietro**.
Infine potrete riprendere l'esecuzione del programma da capo premendo il pulsante **P3** (**RESET**), perché in tal modo sarà come se aveste appena alimentato il circuito.
In questo caso il conteggio ripartirà da **00** e verrà effettuato in **avanti**.

COLLAUDO PROGRAMMA LOTTO

Dopo aver programmato il microprocessore con il programma **Lotto** ed averlo inserito nel circuito stampato siglato **LX.1171**, dovete innestare il connettore **maschio** del circuito a **display** nel connettore **femmina** presente sul circuito test, poi dovete alimentare quest'ultimo con una tensione di **5 volt** stabilizzati.

Dopo aver alimentato il circuito vedrete comparire sui display due lineette (**--**) ed ogni volta che premerete il pulsante **P1** comparirà un numero sempre diverso compreso fra **01** e **90**, cioè i numeri della tombola o del **lotto**.
Ogni volta che premete **P1** il numero non sarà **mai uguale** ai **precedenti**, quindi potrete simulare una reale estrazione di numeri.
Una volta estratti tutti i **90 numeri**, vedrete comparire sui display le due lineette (**--**), quindi saprete che sono stati estratti tutti i **90 numeri** disponibili.
Quando compaiono le due lineette (**--**), per iniziare una nuova estrazione basterà premere **P1**, e così sempre in maniera casuale ricompariranno i numeri compresi fra **00** e **90**.
Se invece volete iniziare una nuova estrazione interrompendo quella in corso, sarà sufficiente **resettare** il microprocessore premendo il pulsante **P3** (**RESET**). In tal modo il programma verrà eseguito da capo, esattamente come se aveste appena alimentato il microprocessore, anche se i numeri non sono stati tutti estratti.
In questo programma il pulsante **P2** non viene utilizzato.

PER VEDERE il LISTATO di un PROGRAMMA

Le informazioni seguenti vi saranno utili quando vorrete entrare nel listato di un programma per modificarlo.
Per visualizzare un qualunque listato di uno dei pro-

Fig.3 Per poter vedere il listato dei programmi, quando sullo schermo appare il menu principale, pigiate F3 quindi scegliete quello che vi interessa, cioè Conta, Led o Lotto e premete Enter.

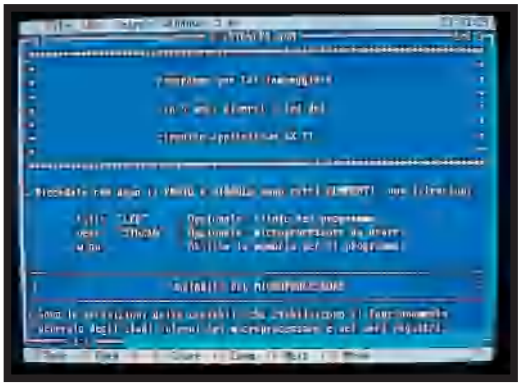


Fig.4 Se sceglierete il programma Led, sullo schermo del computer appariranno tutte le righe di tale programma. Per uscire da questo listato dovete premere Alt ed F3 e apparirà il menu principale.

Fig.5 I più esperti potranno apportare personali modifiche a questi programmi pigiando il tasto F2 per memorizzarle. Pigiando Alt+F3 potrete non confermare le modifiche apportate.

gramma per **ST62**, anche senza bisogno di modificarlo, dovete innanzitutto caricare il programma. Quando sul monitor del computer compare:

C:\>

digitate:

C:\>CD ST6 poi Enter
C:\ST6>ST6 poi Enter

Così entrerete nel menu principale di fig.3.

Premete il tasto **F3** per visualizzare l'elenco dei files contenenti i programmi per **ST62**.
Premete **Enter** e dopo aver portato il cursore sul nome del file desiderato, premete ancora **Enter**. In questo modo comparirà il listato del programma contenuto in quel file.

Per muovervi all'interno del listato e vedere così tutte le istruzioni usate i tasti freccia **su/giù** oppure i due tasti **pagina su/giù**.

Per **uscire** dal listato di un file, dovete tenere premuto **Alt** e premere **F3**. Ritornerete così al menu principale (vedi fig.3).

Nota: Se mentre visualizzate il listato premete per errore i tasti scrivendo nel file dei caratteri indesiderati, senza curarvi di andarli a cancellare, potete uscire dal file **senza** salvare le modifiche.
Per compiere questa operazione è sufficiente tenere premuto il tasto **Alt** e premere **F3**, e quando appare la finestra di conferma di fig.7 dovete premere il tasto **N**.
Ricordate che se dopo aver modificato il file senza volerlo, premete inavvertitamente **F2**, le modifiche verranno **salvate**, quindi premendo **Alt+F3** la finestra di conferma modifiche (vedi fig.7) **non apparirà**.
In questo caso l'unico modo per correggere le modifiche è **entrare** di nuovo nel file, cercare la riga del listato dove avete apportato le modifiche e correggerla.
Nel caso non riusciate a correggere l'errore neanche in questo modo, non vi rimane altro che **installare** di nuovo il programma, perché in tal caso caricherete nell'Hard-Disk i **programmi originali** contenuti nei files **LED - CONTA - LOTTO**.

Per MODIFICARE un PROGRAMMA

Chi sa già programmare potrebbe trovare questo paragrafo poco interessante, ma poiché dobbiamo pensare anche a tutti i principianti, riteniamo necessario spiegare anche quello che per molti potrebbe essere ovvio.

Fig.6 Una volta salvate le modifiche con F2 (vedi fig.5) dovrete riassemblare tutto il programma. Premete Alt+T e quando apparirà questa finestra pigiate il tasto A. L'assemblaggio dura solo pochi secondi.

Fig.7 Se non volete memorizzare le modifiche effettuate NON dovrete pigiare il tasto F2, ma solo Alt+F3 e così apparirà questa finestra. A questo punto dovrete semplicemente pigiare N.

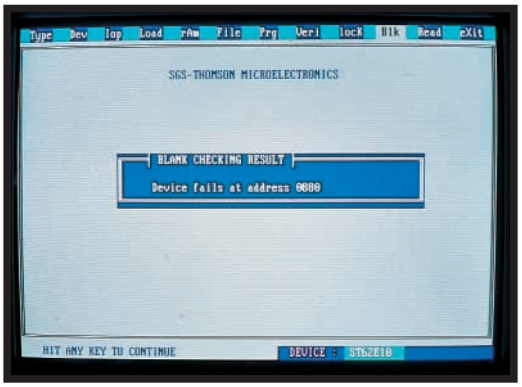


Fig.8 Se prima di programmare un ST62E10, seguendo quanto descritto da pag.26, pigiate il tasto B, il computer vi dirà se il microprocessore inserito nello zoccolo textool è vergine.

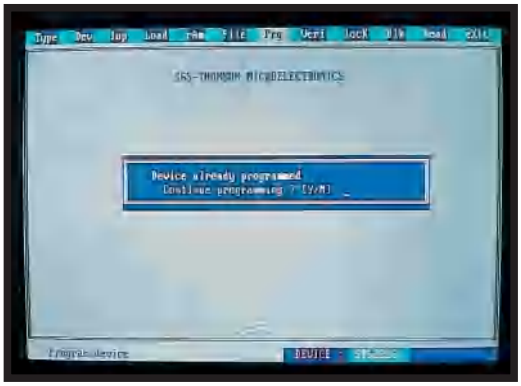


Fig.9 Se tentate di programmare un ST62E10 già programmato il computer vi mostrerà questa scritta. Se volete inserire un diverso programma, dovrete premere N e cancellare il micro.

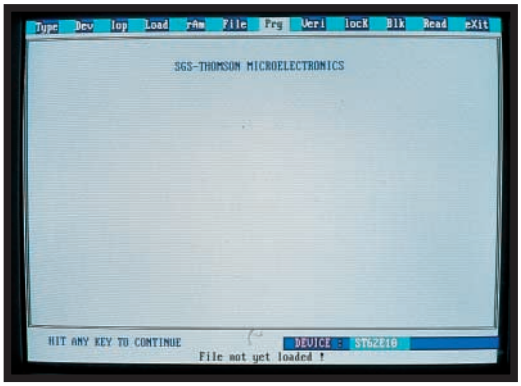


Fig.10 Se nel programmare un microprocessore vi dimenticate di scrivere il nome del file del programma, Conta - Led - Lotto, il computer lo segnalerà con questo messaggio.

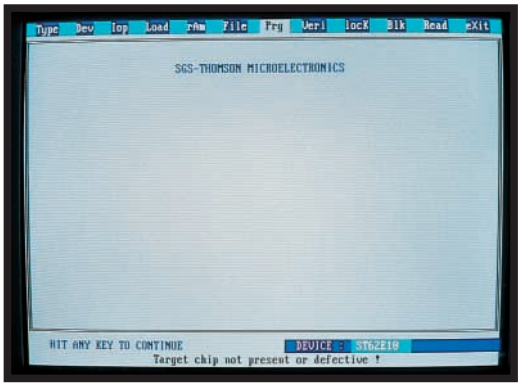


Fig.11 Se il microprocessore è stato inserito nello zoccolo textool in modo errato o se il programmatore non è alimentato, sullo schermo del computer apparirà questo messaggio.

Sapere come **entrare** in un file e come procedere per **modificare** qualcuna delle istruzioni dei programmi che vi abbiamo fornito, costituisce un primo importante passo per tutti coloro che non hanno mai visto come sono scritte le diverse righe di un programma e che in un secondo tempo vorranno provare a realizzare dei semplici e personali programmi.

Per spiegarvi queste prime cose prenderemo spunto dal programma più semplice che vi abbiamo proposto, quello cioè chiamato **LED**, e su questo vi insegneremo come si deve procedere per **cambiare** le modalità di lampeggio degli 8 diodi led, cioè per fare in modo che i diodi led possano lampeggiare in modo **diverso** da quello da noi proposto.

Quando, dopo aver caricato il programma, compare il menu principale di fig.3, premete il tasto **F3** (tasto per l'**apertura dei files**).

Apparirà la finestra con l'elenco dei file dei programmi, cioè:

CONTA.ASM
LED.ASM
LOTTO.ASM
STANDARD.ASM

Nota: Il file **STANDARD.ASM** non contiene un programma vero e proprio, ma delle utili indicazioni per capire il significato, l'uso e l'importanza delle varie istruzioni di ogni programma per **ST62**. Come vi spiegheremo nel prossimo paragrafo, potrete entrare in questo file e leggere tutti gli utili commenti che abbiamo inserito.

A questo punto premete **Enter**, portate il **cursore** sulla riga **LED.ASM** e premete ancora **Enter**.

Sul monitor comparirà tutto il **listato** del programma contenuto nel file **LED.ASM** (vedi fig.4).

In basso a sinistra sono presenti due numeri separati dai **due punti** (**:**). Il **primo** numero vi permette di identificare la riga del programma, il **secondo** la colonna del file.
Con i tasti **freccia giù** o **pagina giù** portate il cursore in prossimità della riga **255**, cioè scendete con il cursore fino a quando in basso a sinistra non leggete **255:1**.
Dalla riga **255** in poi (vedi fig. 5) compaiono delle istruzioni del tipo:

```
lamp1    .byte 11111110b ; Prima istruzione
         .byte 11111101b ; Seconda istruzione
```

.byte 11111011b ; Terza istruzione
.byte 11110111b ; Quarta istruzione
.byte 11101111b ; Quinta istruzione
.byte 11011111b ; Sesta istruzione
.byte 10111111b ; Settima istruzione
.byte 01111111b ; Ottava istruzione

dove **lamp1** sta ad indicare che le istruzioni successive sono relative alla prima modalità di lampeggio dei diodi led.

Nota: Le scritte dopo il **punto e virgola** (**;**) **non** sono **istruzioni**, ma **commenti**, che abbiamo inserito appositamente nel listato per rendere più comprensibili le spiegazioni che ora vi daremo. Per questo motivo vi consigliamo di **non** cambiarle.

Per capire in che modo è possibile cambiare queste istruzioni per variare la modalità di accensione dei vari diodi led, cercheremo di spiegarvi in modo molto semplice come funzionano queste istruzioni.

Le **cifre** siglate **1** e **0** che trovate dopo l'istruzione **.byte** compongono un numero **binario**, riconoscibile per la presenza della lettera **b = b**inario.
Come noterete, queste **cifre binarie** sono **8** e ad ognuna di esse corrisponde un diverso **diodo led** del circuito test **LX.1171/B** (vedi fig.1).
Ad esempio, alla prima cifra da **destra** corrisponde **DL1**, alla **seconda** corrisponde **DL2**, e così via fino all'**ottava** cifra da destra, alla quale corrisponde **DL8**.
Ogni volta che il microprocessore esegue un'istruzione come:

.byte 11111011b ; Terza istruzione

i diodi led corrispondenti alle **cifre binarie** uguali a **0** vengono **accesi**, mentre i diodi led corrispondenti alle **cifre binarie** uguali ad **1** rimangono **spenti**.
Quindi quando il microprocessore esegue questa istruzione, viene **acceso** il solo diodo **DL3**, mentre tutti gli **altri** rimangono **spenti**.

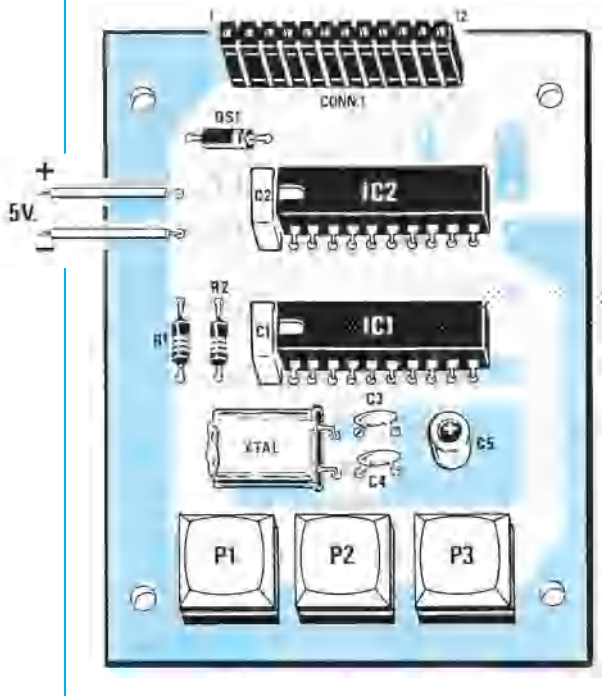


Fig.12 Schema pratico di montaggio della scheda sperimentale LX.1171. L'integrato IC1 è il microprocessore ST62E10 che vi abbiamo fatto programmare con il progetto pubblicato a pag.26. I pulsanti P1 - P2 vi serviranno per modificare le funzioni o la velocità (leggere articolo), mentre il pulsante P3 serve per resettare il circuito.

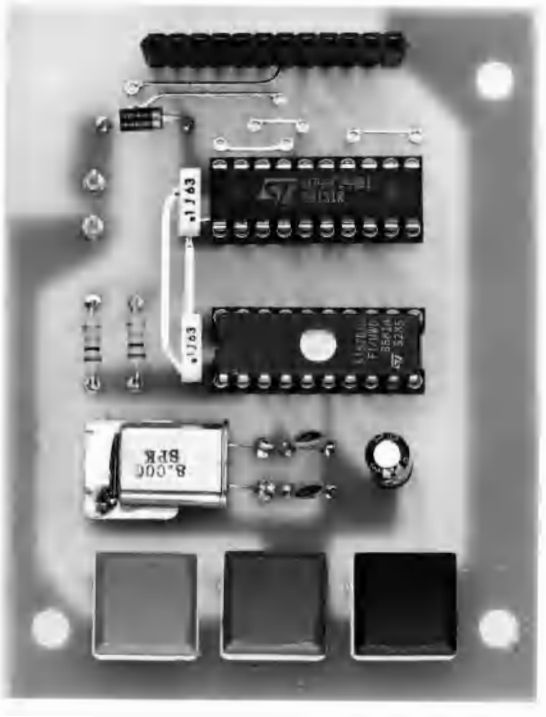


Fig.13 Foto del circuito LX.1171 come si presenta a montaggio ultimato. Si noti nello zoccolo IC1 il microprocessore ST62E10 provvisto della finestra di cancellazione ed in alto il connettore per poter inserire la scheda con i diodi led (vedi fig.14) o con i display (vedi fig.15) Il circuito va alimentato con una tensione di 5 volt.

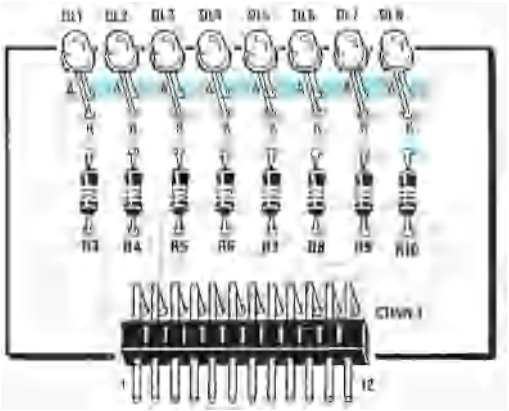


Fig.14 Schema pratico di montaggio della scheda a diodi led da usare se avete memorizzato nell'ST62E10 il programma LED.

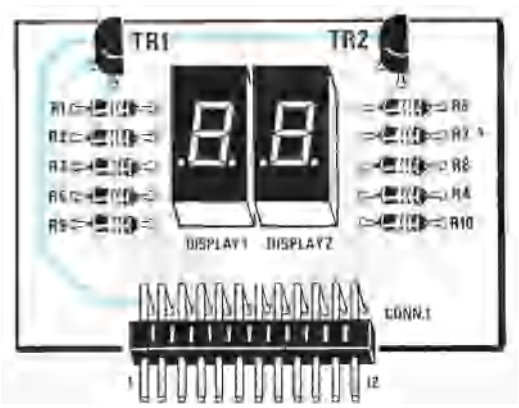


Fig.15 Schema pratico della scheda display da usare se nell'ST62E10 avete memorizzato il programma CONTA oppure LOTTO.

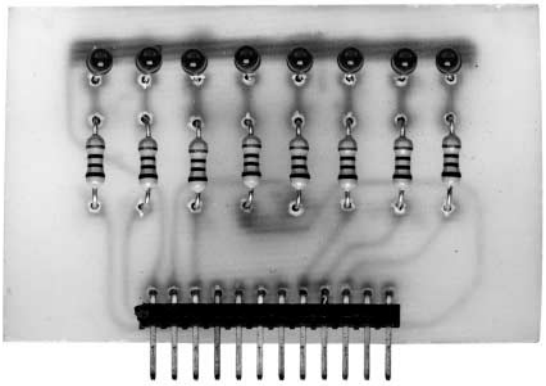


Fig.16 Foto della scheda LX.1171/B dei diodi Led a montaggio ultimato.

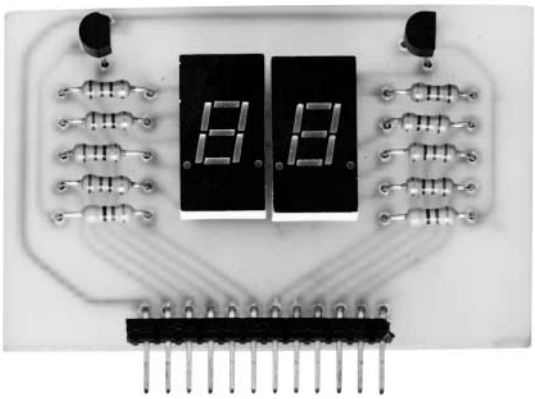


Fig.17 Foto della scheda LX.1171/D dei display a montaggio ultimato.

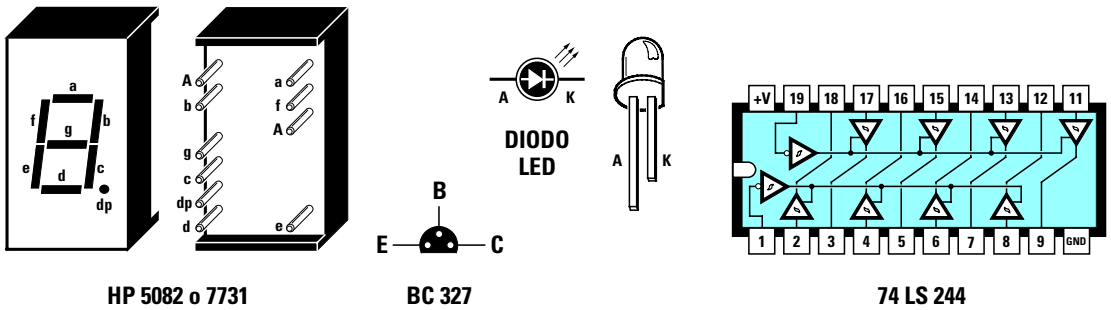


Fig.18 Connessioni dei display viste da dietro, del transistor BC.327 viste da sotto e dell'integrato 74LS244 viste da sopra. Il terminale più lungo presente nei diodi led è l'Anodo.

Se nelle diverse righe di istruzione si scrivono differenti numeri binari, il microprocessore accenderà ogni volta dei diodi led diversi, ed in questo modo potrete creare differenti giochi di luce.

Se ad esempio considerate le istruzioni del lampeggio chiamato **lamp1**, vedete che quando viene eseguita la **prima istruzione** si accende solo il diodo **DL1**, perché solo la cifra più a destra è uno **0**, poi quando viene eseguita la **seconda** istruzione si accende solo **DL2** e così via. In questo modo si è realizzata una semplice accensione in sequenza di un solo diodo alla volta.

Cambiando le cifre **1** e **0** che compongono i vari numeri binari potete realizzare con un po' di fantasia tutti i giochi di lampeggio che vorrete.

Per cambiare le cifre che compongono i numeri binari è sufficiente che vi portiate col cursore sulla cifra che volete modificare, dopodiché potete scrivere **1** o **0**.Per cancellare la cifra binaria che volete sostituire portate il cursore su quella cifra e premete il tasto **Canc**.

Ricordate che ogni numero binario deve essere composto da **non più** di **8 cifre**, altrimenti il programma non funzionerà.
Se scrivete un numero di cifre **inferiori** ad **8** il programma funzionerà ugualmente, ma i diodi corrispondenti alle cifre **non utilizzate** rimarranno sempre **accesi**. Ad esempio, scrivendo **11010b**, cioè tralasciando le tre cifre corrispondenti ai diodi **DL6 - DL7 - DL8**, questi diodi rimarranno sempre **accesi**.

Se volete potete cambiare le cifre binarie di tutti i **5 giochi** proposti, quindi potete modificare anche le istruzioni scritte dopo le etichette **lamp2 - lamp3 - lamp4 - lamp5**. Troverete queste scritte scorrendo con il cursore il listato del programma.
I giochi di luce supportati da questo programma sono solo **5**, quindi non aggiungete altre etichette del

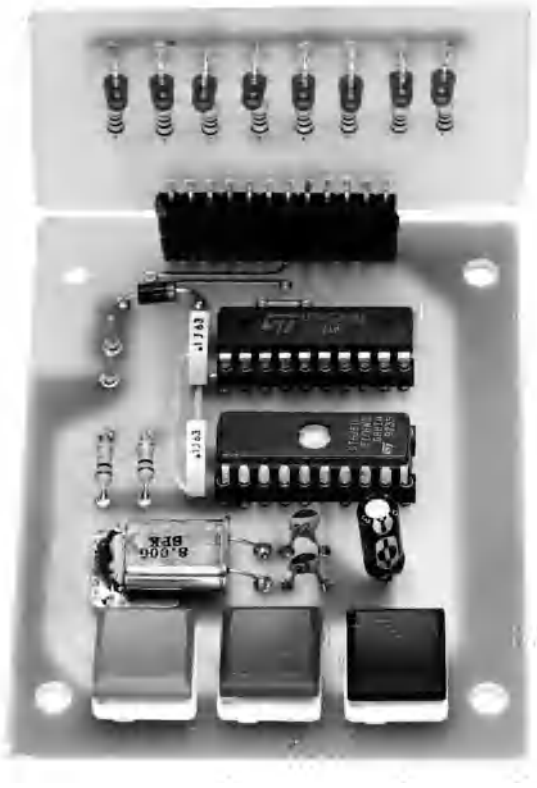


Fig.19 Se avete programmato il microprocessore con il programma LED dovrete inserire nel connettore dell'LX.1171 il connettore maschio della basetta di fig.14.

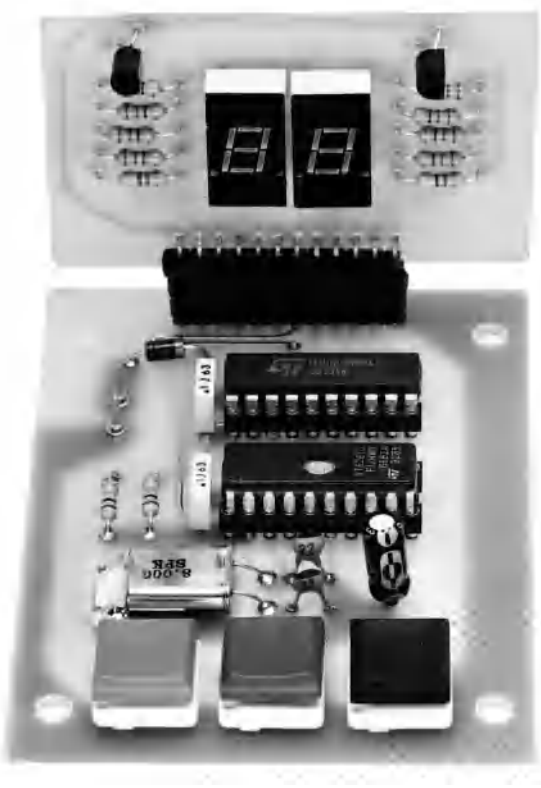


Fig.20 Se avete programmato il microprocessore con il programma CONTA o LOTTO dovrete inserire nel connettore femmina dell'LX.1171 la basetta visibile in fig.15.

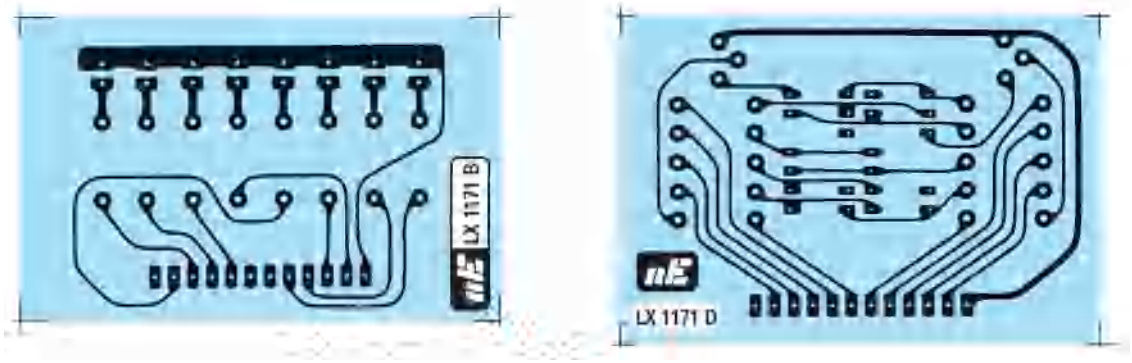


Fig.21 Disegno a grandezza naturale dei due stampati da inserire nella scheda LX.1171 (vedi figg.19-20) visti dal lato rame. Il circuito di sinistra siglato LX.1171/B serve per il programma Led, mentre quello siglato LX.1171/D serve per i programmi Conta e Lotto.

tipo **lamp6 - lamp7** ecc. seguite dalle istruzioni **.byte**, perché il programma non funzionerebbe.

NOTA IMPORTANTE: Come avrete notato, le istruzioni di tipo **.byte** per accendere i led, che seguono le cinque etichette **lamp1 - lamp2 - ecc.**, sono **8**, e così devono sempre rimanere. Se cancellate alcune di queste istruzioni oppure ne aggiungete altre analoghe, il programma **non funzionerà**.

Eseguite le vostre modifiche dovete **salvarle**, altrimenti anche se leggerete sul monitor le istruzioni che avete appena scritto, il programma non risulterà modificato.

Per **salvare** le variazioni basterà **premere** il tasto **F2**: dopo pochi istanti vedrete accendersi la luce dell'Hard-Disk, quindi sarete sicuri che il file modificato è stato aggiornato.

Se **non** desiderate **salvare** le modifiche dovete tenere premuto **Alt** e poi premere **F3**.
Comparirà la finestra di fig.7 dove vi verrà chiesto se volete salvare (tasto **Y**) oppure no (tasto **N**) le modifiche apportate.
Premendo uno qualsiasi di questi tasti tornerete nel menu di fig.3.

Una volta apportate e salvate le modifiche, **prima** di trasferire i dati del programma nelle memorie del microprocessore dovete eseguire un'operazione supplementare, cioè lanciare il programma **assembla**, che serve per convertire le istruzioni del programma in dati che il microprocessore utilizza per eseguire il programma.

Senza uscire dal listato del programma, dopo aver apportato le modifiche ed averle salvate con il tasto **F2**, premete i tasti **Alt+T** e di seguito **A** (vedi fig.6).
In questo modo lo schermo del vostro computer diventerà tutto **nero** e dopo alcuni secondi vedrete apparire questa scritta:

***** SUCCESS *****

che conferma che l'**assemblaggio** è stato completato **senza errori**.
Finita l'operazione di assemblaggio, premendo un tasto qualsiasi tornerete al listato del programma.

Se anziché apparire la scritta ***** SUCCESS ***** compare ad esempio:

ERROR C:\ST62\LED.ASM 256:

significa che nella riga **256** del file **LED.ASM** avete involontariamente inserito un **errore**.

Nota: Un errore molto comune nel quale si può incappare è quello di scrivere un numero binario con un numero di **cifre** superiore ad **8**. Se per esempio scrivete un numero binario a **9 cifre** del tipo:

.byte 110101001b ; Seconda istruzione

dopo aver lanciato il programma **assembla** comparirà il messaggio di errore:

ERROR C:\ST62\LED.ASM 256:
(81) 8-bit value overflow

Il numero tra parentesi **(81)** identifica il tipo di errore e non vi interessa, mentre la scritta **8-bit va-**

lue overflow significa che avete utilizzato un numero binario con più di **8 bit**

Per correggere questo errore dovete tornare al listato del programma e per questo basterà premere un tasto qualsiasi.

Una volta nel listato, poiché l'errore era stato segnalato nella riga **256**, dovete portarvi con il cursore su questa riga e controllare che risulti effettivamente scritto un numero di **8 bit** (**8 cifre**).

Questo **errore** si verificherà raramente perché scrivere un'istruzione così semplice non sarà per voi un problema, comunque lo abbiamo voluto segnalare, perché se un domani dovesse apparire per un vostro programma un **qualunque** messaggio di errore, sappiate che questo è presente nella riga indicata prima dei **due punti** (**:**).

Per **uscire** dal file **LED.ASM** dovete tenere premuto **Alt** e poi premere **F3**. Se non avete ancora salvato la correzione, comparirà la finestra di fig.7, in cui dovrete indicare se volete salvare (tasto **Y**) oppure no (tasto **N**) le modifiche.
Premendo **Y** registrerete queste modifiche, premendo **N** invece **non** le **salverete**.
Facciamo presente che premendo uno qualsiasi di questi tasti, **Y** o **N**, il listato del file **LED** scomparirà e ritornerete nel menu principale.

Il FILE STANDARD.ASM

Come abbiamo già avuto modo di sottolineare, premendo **F3** dal menu principale, oltre ai tre files di tipo **.ASM** contenenti i programmi (–) test per **ST62**, compare un file chiamato **STANDARD.ASM**, che **non** contiene un **programma** vero e proprio.
In questo file trovate l'elenco delle istruzioni che devono comparire sempre in ogni programma, e la cui conoscenza è **basilare** se si desidera realizzare programmi personali.
Sono inoltre presenti tantissimi **commenti**, che vi aiuteranno a capire il significato delle varie istruzioni e tante note utilissime per realizzare programmi per **ST62** senza incorrere negli errori più banali.
Per visualizzare questo file dovete eseguire di nuovo le operazioni spiegate nel paragrafo **Per vedere il listato di un programma**, e quando compare l'elenco dei files dovete premere **Enter**, portare il cursore sul nome **STANDARD.ASM** e premere di nuovo **Enter**.
Questo file risulterà molto utile sia ai più esperti, che vogliono cimentarsi nella realizzazione di programmi senza attendere l'uscita della prossima rivista, sia a chi è alle prime armi, perché potrà iniziare a riconoscere le istruzioni principali che ricorrono in qualsiasi programma per **ST62**.
Probabilmente a molti di voi queste scritte appariranno ancora oscure e prive di significato, quindi non perdete il prossimo numero in cui vi spiegheremo il significato di **tutte** le istruzioni.

Per TORNARE al DOS

Quando avete terminato le operazioni di programmazione potete uscire dal programma e ritornare al **DOS**.

Se vi trovate nel menu di fig.3, basterà tenere premuto **Alt** e poi pigiare **X** e così sul monitor comparirà:

C:\ST6>

a questo punto per uscire dalla directory **ST6** e lanciare altri programmi dovrete digitare:

**C:\ST6>CD ** poi Enter

e così comparirà:

C:\>

Arrivederci al prossimo numero.

COSTO DI REALIZZAZIONE

Costo del kit LX.1171 completo di circuito stampato, dell'integrato 74LS244 (non c'è l'ST62E10 perché inserito nel kit LX.1170) più il quarzo, i pulsanti ed il circuito stampato LX.1171/B (vedi fig.14) con gli 8 diodi led ..... € 12,90

Il kit LX.1171/D con i due display ed i due transistor BC.327 (vedi fig.15)..... € 4,90

Costo del solo stampato LX.1171 ..... € 3,72
Costo del solo stampato LX.1171/B..... € 0,93
Costo del solo stampato LX.1171/D..... € 1,08

I prezzi sopra riportati sono già compresi di IVA, ma non sono incluse le spese postali di spedizione a domicilio.

Come vi abbiamo anticipato nella precedente rivista, per programmare un **ST6**, come del resto un qualunque altro microprocessore, è assolutamente necessario conoscere le **basi** del linguaggio di programmazione, perché senza queste è impossibile scrivere un programma.
Tanto per fare un esempio, se vi proponessimo di progettare un amplificatore utilizzando un **integrato operazionale**, senza precisare come si collega il piedino **invertente** o quello **non invertente** o quali modifiche vanno apportate per alimentare il circuito con una tensione **singola** anziché **duale**, incontrereste parecchie difficoltà nella sua realizzazione.

IMPARARE a programmare i

Evidenziamo questo perché non vogliamo comportarci come tanti altri, che spiegano poco o niente ed illudono i loro lettori sostenendo che non c'è nulla di più facile che programmare un ST6.
Prima di insegnarvi a programmare sarà quindi utile spiegare, anche solo a grandi linee, cosa sono un **registro** e una **subroutine**, il significato di tutte le istruzioni, quali **jp - jrr - jrs - ld - cp** ecc., e come si utilizza una **memoria**.
A questo proposito vogliamo sottolineare che non è sufficiente imparare a memoria il significato di tutte le istruzioni, ma occorre anche sapere **come - dove - quando** utilizzarle, se si vuole che il programma funzioni correttamente.
Non illudetevi pensando di diventare esperti programmatori in pochi giorni, perché andreste incontro ad una delusione: sono necessari infatti alcuni mesi di pratica per acquisire una sufficiente padronanza della materia.
Per accelerare l'apprendimento vi suggeriamo di iniziare a leggere i programmi già funzionanti, perché studiare le soluzioni adottate per scrivere una subroutine, utilizzare un interrupt, o ancora vedere come si sfrutta la memoria, vi sarà di valido aiuto.
In altre parole, anche se sapete scrivere una lettera o una cartolina, non è detto che siate capaci di scrivere un libro giallo o un bel romanzo di avventura, quindi se decidete di dedicarvi all'attività di scrittore, dovete prima acquisire un po' di esperienza facendovi seguire da un letterato o leggendo testi d'autore per apprendere come impostare i vari capitoli.
Chi sa già programmare in **Basic - Fortran - Pascal - C** è avvantaggiato rispetto a chi inizia da "zero", anche se come vedrete, tutti i microprocessori **ST6** utilizzano un **linguaggio assembler** molto semplificato.

PER SCRIVERE un PROGRAMMA

Prima di scrivere qualsiasi programma è necessario sapere quali operazioni deve eseguire il microprocessore, perché in funzione della memoria occupata, dovrete scegliere il micro più idoneo.
Infatti se avete un programma che non supera i **2K**, potete utilizzare un **ST62E10**, se invece avete un programma che occupa più di **2K** e non supera i **4K**, dovete necessariamente adoperare un **ST62E20**.
Nelle **Tabelle N.1-2** riportiamo per ogni microprocessore la **memoria** disponibile ed il massimo numero di **ingressi/uscite** utilizzabili, cioè il numero di piedini che potete adoperare per i segnali.

TABELLA N.1 micro NON CANCELLABILI

Sigla Micro	memoria utile	Ram utile	zoccolo piedini	piedini utili per i segnali
ST62T.10	2 K	64 byte	20 pin	12
ST62T.15	2 K	64 byte	28 pin	20
ST62T.20	4 K	64 byte	20 pin	12
ST62T.25	4 K	64 byte	28 pin	20

TABELLA N.2 micro CANCELLABILI

Sigla Micro	memoria utile	Ram utile	zoccolo piedini	piedini utili per i segnali
ST62E.10	2 K	64 byte	20 pin	12
ST62E.15	2 K	64 byte	28 pin	20
ST62E.20	4 K	64 byte	20 pin	12
ST62E.25	4 K	64 byte	28 pin	20

Quando scrivete un programma, dovete inserire tutte le istruzioni nella sequenza in cui volete che siano eseguite dal microprocessore.
Ad esempio, se voleste programmare un **robot** per

MICROPROCESSORI ST6

Dopo avervi presentato sul N.172/173 un programmatore per microprocessori ST6 ed un circuito per i test, sarete curiosi di conoscere le procedure per scrivere i vostri programmi, ad esempio per realizzare un orologio, per pilotare dei display alfanumerici LCD, per realizzare generatori d'impulsi ecc. Se ci seguirete, cercheremo di spiegarvi con facili esempi tutte le istruzioni necessarie per scrivere i programmi per l'ST6.

cuocere degli spaghetti, dovreste fornirgli nell'ordine queste istruzioni:

- Prendi una pentola
- Riempila per metà di acqua
- Metti il tutto sul fornello
- Accendi il gas sotto la pentola
- Attendi che l'acqua bolla
- Versaci un po' di sale
- Immergi gli spaghetti nell'acqua
- Attendi 5-6 minuti
- Spegni il fornello
- Togli la pentola dal fornello
- Scola la pasta

Se vi dimenticate anche una sola di queste istruzioni, quale ad esempio quella di **accendere il fornello**, non riuscirete mai a cuocere gli spaghetti.
E così se vi dimenticate di inserire l'istruzione **riempi la pentola con acqua**, non potrete mai arrivare alla condizione di vedere l'acqua **bollire**.

Dunque prima di apprestarvi a scrivere un programma dovete sapere:

Come aprire un file per il programma
Come scrivere le istruzioni richieste
Come impostare il programma
Come utilizzare la memoria

COME CREARE un FILE SORGENTE

Chi ha già richiesto il kit **LX.1170** del **Programmatore per ST6** pubblicato sulla rivista **N.172/173** avrà ricevuto un **dischetto floppy**, che oltre a servire per trasferire un programma dall'Hard-Disk nella memoria di un microprocessore **ST6**, serve per creare i **files sorgenti** necessari per scrivere qualsiasi vostro programma.
Infatti in questo dischetto è stato memorizzato un ottimo **editor**, corredato di tantissime **opzioni** che vi saranno utili per scrivere le istruzioni, per duplicarle, per cancellarle ed anche per salvare i files modificati; nel floppy è stato inoltre incluso un **assemblatore**.
Nella rivista precedente (se non ne siete in possesso potete richiedercela, perché abbiamo ancora delle copie disponibili), vi abbiamo spiegato co-

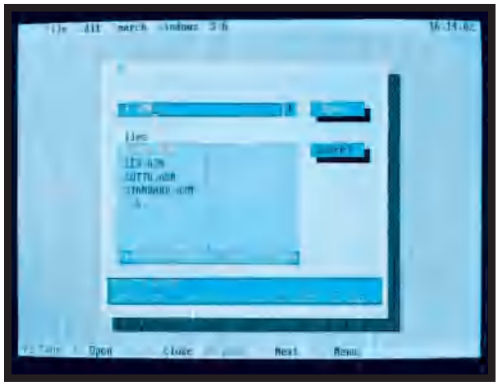


Fig.1 Per aprire un nuovo file dovete richiamare il menù principale e a questo punto se premete il tasto F3 appariranno tutti i nomi dei files del programma ST6 con estensione .ASM.

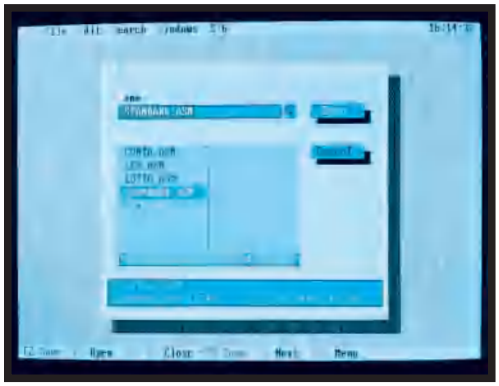


Fig.2 Dalla fig.1 premete Enter, poi portate il cursore sulla riga STANDARD.ASM e premete nuovamente Enter. Aprirete così la SORGENTE STANDARD per scrivere un nuovo programma.

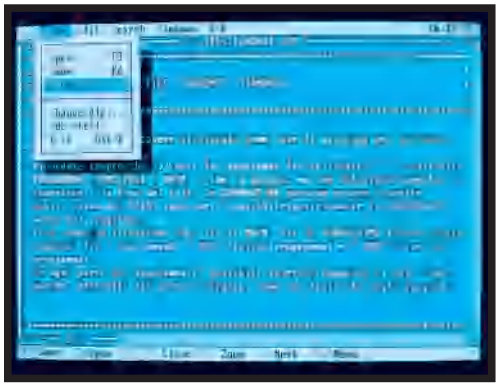


Fig.3 Per ricopiare la SORGENTE STANDARD dovete assegnarle un altro nome. Per fare questo dovete premere i tasti ALT+F poi premere il tasto A. Da questa figura si passerà alla fig.4.

me caricare il **programma ST6** nel vostro computer.

Per richiamare il programma dovete semplicemente digitare:

C:\ST6>ST6 poi Enter

e così compare sul monitor il menu principale.

A questo punto dovete **aprire** un **nuovo file**, nel quale scrivere il vostro programma.

Per aiutarvi fin da queste prime fasi, abbiamo inserito nel dischetto floppy il file **STANDARD.ASM**, che racchiude l'elenco delle istruzioni che devono necessariamente comparire in **ogni** programma.
Copiando il file **STANDARD** con un altro nome, avrete subito a disposizione la struttura base per scrivere il vostro programma.

Quindi per aprire un nuovo **file** dovete procedere come segue:

1° - Quando appare il menu principale premete il tasto funzione **F3** (vedi fig.1), poi premete Enter e portate il cursore sul nome **STANDARD.ASM** (vedi fig.2), quindi premete ancora Enter.

2° - Prima di qualsiasi altra cosa, dovete salvare questo file con un altro nome, quindi premete i tasti **Alt+F** e di seguito selezionate l'opzione **Save as** (che significa "salva con nome") premendo la lettera **A** (vedi fig.3).

3° - Sul monitor appare una finestra nella quale dovete digitare, oltre a **C:\ST6**, il nome del vostro **programma** (vedi fig.4).
Questo **nome**, che vi servirà quando vorrete trasferire il programma dall'Hard-Disk al microprocessore **ST6**, non deve mai superare gli **8 caratteri**.
Dopo il nome non dovete dimenticarvi di aggiungere l'estensione **.ASM**, che sta ad indicare che si tratta di un programma in **assembler**.
Cercate un nome che abbia una logica attinenza col programma che scriverete, per poterlo poi facilmente riconoscere tra gli altri. Per esempio potreste chiamare i programmi:

LED.ASM
LOTTO.ASM
OROLOGIO.ASM
TIMER.ASM ecc.

4° - Dopo aver scritto il nome per esteso (ad esempio, **C:\ST6\TIMER.ASM**) premete Enter, ed in

alto, nella pagina dell'editor visibile in fig.5, vedrete apparire la scritta:

C:\ST6\TIMER.ASM

che vi conferma che il file chiamato **TIMER.ASM** è stato creato.

COME ASSEMBLARE un PROGRAMMA

Importante: Quando avrete terminato di scrivere il programma, come più avanti vi spiegheremo, e l'avrete controllato apportando le modifiche necessarie, dovrete **assemblarlo**, altrimenti non potrete memorizzarlo nel microprocessore.
Per questo motivo, prima di chiudere l'editor, cioè il file del programma, premete i tasti **Alt+T** e di seguito il tasto **A** = assembla (vedi fig.6).
Se non avete commesso errori, dopo qualche secondo apparirà sul monitor la scritta **SUCCESS**.
In caso contrario, apparirà un messaggio che vi indicherà il tipo di errore commesso e la riga di istruzione in cui si trova.
Per correggere l'errore dovete tornare all'editor premendo un tasto qualsiasi.
Per trasferire il programma all'interno dell'**ST6**, seguite le istruzioni ampiamente descritte sulla rivista **N.172/173**.

COME si SCRIVE un'ISTRUZIONE

Quando scrivete un programma dovete rispettare alcune semplici regole che ora vi indicheremo, altrimenti quando l'**assemblerete** compariranno dei messaggi relativi agli **errori**, che dovrete **correggere** per poter proseguire.
Ogni istruzione deve essere scritta su una diversa riga di programma e deve essere composta da un'**etichetta**, da un'**istruzione** e da un **operando** dell'istruzione.
Ad esempio nella riga di programma:

pippo **ldi** **a,10h**

pippo è l'**etichetta**
ldi l'**istruzione**
a,10h è l'**operando** dell'istruzione

ETICHETTA

L'**etichetta** è un riferimento **non obbligatorio** che deve partire **sempre** dall'estremo sinistro della riga.
Un'etichetta serve come **punto di riferimento** per poter ritornare nuovamente, tramite l'istruzione di

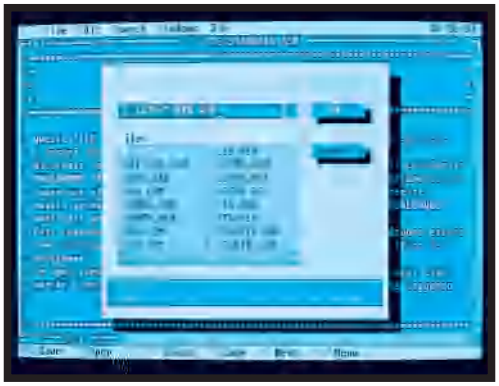


Fig.4 Ammesso che vogliate chiamare il nuovo programma TIMER (ricordate che il nome non può mai superare gli 8 caratteri) scrivete per esteso C:\ST6\TIMER.ASM poi premete Enter.

Fig.5 Una volta creato il file TIMER vedrete apparire nella prima riga in alto della finestra blu dell'editor C:\ST6\TIMER.ASM. A questo punto potete iniziare a scrivere il vostro nuovo programma.

Fig.6 Una volta completato il programma prima di memorizzarlo nell'ST6 lo dovete Assemblare. Premete quindi i tasti ALT+T poi il tasto A. Se avete commesso un errore nel programma, vi verrà segnalato.

salto (indicata con la sigla **jp**, abbreviazione di **jump**), su quella riga di programma.
Quando scrivete un'**etichetta** dovete rispettare queste regole:

La parola non deve superare gli 8 caratteri

Potete scrivere **nota** - **PIPPO** - **RIF** ecc., ma non **parole** che superino gli **8** caratteri.

L'etichetta non deve iniziare con un numero

Non potete scrivere **1nota** - **2nota** - **1PIPPO** - **2PIPPO** ecc., ma potete posizionare un **numero** dopo la parola, senza interporre **spazi** tra parola e numero. E' quindi **corretto** scrivere **nota1** - **PIPPO1** ecc.

Non si può usare lo stesso nome due volte

E' scorretto definire più etichette con lo stesso nome, cioè scrivere **PIPPO - PIPPO**. Potete usare la stessa parola solo se inserite un numero progressivo, ad esempio **PIPPO1 - PIPPO2 - PIPPO3** ecc.

Non si può lasciare alcuno spazio a sinistra dell'etichetta

Se premete **spazio**, poi scrivete:

```
 PIPPO
```

commettete un **errore**, quindi dovete partire da sinistra senza spazio:

```
PIPPO
```

Carattere dell'etichetta

Anche se è possibile scrivere l'etichetta sia in **minuscolo** sia in **maiuscolo**, vi consigliamo di scrivere sempre in **minuscolo** così non vi sbaglierete mai.
Quindi scrivete **pippo1 - pippo2 - nota - rif** ecc.

ISTRUZIONE

Le istruzioni da inserire dopo l'etichetta sono proprie dell'**assembler** degli **ST6**.
Queste istruzioni devono sempre essere scritte dopo aver lasciato **uno spazio**.

Se è già presente il nome dell'**etichetta**, dovete comunque separare l'istruzione con **uno spazio**.
Quindi se avete l'etichetta **pippo1** e l'istruzione **ldi x,10**, dovete scrivere:

```
pippo1  ldi  x,10
```

Se nella riga di istruzione manca il nome dell'etichetta, dovete comunque lasciare **uno spazio** :

```
        ldi  x,10
```

Data l'importanza di scrivere correttamente l'**istruzione**, vi consigliamo di utilizzare la funzione **tabulazione** premendo il tasto **TAB** prima di scrivere qualsiasi **istruzione**.
In questo modo avrete tutte le istruzioni perfettamente incolonnate ed il programma risulterà più comprensibile quando dovrete rileggerlo.

OPERANDO

Nella riga riportata precedentemente, cioè:

```
    ldi  x,10
```

ldi è l'**istruzione**, che significa **carica**
x,10 è l'**operando**

Questa riga indica: **carica** nella **cella di memoria X** il numero **10**.

L'**operando** deve sempre essere separato dall'**istruzione** tramite **uno spazio**, quindi se scrivete:

```
ldix,10
```

commette un grosso errore, mentre se scrivete:

```
ldi  x,10
```

l'intera istruzione è corretta.

Dopo l'**istruzione** e l'**operando** potete inserire, se lo ritenete opportuno, un **commento**.

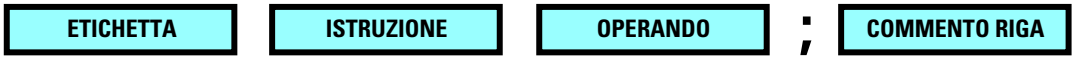


Fig.7 Dovrete sempre ricordare che ogni "riga" di programma è composta da quattro blocchi principali: Etichetta - Istruzione - Operando - Commento. I primi tre blocchi andranno tenuti separati tra loro da uno o più SPAZI (o ancora meglio usate il tasto TAB della tastiera), mentre l'ultimo blocco del COMMENTO dovrà essere separato da un punto e virgola. Se non userete l'ETICHETTA dovrete comunque lasciare uno o più spazi, mentre se non scriverete il COMMENTO non dovrete mettere il punto e virgola.

Questo deve essere sempre preceduto da un **punto e virgola** (**;**), diversamente il computer segnalerà **errore**.
Utilizzando l'istruzione precedente potete scrivere:

```
    ldi   x,10 ; inserire 10 in x
```

I commenti possono essere scritti anche all'inizio di una riga, ma senza lasciare spazi e ricordando di mettere sempre prima un **punto e virgola** (**;**).

Ogni volta che completate un'istruzione dovete sempre e necessariamente andare a capo premendo il tasto **Enter**.
Tenete presente che le **istruzioni** possono essere scritte sia in **minuscolo** sia in **maiuscolo**:

```
ldi   x,10   oppure   LDI   X,10
```

COME scrivere i NUMERI

Nell'esempio sopra riportato noi abbiamo scritto **x,10**, in altre parole abbiamo utilizzato un numero **decimale**.
Tuttavia può risultare più **vantaggioso** in alcune istruzioni scrivere i numeri in base **esadecimale - ottale - binaria**.
Per far capire al **computer** che tipo di numero avete inserito, dovete scrivere una **lettera** come qui sotto specificato:

o oppure **O** se il numero è **ottale**
h oppure **H** se il numero è **esadecimale**
b oppure **B** se il numero è **binario**

Ad esempio:

10o = numero **ottale**
01Ah = numero **esadecimale**
00100101b = numero **binario**

Se dopo il numero **non mettete** nessuna lettera, il computer considererà questo numero **decimale**.
Quando scrivete un numero **esadecimale**, dovete sempre mettere **davanti** ad ogni numero uno **0** (**zero**), quindi **01A - 0ED - 0AC** ecc., ed alla fine deve seguire la lettera **H**, per indicare che il numero è **esadecimale**, quindi i numeri sopra riportati vanno scritti 01A**H** - 0ED**H** - 0AC**H**.

I **numeri decimali**
iniziano da **0** e terminano a **255**

I **numeri ottali**
iniziano da **0** e terminano a **377**

I **numeri esadecimali**
iniziano da **0** e terminano a **FF**

I **numero binari**
iniziano da **0** e terminano a **11111111**

STRUTTURA di un PROGRAMMA

Per scrivere un programma per **ST6** si devono seguire delle precise regole che sono:

Definire lo spazio in MEMORIA
Definire le VARIABILI
Definire i REGISTRI
Scrivere il PROGRAMMA PRINCIPALE
Scrivere le SUBROUTINE
Scrivere eventuali subroutine di INTERRUPT
Definire i VETTORI di INTERRUPT

Per facilitarvi, abbiamo inserito nel **dischetto floppy**, che avete ricevuto assieme al **programmatore LX.1170** (vedi rivista N.172/173), un **file** chiamato **STANDARD.ASM** che vi spiega come impostare il programma, dove scrivere le varie istruzioni, come definire lo spazio di memoria ed i registri, dove posizionare le subroutine, come inizializzare l'**ST6**, insomma tutti i consigli e le informazioni necessarie per non sbagliare.
Come abbiamo già descritto nel paragrafo "Come creare un file sorgente", tutte le volte che dovrete scrivere un **nuovo** programma **copiate** il file **STANDARD.ASM** con il **nome** del programma che volete scrivere e tutto risulterà più facile.

La MEMORIA dell'ST6

All'interno dei microprocessori tipo **ST62E10 - ST62T10 - ST62E15 - ST62T15** risultano disponibili per il programma **2K** di **memoria ROM**, mentre nei microprocessori **ST62E20 - ST62T20 - ST62E25 - ST62T25** sono disponibili **4K** di **memoria ROM**.
All'interno di ciascun microprocessore sono presenti anche **64 byte** di **memoria RAM** che servono per i **registri** e le **variabili**.
La memoria **ROM** mantiene tutte le informazioni, cioè il programma, inserite durante la programmazione del microprocessore anche in assenza di alimentazione.
La memoria **RAM** viene usata per le variabili, cioè per i dati che devono essere di volta in volta letti, scritti e modificati, e quindi può essere "aggiornata" dallo stesso microprocessore durante il funzionamento del programma.

La **memoria** (sia ROM sia RAM) può essere considerata come un insieme di piccole **celle** ed all'interno di ognuna può essere inserito un solo **dato**. Per portarvi un esempio pratico, potete paragonare queste **celle** a quelle presenti in un **favo** per **api**. Quando le api hanno riempito una cella, passano a riempire la seconda, poi la terza ecc. fino a riempire tutto il **favo**.
In un microprocessore da **2K** ci sono esattamente **1.828 celle** in grado di contenere un programma composto da circa **900 - 990 righe** di **programma**.
In un microprocessore da **4K** ci sono esattamente **3.872 celle** in grado di contenere un programma composto da circa **1.800 - 2.000 righe** di **programma**.

DEFINIZIONE delle VARIABILI

Innanzitutto la **variabile** è un numero, **decimale - ottale - binario** oppure **esadecimale**, che il microprocessore può modificare tramite una particolare **istruzione**.
Le **variabili** non vengono inserite nella **memoria ROM**, ma sempre e solo nella **memoria RAM**; in questo modo è possibile variare questi numeri secondo le diverse esigenze.
Ad esempio, per realizzare un **orologio** servono **3 variabili**, una per le **ore**, una per i **minuti** ed una per i **secondi**, che tramite opportune e precise istruzioni, si possono incrementare per ottenere le seguenti funzioni.
La **prima variabile** dei **secondi** viene aumentata dal programma di **1** per ogni **secondo** trascorso.
Raggiunto il numero **60**, il programma aumenta di **1** la **seconda variabile** dei **minuti** e porta a **00** la **prima variabile** dei **secondi**.
Quando la **variabile** dei **minuti** raggiunge il numero **60**, il programma aumenta di **1** la **terza variabile** delle **ore** e porta a **00** le variabili dei **minuti** e dei **secondi**.
Quando la **variabile** delle **ore** raggiunge il numero **24**, il programma porta a **00** le tre variabili **ore - minuti - secondi**.

Quando inserite una variabile dovete ricordare che se dopo il numero **non mettete** nessuna lettera, il computer lo considera un numero **decimale**.
Per informare il computer che avete inserito un numero con base diversa da 10, seguite le istruzioni spiegate nel paragrafo "Come scrivere i numeri".

In ogni programma potete inserire un **massimo** di **60 variabili** e poiché queste sono situate nelle celle della **memoria RAM**, dovete dare ad ogni **cella** un numero di **riferimento**, così da poter ritrovare la variabile.

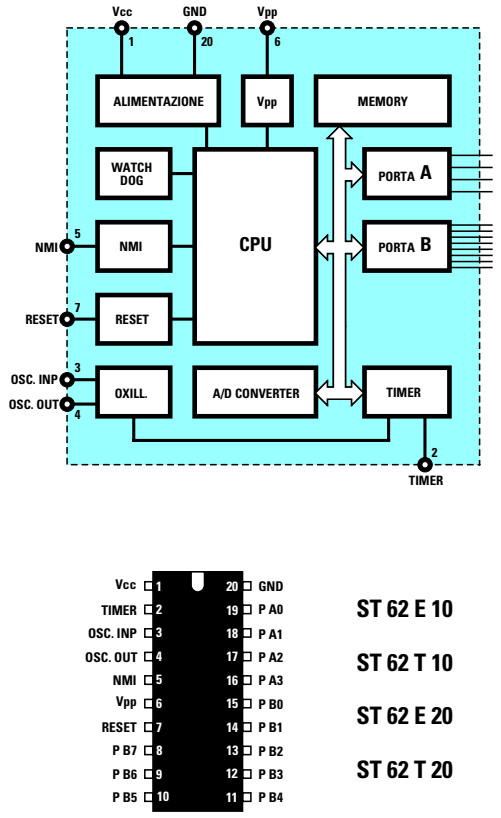
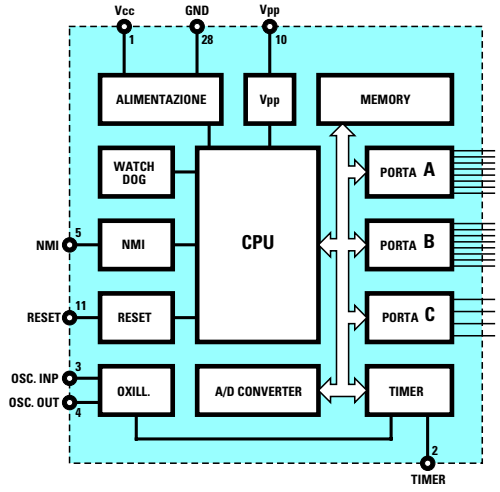
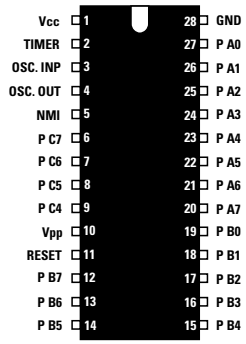


Fig.8 Nei microprocessori ST6 tipo E10-T10 ed E20-T20 sono disponibili due sole porte, la A e la B. La porta A è siglata A0-A1-A2-A3 mentre la porta B è siglata B0-B1-B2-B3-B4-B5-B6-B7. Tutti questi piedini possono essere utilizzati sia come Entrate sia come Uscite. Nei prossimi numeri vi spiegheremo come predisporli.

Questo numero di riferimento si chiama **indirizzo** e poiché ogni variabile occupa una sola **cella** di memoria, queste hanno un numero progressivo.
Se usate i numeri in base **dieci**, la prima cella porta il numero **132** e l'ultima il numero **191**.
Se usate i numeri in base **sedici**, la prima cella porta il numero **084H** e l'ultima il numero **0BFH**.

Ricordate che il **nome** che assegnate alla **variabile** deve sempre essere scritto partendo da **sinistra**, senza lasciare **nessuno spazio**, ed a questo nome deve seguire, spaziandola, la dicitura **.def**. Dopo questa abbreviazione, dovete lasciare un altro





ST 62 E 15

ST 62 T 15

ST 62 E 25

ST 62 T 25

Fig.9 Nei microprocessori ST6 tipo E15-T15 ed E25-T25 sono disponibili tre porte, la A, la B e la C. La porta A è siglata A0-A1-A2-A3-A4-A5-A6-A7, la porta B è siglata B0-B1-B2-B3-B4-B5-B6-B7, mentre la porta C è siglata C4-C5-C6-C7. Anche tutti questi piedini possono essere utilizzati sia come Entrate sia come Uscite.

spazio e poi scrivere il **numero**, che rappresenta l'**indirizzo** della cella di memoria in cui volete allocare questo dato.

Ricordatevi inoltre che se assegnate la stessa cella di memoria a due **diverse variabili**, il microprocessore **non segnalerà** nessun **errore**, ma in questo caso il programma funzionerà in modo **anomalo** e voi non otterrete le funzioni che vi eravate prefissati.

Per il programma che serve a far funzionare un orologio, potete definire nel seguente modo le **3 variabili**:

```
secondi   .def 132
minuti    .def 133
ore       .def 134
```

Potete definire queste **variabili** anche con un numero **esadecimale**, senza che ciò modifichi il funzionamento dell'orologio:

```
secondi   .def 084H
minuti    .def 085H
ore       .def 086H
```

Come ultima indicazione, tenete presente che le **variabili** vanno definite fin dal principio, vanno cioè inserite all'**inizio** del programma e non a metà o alla fine.
Vi consigliamo di scrivere il nome delle **variabili** sempre in **minuscolo**.

I REGISTRI del MICROPROCESSORE

Nella **memoria RAM** del microprocessore **ST6**, oltre allo spazio riservato alle **variabili** in precedenza descritte, sono presenti delle altre **celle di memoria** chiamate **registri**, che permettono di eseguire precise funzioni già definite.
Ad esempio, c'è un **registro** che permette di definire quali piedini delle porte **A - B - C** vanno utilizzati come **ingressi** e quali come **uscite** (vedi figg. 8-9).
Questi **indirizzi** devono essere definiti sempre all'inizio di ogni programma. Questa è un'operazione che **non dovete** eseguire se utilizzate, come abbiamo spiegato, il file **STANDARD.ASM**, perché abbiamo già **definito** noi tutti i registri, così da evitarvi errori.

IL REGISTRO ACCUMULATORE

Nel microprocessore **ST6** c'è un particolare **registro** chiamato **accumulatore** ed indicato sempre con la lettera **a**, molto importante, perché esegue le seguenti **operazioni matematiche**:

fa la somma
fa la sottrazione
fa la tavola della verità di un AND
fa una comparazione tra due numeri
fa il complemento di un numero

Tutte queste operazioni **matematiche** si possono eseguire **solo** con il numero che avete provveduto ad inserire nell'**accumulatore;** il **risultato** ottenuto subentra poi automaticamente a sostituire il numero prima presente nel **registro accumulatore**.

Ritorniamo all'esempio dell'**orologio** e supponiamo che siano le **10:25:30**.
Poiché la funzione dell'orologio presuppone che si debba sempre **sommare 1**, tramite l'istruzione **ld** (load) spostiamo nell'accumulatore **a** il numero presente nella **variabile** dei **secondi**, cioè **30**. A questo punto possiamo sommare a questo il numero **1,** ottenendo **31**.
Ora sempre con l'istruzione **ld**, spostiamo nuovamente il risultato dal registro accumulatore alla **variabile** dei **secondi**, che di conseguenza risulta ora **31**.
Trascorso un secondo, si ripete il ciclo di istruzioni: spostiamo il numero dalla **variabile** dei **secondi** nel registro **accumulatore**, **sommiamo** a questo **1** ed il nuovo risultato, **32**, lo spostiamo nuovamente nella **variabile** dei **secondi**.

L'intero ciclo appena descritto si riduce a tre istruzioni:

```
      ld      a,secondi   ; carica i secondi in a
      addi    a,1         ; somma ad a il numero 1
      ld      secondi,a   ; carica somma nella variabile
```

REGISTRI SPECIALIZZATI

All'interno del microprocessore **ST6** ci sono dei registri **specializzati** che noi abbiamo definito nel nostro file **STANDARD.ASM** con le lettere **x - y - v - w** e che vi potrebbero servire per semplificare particolari operazioni.
Ad esempio se voleste ottenere un **impulso** della durata di **1 millisecondo** potreste eseguire queste istruzioni:

```
        set     0,port_b    ; metti a 1 il piedino PB0
        ldi     x,103       ; assegna 103 a x
ripeti  dec     x           ; sottrai 1 a x
        jrnz    ripeti      ; ripeti se x non è a 0
        res     0,port_b    ; metti a 0 il piedino PB0
```

Questa sequenza di istruzioni fa sì che, iniziando dal numero **103** e continuando a sottrargli **1** fino a quando non si è raggiunto lo **0**, passi esattamente **1 millisecondo**.

Il SET di ISTRUZIONI

Le istruzioni del linguaggio **assembler** usate dal microprocessore **ST6** sono molto semplici e possono essere così suddivise:

1° CARICAMENTO DATI

LD - spostamento di **dati** tra due registri
LDI - caricamento di un **numero** in un registro

2° ARITMETICHE

ADD - somma tra **variabile** e **accumulatore**
ADDI - somma tra **numero** e **accumulatore**
AND - funzione **And** tra **variabile** e **accumulatore**
ANDI - funzione **And** tra **numero** e **accumulatore**
CLR - azzera una **variabile**
COM - complementa i **bit** nell'**accumulatore**
CP - comparazione tra **variabile** e **accumulatore**
CPI - comparazione tra **numero** e **accumulatore**
DEC - sottrae **1** ad una **variabile**
INC - somma **1** ad una **variabile**
RLC - sposta una **variabile** a sinistra con **riporto**
SLA - sposta una **variabile** a sinistra senza **riporto**
SUB - sottrazione tra **variabile** e **accumulatore**
SUBI - sottrazione tra **accumulatore** e **numero**

3° SALTI CONDIZIONATI sull'ETICHETTA

JRC - salta se c'è un **riporto**
JRNC - salta se **non** c'è un **riporto**
JRZ - salta se l'operazione dà **0**
JRNZ - salta se l'operazione **non** dà **0**
JRR - salta se un **bit** è **0**
JRS - salta se un **bit** è **1**

4° SALTI INCONDIZIONATI sull'ETICHETTA

CALL - esegui una **subroutine**
JP - salta sempre sull'**etichetta**

5° SETTAGGIO dei BIT

SET - metti un **bit** a **1**
RES - metti un **bit** a **0**

6° CONTROLLO CPU

NOP - serve per ottenere dei **ritardi**
RET - ritorna da una **subroutine**
RETI - ritorna da un **interrupt**
STOP - blocca tutte le funzioni del **micro**
WAIT - arresta l'esecuzione del **programma**

TEMPI di ESECUZIONE

I tempi per eseguire un'istruzione si calcolano a **cicli macchina** e vanno da un **minimo** di **2 cicli** ad un **massimo** di **5 cicli macchina**.
Tutte le istruzioni di **Caricamenti Dati - Funzioni Aritmetiche - Salti Incondizionati - Settaggio Bit** impiegano **4 cicli macchina**.
Tutte le istruzioni **Controllo CPU** e le istruzioni **JRC - JRNC - JRZ - JRNZ** dei **Salti Condizionati** impiegano **2 cicli macchina**.
Tutte le istruzioni **JRR - JRS** dei **Salti Condizio-**

nati impiegano **5 cicli macchina**.
Il **tempo** di un **ciclo macchina** dipende dalla frequenza del **quarzo** utilizzato per il **clock**.
Per calcolare il **tempo** di un'istruzione potete usare questa formula:

microsecondi = (13 : MHz quarzo) x N cicli

Ad esempio, se usate un quarzo da **2 MHz** per eseguire un'istruzione **aritmetica** che necessita di **4 cicli**, il microprocessore per svolgerla impiegherà:

(13 : 2) x 4 = 26 microsecondi

Se usate un quarzo da **8 MHz**, la stessa istruzione sarà eseguita in un tempo di:

(13 : 8) x 4 = 6,5 microsecondi

Per il **clock** potete usare dei quarzi di qualsiasi frequenza, da **2,45 - 3,4 - 4,7 - 6,5 - 7 - 8 MHz**. Normalmente si utilizza la frequenza massima di **8 MHz** per rendere più veloce l'esecuzione di un programma.
Non usate mai quarzi **superiori** agli **8 MHz**, perché questa è la frequenza **massima** accettata dall'oscillatore interno del microprocessore **ST6**.

COME usare le varie ISTRUZIONI

Le istruzioni vanno scritte secondo precisi criteri, ed è quindi abbastanza facile che un principiante incontri qualche difficoltà nell'impostarle.
Ripetiamo nuovamente che se non mettete il nome di un'**etichetta** dovete sempre lasciare **uno spazio** prima di scrivere l'istruzione o, ancora meglio, premete il tasto **TAB**, così da avere tutte le istruzioni incolonnate.
Per ognuna delle istruzioni utilizzate dal linguaggio di programmazione Assembler, diamo di seguito una semplice spiegazione correlata da un esempio.

ADD

Per eseguire una **somma** tra il numero presente nella **variabile** ed il numero presente nell'**accumulatore**, dovete scrivere l'istruzione in questo modo:

```
        add     a,secondi
```

Per questa istruzione abbiamo usato come **variabile** il nome **secondi**, ma potevamo utilizzare un nome diverso, a patto che fosse stato sempre di **8 caratteri**, come ad esempio **gradi - metri - litri** ecc.
Se nell'**accumulatore** è presente il numero **22** e nella **variabile secondi** è presente il numero **15**, dopo questa istruzione nell'accumulatore **a** è presente il numero **37**, perché **22 + 15 = 37**.

ADDI

Questa istruzione è identica alla precedente con la sola differenza che il **numero** da **sommare** a quello presente nel registro **accumulatore** non è preso dalla variabile, ma immesso direttamente da voi.
Ad esempio, se al numero presente nell'**accumulatore**, che potrebbe essere **37**, volete sommare il **numero 30**, dovrete scrivere:

```
        addi    a,30
```

Dopo questa istruzione nell'accumulatore **a** è presente il numero **30 + 37 = 67**.

AND

Questa istruzione permette di eseguire un'operazione **AND** tra il numero contenuto nell'**accumulatore** e quello nella **variabile**.
Per farvi comprendere meglio come viene effettuata questa operazione, riportiamo la **tavola della verità** con i numeri **binari**.

Tavola della verità

accumulatore	**0 0 1 1**
variabile	**0 1 0 1**
risultato	**0 0 0 1**

Secondo questa tavola, quando è presente un **valore logico 1** sia nell'**accumulatore** sia nella **variabile**, si ha come risultato **1**; in ogni altra condizione si ha sempre come risultato **0**.
L'istruzione va scritta così:

```
        and     a,secondi
```

Se nella **variabile secondi** è presente il numero decimale **30**, che convertito in **binario** è uguale a **00011110**, e nell'**accumulatore** è presente il numero decimale **25**, che convertito in **binario** è uguale a **00011001**, il risultato dell'operazione **AND** tra questi numeri è:

```
00011110
00011001
--------
00011000  risultato
```

che corrisponde al numero **decimale 24**.

ANDI

A differenza della precedente, questa istruzione esegue un'operazione **AND** tra il numero contenuto nell'**accumulatore** ed un **numero binario** scritto direttamente da voi sulla stessa riga dell'istruzione.
Quando nell'**accumulatore** e nel **numero** inserito è presente un valore **logico 1** si ha come risultato **1**; in ogni altra condizione si ha sempre come risultato **0**.
Ammesso di avere nell'**accumulatore** il numero **binario 00011110** e di voler eseguire l'operazione **AND** con il **numero binario 11111001**, l'istruzione va scritta:

```
        andi    a,11111001B
```

Come avrete notato, alla fine di questo numero abbiamo messo una **B** affinché il **computer** possa riconoscere che il numero è **binario**.
Il risultato di questa operazione è:

```
00011110
11111001
--------
00011000  risultato
```

Le due funzioni **AND** e **ANDI** possono essere utilizzate per modificare il livello logico sugli **otto** piedini di uscita di una **porta** del **microprocessore**.
Sapendo su quali di questi piedini è presente un **livello logico 0** e su quali è presente un **livello logico 1**, è possibile accendere ad esempio dei diodi led.
Nel nostro esempio, poiché il risultato è **00011000**, saranno **spenti** i primi tre diodi led, **accesi** i successivi due e **spenti** gli ultimi tre.
Per portare il **risultato** dell'istruzione **ANDI** sulla **porta** di uscita **B**, bisogna scrivere questa istruzione:

```
        ld      port_b,a
```

che in pratica significa: **carica** (**ld**) sulla **porta B** il risultato contenuto nell'**accumulatore a**.

CALL

Questa istruzione viene adoperata quando si vuole far eseguire al microprocessore una **subroutine**, cioè un parte di programma identificata con un'**etichetta**.
Le **subroutine** sono utili per eseguire **più volte** lo stesso set di **istruzioni**.

Ad esempio potrebbe verificarsi di dover ripetere più volte la seguente funzione di **ritardo**.

```
ritardo ldi     x,103   ; assegna 103 a x
ripeti  dec     x       ; sottrai 1 a x
        jrnz    ripeti  ; ripeti se x non è a 0
        ret             ; ritorna al programma
```

Tutte le volte che vorrete ripetere queste istruzioni in una parte del programma, basterà scrivere:

```
        call    ritardo ; chiama subroutine ritardo
```

Come potete notare con due sole parole, **call ritardo**, farete ripetere esattamente le stesse istruzioni contrassegnate dall'etichetta **ritardo**, senza bisogno di riscriverle.
In questo modo non solo eviterete di occupare altra memoria nel microprocessore, ma soprattutto non correrete il rischio di compiere qualche **errore** nel riscrivere le istruzioni.
Eseguita la subroutine **ritardo**, il microprocessore proseguirà con le **istruzioni** successive alla riga **call ritardo**, perché alla fine della **subroutine** è presente l'istruzione **ret**, che significa: ritorna al programma nella riga successiva alla quale era scritto **call ritardo**.

CLR

Questa istruzione serve per portare a **0** una **variabile**.
Per spiegarci meglio riconsideriamo il programma per realizzare un orologio.
Per ottenere la funzione **orologio** è necessario che il numero delle **ore** riparta da **zero** quando si è raggiunto il numero **24**; allo stesso modo quando i **minuti** ed i **secondi** hanno raggiunto il numero **60**, devono ripartire a contare da **zero**.
Quando volete che le **variabili** chiamate **ore - minuti - secondi** diventino **0** dovete scrivere:

```
        clr     ore
        clr     minuti
        clr     secondi
```

La funzione **CLR** può essere usata anche per azzerare il registro **accumulatore** scrivendo semplicemente:

```
        clr     a
```

In questo modo **cancellerete** eventuali numeri rimasti nell'**accumulatore** da un'operazione precedente.

COM

Questa funzione serve per **complementare** il **numero binario** presente nell'**accumulatore**.
In altre parole, questa istruzione **inverte** ogni singolo **bit**, quindi dove c'è **0** si ha **1** e dove c'è **1** si ha **0**.
Se nell'**accumulatore** è presente il numero **binario 00011000** scrivendo:

```
        com     a
```

il contenuto dell'**accumulatore** diventerà **11100111**.

CP

Questa istruzione **confronta** il numero contenuto nell'**accumulatore** con quello presente nella **variabile**.
Da questo confronto il microprocessore ricava queste tre sole **condizioni**:

- il numero dell'**accumulatore** è **minore** rispetto a quello della **variabile**.

- il numero dell'**accumulatore** è **uguale** a quello della **variabile**.

- il numero dell'**accumulatore** è **maggiore** rispetto a quello della **variabile**.

Questo confronto è utile quando occorre far compiere dei **salti condizionati** al programma, per eseguire le operazioni che desiderate.

Prendiamo ancora una volta l'esempio dell'orologio.
Quando il numero nella variabile **secondi** giunge a **60**, bisogna ripartire da **0** ed **aumentare** di **1** il numero nella variabile **minuti**.
Quando il numero nella variabile **minuti** giunge a **60** , bisogna ripartire da **0** ed **aumentare** di **1** il numero nella variabile **ore**.
Quando il numero nella variabile **ore** giunge a **24**, bisogna riportare a **0** le variabili **ore - minuti - secondi**.
Il programma per eseguire queste funzioni va scritto nel seguente modo:

```
        inc     secondi     ; incrementa di 1 i secondi
        ldi     a,60        ; carica in a il numero 60
        cp      a,secondi   ; confronta a con variabile sec.
        jrnz    fine        ; salto condizionato a fine
        clr     secondi     ; azzera i secondi
        inc     minuti      ; incrementa di 1 i minuti
        ldi     a,60        ; carica in a il numero 60
        cp      a,minuti    ; confronta a con variabile min.
        jrnz    fine        ; salto condizionato a fine
        clr     minuti      ; azzera i minuti
        inc     ore         ; incrementa di 1 le ore
        ldi     a,24        ; carica in a il numero 24
        cp      a,ore       ; confronta a con variabile ore
        jrnz    fine        ; salto condizionato a fine
        clr     ore         ; azzera le ore
fine                        ; fine dell'incremento
```

Quando il numero dei **secondi** è **diverso** da **60**, numero caricato nell'**accumulatore**, si fa fare al programma un **salto condizionato**, cioè si **salta** alla riga di programma con l'**etichetta** chiamata **fine**.
Quando il numero dei **secondi** è **uguale** a **60** questo **salto** non avviene, quindi il programma passa alla riga successiva **aumentando** di **1** la **variabile** dei **minuti** ed **azzerando** quella dei **secondi**.
Fino a quando la **variabile** dei **minuti** non avrà raggiunto il numero **60**, si fa fare un **salto condizionato** sull'etichetta **fine**.
Quando il numero dei **minuti** è **uguale** a **60** questo **salto** non avviene, quindi il programma passa alla riga successiva **aumentando** di **1** la **variabile** delle **ore** ed **azzerando** quella dei **minuti**.
Il programma **confronta** il numero presente in questa **variabile** con quello presente nell'**accumulatore**, che è **24**, e quando nella **variabile** è presente il numero **24**, il programma passa alla riga successiva, vedi **clr ore**, per azzerare la variabile **ore**.
Se il microprocessore ripete questo programma **ogni secondo**, compiendo nel frattempo altre istruzioni di programma, avrete ottenuto la funzione orologio.

CPI

Questa istruzione si differenzia dalla precedente perché **confronta** il numero contenuto nell'**accumulatore** con un **numero** direttamente scritto da voi.
Da questo confronto il microprocessore ricava sempre queste tre sole **condizioni**:

- il numero dell'**accumulatore** è **minore** rispetto a quello della **variabile**.

- il numero dell'**accumulatore** è **uguale** a quello della **variabile**.

- il numero dell'**accumulatore** è **maggiore** rispetto a quello della **variabile**.

Questo confronto è utile quando occorre far compiere dei **salti condizionati** al programma, per eseguire le operazioni che desiderate.

Ad esempio, se volete realizzare un **termostato** che **disecciti** un **relè** quando la temperatura ha raggiunto i **20 gradi**, un modo per scrivere le istruzioni potrebbe risultare il seguente:

```
      ld    a,gradi    ; carica in a i gradi
      cpi   a,20       ; compara i gradi con numero 20
      jrc   funz       ; salta a funz se minore di 20
      res   7,port_b   ; diseccita il relè
funz                   ; non diseccitare il relè
```

La **temperatura** prelevata da una sonda viene messa nell'**accumulatore** per essere poi **comparata** con il numero **20**.
Se la temperatura è **minore** di **20**, il programma salta all'**etichetta** siglata **funz** e quindi il relè non si diseccita.
Quando la **temperatura** ha raggiunto i **20 gradi**, il microprocessore passa ad eseguire l'istruzione presente nella **quarta** riga, cioè si **resetta,** portando a **livello logico 0** il **piedino 7** della porta **B**. In questo modo il relè collegato su questo piedino si **diseccita**.

DEC

Questa istruzione serve per **decrementare di 1** il numero presente nella **variabile** specificata di seguito.
Ad esempio, se volete che trascorsi **40 minuti** si disecciti un **relè**, dovete scrivere il programma nel seguente modo:

```
      ldi   tempo,40   ; carica in tempo il numero 40
      .....  .......   ;
      dec   tempo      ; decrementa la variabile tempo
      ld    a,tempo    ; carica in a la variabile tempo
      cpi   a,0        ; confronta a con il numero 0
      jrnz  funz       ; se è diverso da 0 salta in funz
      res   7,port_b   ; diseccita relè
funz                   ; non diseccitare il relè
```

Dopo avere caricato il numero **40** nella **variabile tempo**, dovete **completare** il programma (spazio indicato con **puntini**) per far eseguire al programma un decremento ogni **60 secondi**.

INC

Questa istruzione serve per **aumentare di 1** il numero presente nella **variabile** specificata di seguito.
Ritornando all'esempio scritto per l'istruzione **CP**, la prima riga conteneva l'istruzione:

```
      inc   secondi    ; incrementa la variabile secondi
```

quindi il numero presente nella **variabile secondi** viene **aumentato** di **1**.

NOTA importante per le istruzioni DEC e INC

Quando utilizzate queste istruzioni dovete tenere presente quanto segue:
- un ulteriore **decremento** (istruzione **DEC**) quando la **variabile** è arrivata al numero **decimale 0** porta il valore della **variabile** a **255**.
- un ulteriore **incremento** (istruzione **INC**) quando la **variabile** è arrivata al numero **decimale 255** porta il valore della **variabile** a **0**.

JP

Questa istruzione consente di effettuare un **salto incondizionato** in un punto qualsiasi del programma marcato da un'**etichetta**.
Ad esempio, se dovete far **lampeggiare** un **diodo led** con una cadenza di **1 secondo**, dovete scrivere:

```
inizio                 ; etichetta
      com   a          ; se acceso spegni o viceversa
      ld    port_b,a   ; sposta su b quello che c'è in a
      call  ritardo    ; chiama funzione ritardo
      jp    inizio     ; ripeti funzione dall'inizio
```

Vi ricordiamo che **ritardo** è un'**etichetta** (vedi istruzione **CALL**).

JRC

Questa istruzione viene sempre inserita nei programmi **dopo** un'istruzione **CP** o **CPI** per effettuare un salto **condizionato**.
Se dalla **comparazione** il microprocessore rileva che il numero presente nell'**accumulatore** è **minore** di quello presente nella **variabile**, viene effettuato un **salto** sull'**etichetta**.
Nella funzione **CPI**, in cui vi abbiamo presentato un esempio di programma per **termostato**, avete trovato utilizzata l'istruzione **JRC**:

```
      ld    a,gradi    ; carica nell'accumul. i gradi
      cpi   a,20       ; compara i gradi con numero 20
      jrc   funz       ; salta a funz se minore di 20
      res   7,port_b   ; diseccita il relè
funz                   ; non diseccitare il relè
```

In questo caso il **salto** viene effettuato sull'**etichetta** siglata **funz** fino a quando la temperatura non raggiunge i **20 gradi**.

Nota importante: Il salto **jrc** riesce a raggiungere un'**etichetta** solo se questa si trova ad una distanza pari a circa **8 righe** di **programma**.
Se eseguite un salto **jrc** su un'**etichetta** che dista più di **8 righe**, quando **assemblerete** il programma, vi verrà segnalato **errore** con la scritta "**5-bit displacement overflow**".

JRNC

Questa istruzione viene sempre inserita **dopo** un'istruzione **CP** o **CPI** per effettuare un salto **condizionato**.
Se dalla **comparazione** il microprocessore rileva che il numero presente nell'**accumulatore** è **maggiore** oppure **uguale** a quello presente nella **variabile**, viene effettuato un **salto** sull'**etichetta**.
Nella funzione **CPI** il relè si **diseccitava** quando la temperatura **superava** i **20 gradi**; se ora volete che il relè si **disecciti** quando la temperatura **scende** sotto i **20 gradi**, dovete modificare nella **terza** riga l'istruzione **jrc funz** con la scritta **jrnc funz**, come qui sotto riportato:

```
      ld    a,gradi    ; carica nell'accumul. i gradi
      cpi   a,20       ; compara i gradi con numero 20
      jrnc  funz       ; salta a funz se maggiore di 20
      res   7,port_b   ; diseccita il relè
funz                   ; non diseccitare il relè
```

In questo caso il **salto** viene effettuato sull'**etichetta** siglata **funz** solo se la temperatura è **maggiore** o **uguale** a **20 gradi**.
In pratica si ottiene la funzione **opposta** quella che si otteneva con l'istruzione **jrc**.

Nota importante: Il salto **jrnc** riesce a raggiungere un'**etichetta** solo se questa si trova ad una distanza pari a circa **8 righe** di **programma**.
Se eseguite un salto **jrnc** su un'**etichetta** che dista più di **15 righe**, quando **assemblerete** il programma, vi verrà segnalato **errore** con la scritta "**5-bit displacement overflow**".

JRNZ

Questa istruzione viene sempre inserita nel programma **dopo** un'istruzione **CP** o **CPI** per effettuare un salto **condizionato**.
Se dalla **comparazione** il microprocessore rileva che il numero presente nell'**accumulatore** è **diverso** da quello presente nella **variabile**, viene effettuato un **salto** sull'**etichetta**.

Nell'**istruzione DEC** avevamo riportato un esempio per far **diseccitare** un **relè** dopo **40 minuti**.

```
      ldi   tempo, 40  ; carica in tempo il numero 40
      .....  .......
      dec   tempo      ; decrementa la variabile tempo
      ld    a, tempo   ; carica in a la variabile tempo
      cpi   a, 0       ; confronta a con il numero 0
      jrnz  funz       ; se è diverso da 0 salta in funz
      res   7, port_b  ; diseccita relè
funz                   ; non diseccitare il relè
```

Poiché nella **variabile** abbiamo messo il numero **40** ed il microprocessore **decrementa** questo numero di **1**, avremo via via **39 - 38 - 37 ecc**.
Dopo ogni decremento il numero presente nella variabile viene **comparato** con il numero **0** e fino a quando il numero nella variabile è diverso da **0,** viene effettuato il **salto** nella riga **funz** ed il programma non esegue la successiva istruzione che **diseccita** il relè.
Solo quando il numero presente nella **variabile** è **uguale** a **0**, il microprocessore passa ad eseguire l'istruzione successiva e **diseccita** il relè.

Nota importante: Il salto **jrnz** riesce a raggiungere un'**etichetta** solo se questa si trova ad una distanza pari a circa **8 righe** di **programma**.
Se eseguite un salto **jrnz** su un'**etichetta** che dista più di **8 righe**, quando **assemblerete** il programma, vi verrà segnalato **errore** con la scritta "**5-bit displacement overflow**".

JRZ

Questa istruzione viene sempre inserita in un programma **dopo** un'istruzione **CP** o **CPI** per effettuare un salto **condizionato**.
Se dalla **comparazione** il microprocessore rileva che il numero presente nell'**accumulatore** è **uguale** a quello presente nella **variabile**, viene effettuato un **salto** sull'**etichetta**.

L'esempio riportato nell'**istruzione DEC** permetteva di **diseccitare** un **relè** dopo **40 minuti**.
Di seguito potete vedere le modifiche che abbiamo apportato al programma per usare l'istruzione **JRZ**.

```
      ldi   tempo,0    ; carica in tempo il numero 0
      .....  .....
      inc   tempo      ; incrementa la variabile tempo
      ld    a,tempo    ; carica in a la variabile tempo
      cpi   a,40       ; confronta a con il numero 40
      jrz   funz       ; se è uguale salta in funz
      jp    fine       ; se non è uguale salta in fine
funz                   ; prosegui alla riga dopo
      res   7,port_b   ; diseccita il relè
fine                   ; non diseccita il relè
```

In questo caso abbiamo messo nella **variabile** il numero **0**, poi avendo dato l'istruzione per **incre-**

mentare questo numero di **1**, avremo via via **0 - 1 - 2 - 3 ecc**.
Tutte le volte che viene effettuato un **incremento**, il numero presente nella **variabile tempo** viene **comparato** con il numero **40** e fino a quando questi due numeri sono **diversi** viene effettuato il **salto** sull'etichetta **fine** ed il relè non si diseccita.
Solo quando il numero presente nella variabile è uguale a **40**, il microprocessore compie un **salto** sull'etichetta **funz** ed esegue l'istruzione successiva **diseccitando** il relè.

Nota importante: Il salto **jrz** riesce a raggiungere un'**etichetta** solo se questa si trova ad una distanza pari a circa **8 righe** di **programma**.
Se effettuate un salto **jrz** su un'**etichetta** che dista più di **8 righe**, quando **assemblerete** il programma, vi verrà segnalato **errore** con la scritta "**5-bit displacement overflow**".

JRR

Questa istruzione serve per controllare se una **cifra** di un **numero binario** si trova a **livello logico 0** e quando si rileva questa condizione viene effettuato un **salto**.
L'istruzione **JRR** può risultare utile per controllare se il **piedino d'ingresso** di una qualsiasi porta **A-B-C** si trova a **livello logico 0** o a **livello logico 1** (vedi figg.10-11).
Come sapete ogni porta da **8 bit** è numerata **0A - 1A- 2A - 3A** ecc. **0B - 1B - 2B - 3B** ecc.
Per **controllare** quando l'interruttore posto sulla **porta 6B** è **chiuso** (vedi fig.12), e fare in modo che quando si riscontra questa condizione si ecciti un **relè** di allarme posto sulla porta **d'uscita 2A**, dovete scrivere l'istruzione in questo modo:

```
        jrr   6,port_b,eccita   ; controlla porta 6B
        jp    fine              ; va a fine se 6B è a 1
eccita                          ; etichetta per proseguire
        set   2,port_a          ; eccita il relè su 2A
fine                            ; non eccitare il relè
```

In questa istruzione è necessario fare un **doppio salto**: il primo serve ad eccitare il relè se il **l'interruttore** applicato sulla **porta 6B** è chiuso, il secondo (**jp fine**) serve a **non eccitare** il relè nel caso in cui l'interruttore non risulti chiuso.
Se non avessimo inserito l'istruzione **jp fine**, il microprocessore avrebbe proseguito con le istruzioni successive ed avrebbe ugualmente **eccitato** il relè anche se l'interruttore non fosse stato **chiuso**.

Nota importante: Il salto **jrr** riesce a raggiungere un'**etichetta** solo se questa si trova ad una distanza pari a circa **60 righe** di **programma**.

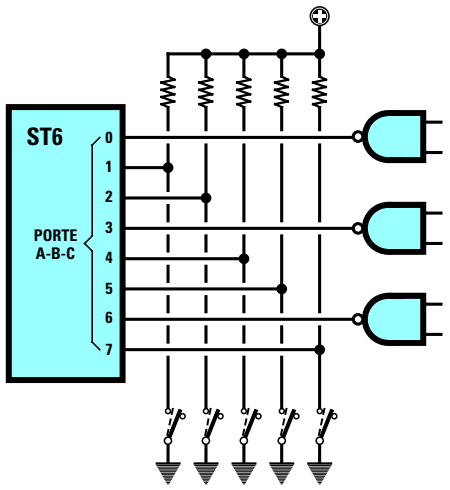


Fig.10 Le istruzioni JRR e JRS possono essere utili per controllare se le porte A-B-C utilizzate come ingressi sono a "livello logico 1" oppure a "livello logico 0". Usando l'istruzione JRR avviene un "salto" se sulla porta d'ingresso è presente un "livello logico 0", mentre usando l'istruzione JRS il "salto" avviene se sulla porta d'ingresso è presente un "livello logico 1".

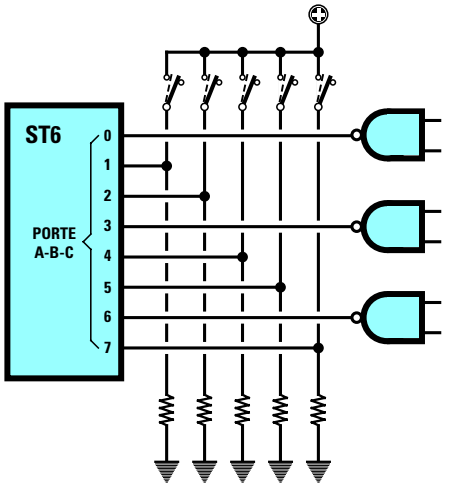


Fig.11 Dopo che vi avremo insegnato come predisporre una porta come Entrata, potrete collegarle degli interruttori o dei pulsanti rivolti verso "massa" (vedi fig.10) o verso il "positivo" (vedi fig.11) oppure l'uscita di una porta Nand - Nor - Or ecc. per controllare il loro livello logico. Nota: Sui vari ingresso dovrete collegare delle resistenze con valori compresi tra 3.300 - 100.000 ohm.

JRS

Questa istruzione serve per controllare se una **cifra** di un **numero binario** si trova a **livello logico 1** e quando si rileva questa condizione viene effettuato un **salto**.
L'istruzione **JRS** può essere utile per controllare se il **piedino d'ingresso** di una qualsiasi porta **A-B-C** si trova a **livello logico 0** o a **livello logico 1** (vedi figg.10-11).
Se volete **controllare** che l'interruttore posto sulla **porta 6B** risulti **chiuso** (vedi fig.13) e quando si riscontra questa condizione, eccitare un **relè** di allarme posto sulla porta d'**uscita 2A**, dovete scrivere l'istruzione in questo modo:

```
        jrs     6,port_b,eccita   ;controlla porta 6B
        jp      fine              ; va a fine se 6B è a 0
eccita                            ; etichetta per proseguire
        set     2,port_a          ; eccita il relè su 2A
fine                              ; non eccitare il relè
```

In questa istruzione è necessario compiere un **doppio salto**: il primo serve ad eccitare il relè se l'interruttore applicato sulla **porta 6B** è chiuso, il secondo (**jp fine**) serve a **non eccitare** il relè nel caso in cui l'interruttore non risulti chiuso.
Se non avessimo inserito l'istruzione **jp fine**, il microprocessore avrebbe proseguito con le istruzioni successive ed avrebbe ugualmente **eccitato** il relè anche se l'interruttore non fosse stato **chiuso**.

Nota importante: Il salto **jrs** riesce a raggiungere un'**etichetta** solo se questa si trova ad una distanza pari a circa **60 righe** di **programma**.

LD

Questa istruzione serve per **caricare** il **numero** contenuto in una **variabile** nell'**accumulatore** o viceversa.
Nella funzione **CPI**, in cui abbiamo riportato un esempio per far **diseccitare** un relè quando la temperatura **supera** i **20 gradi**, la **comparazione** viene effettuata solo con il **numero** presente nell'**accumulatore**, quindi abbiamo dovuto inserire all'interno dell'accumulatore il **numero** che era presente nella **variabile** chiamata **gradi**:

```
        ld      a,gradi   ; carica nell'accumul. i gradi
        cpi     a,20      ; compara i gradi con numero 20
        jrnc    funz      ; salta a funz se maggiore di 20
        res     7,port_b  ; diseccita il relè
funz                      ; non diseccitare il relè
```

Se la **variabile gradi** contiene il numero **15**, dopo l'istruzione **ld a,gradi** anche il numero presente nell'**accumulatore** avrà un valore di **15**.

LDI

Questa istruzione serve per **caricare** in una **variabile** oppure nell'**accumulatore** un qualsiasi **numero** da voi **prescelto** e compreso tra **0** e **255**.
Nell'**istruzione DEC** abbiamo riportato un esempio per **diseccitare** un **relè** dopo **40 minuti**. Questo numero va quindi inserito nella **variabile tempo** come qui sotto riportato:

```
        ldi     tempo,40  ; carica in tempo il numero 40
        ......  ......
        dec     tempo     ; decrementa la variabile tempo
        ld      a,tempo   ; carica in a la variabile tempo
        cpi     a,0       ; confronta a con il numero 0
        jrnz    funz      ; se è diverso salta in funz
        res     7,port_b  ; diseccita relè
funz                      ; non diseccitare il relè
```

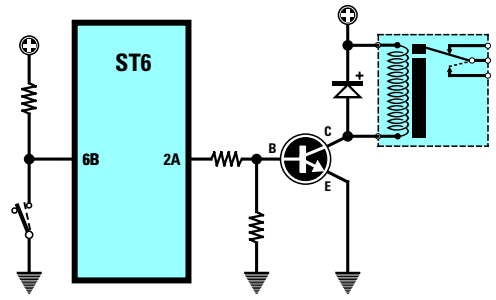


Fig.12 Nel programma riportato al paragrafo JRR il relè si eccita ogni volta che cortocircuitate verso "massa" l'interruttore posto sulla porta 6B. In questo modo sulla porta di uscita 2A ritroverete un livello logico 1 che polarizzerà il transistor.

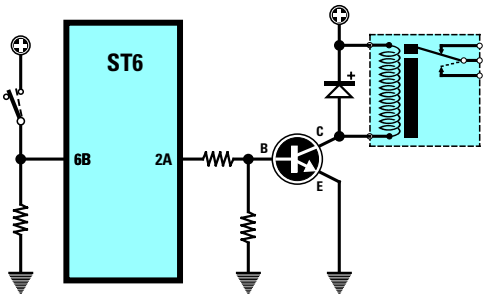


Fig.13 Nel programma riportato al paragrafo JRS il relè si eccita ogni volta che collegate verso il "positivo" l'interruttore posto sulla porta 6B. In questo modo sulla porta di uscita 2A ritroverete un livello logico 1 che polarizzerà il transistor.

Dopo l'istruzione **ldi tempo,40** scritta nella prima riga, la **variabile tempo** è uguale a **40**.

NOP

Questa istruzione viene usata pochissimo, perché serve solamente per ottenere un **ritardo** di qualche **microsecondo**. Infatti fa eseguire al microprocessore **2 cicli macchina** a **vuoto**.
Per eseguire questa funzione è sufficiente scrivere, dopo aver lasciato **uno spazio**, la parola **NOP**; scrivendola più volte aumenterete il **ritardo**.
Se nel microprocessore avete utilizzato un quarzo da **8 MHz**, che esegue **1 ciclo macchina** in un tempo di **1,625 microsecondi**, e scrivete:

```
        nop             ; ritardo 3,25 microsec.
        nop             ; ritardo 3,25 microsec.
        nop             ; ritardo 3,25 microsec.
```

otterrete un ritardo totale di **9,75 microsecondi**.

RES

Questa istruzione serve per **forzare** ad un **livello logico 0** il **bit** di una **variabile**.
Nella funzione **CPI** abbiamo riportato un esempio per **diseccitare** un relè quando la temperatura **supera** i **20 gradi**. Questa operazione è compiuta dalla istruzione **res** a cui bisogna specificare di seguito quale **porta** deve resettare (nel nostro esempio è la **7,port_b**).

```
        ld      a,gradi     ; carica nell'accumul. i gradi
        cpi     a,20        ; compara i gradi con numero 20
        jrnc    funz        ; salta a funz se maggiore di 20
        res     7,port_b    ; diseccita il relè
funz                        ; non diseccitare il relè
```

Ammettendo che tutte le uscite della **porta B** siano a **livello logico 1**, cioè:

11111111

Il programma modificherà il solo bit **7** della **porta B**, cioè il primo bit a sinistra, quindi vi ritroverete con:

01111111

Se voleste azzerare oltre il **piedino 7** della porta **B** anche i **piedini 6 - 5**, dovreste scrivere:

```
        res     7,port_b
        res     6,port_b
        res     5,port_b
```

Vi ricordiamo che il numero di **bit** per ogni porta va da **0** a **7**.

SET

Questa istruzione serve per **forzare** ad un **livello logico 1** il **bit** di una **variabile**.
Nell'istruzione **JRS** abbiamo riportato un esempio per **eccitare** un relè ogni volta che il **pulsante** posto sulla **porta 6B** viene **pigiato**.
Nella riga in cui è posta la funzione **set 2,port_a**, viene forzata l'uscita **2** della **porta A** a **livello logico 1**, in modo che provveda ad eccitare il relè.

```
        jrs     6, port_b,eccita    ; controlla porta 6B
        jp      fine                ; va a fine se 6B è a 0
eccita                              ; etichetta per proseguire
        set     2,port_a            ; eccita il relè su 2A
fine                                ; non eccitare il relè
```

In pratica l'istruzione **SET** compie l'operazione inversa all'istruzione **RES**.

RET

Questa istruzione viene posta alla fine di una **subroutine** per comunicare al microprocessore di ritornare nel punto del programma in cui questa **subroutine** è stata chiamata.
Nell'istruzione **CALL** abbiamo riportato una **subroutine** per ottenere un **ritardo**, cioè:

```
ritardo ldi     x,103       ; assegna 103 a x
ripeti  dec     x           ; sottrai 1 a x
        jrnz    ripeti      ; ripeti se x non è a 0
        ret                 ; ritorna al programma
```

Volendo ottenere un **ritardo** dovete richiamare la **subroutine** chiamata **ritardo** scrivendo:

```
        call    ritardo     ; chiama subroutine ritardo
```

Alla fine della **subroutine** abbiamo posto l'istruzione **RET** per far tornare il programma alla riga posta di seguito all'istruzione **call ritardo**.
Vi ricordiamo che al termine della **subroutine** occorre sempre mettere l'istruzione **RET**, diversamente il programma **non** ritornerà nel punto in cui abbiamo chiamato la subroutine, ma proseguirà con le righe successive, senza segnalare nessun **errore**, ma causando anomalie nel funzionamento del programma.

RETI

Questa istruzione viene utilizzata alla fine di particolari tipi di **subroutine** che si chiamano **interrupt** e che sono identificati con le **etichette**:

```
ad_int                      ; serve per il convertitore A/D
tim_int                     ; serve per il timer
BC_int                      ; serve per le porte B-C
A_int                       ; serve per la porta A
nmi_int                     ; serve per il piedino nmi
```

Dopo una di queste etichette, si **interrompono** momentaneamente le funzioni del programma **principale** e vengono eseguite le istruzioni che si trovano dopo l'**interrupt**.
Alla fine di questa **subroutine** di **interrupt** dovrete inserire l'istruzione **RETI** per ritornare al programma principale, nel punto in cui si trovava prima dell'**interruzione forzata**.
Come noterete, nel microprocessore **ST6** c'è un piedino chiamato appunto **nmi** (corrisponde al piedino **5**) sul quale potete collegare un pulsante (vedi fig.14) che potrete usare per **interrompere** forzatamente una funzione.
Ammettiamo che abbiate scritto un programma per l'automazione di un **trapano** e mentre questo sta eseguendo la foratura, **si spezzi la punta**.
In questo caso dovete **immediatamente** interrompere il programma e togliere la tensione di alimentazione dal trapano per sostituire la punta.
Il programma va scritto nel seguente modo:

```
nmi_int                    ; etichetta nmi di interrupt
        res   5,port_b     ; resetta la porta 5B
        reti               ; termine subroutine
```

La **porta 5B** è quella che eccita o diseccita il **teleruttore** che alimenta il trapano.

RLC

Questa istruzione serve per **spostare** verso sinistra tutte le **cifre** di un **numero binario** presente nell'**accumulatore**.
Sulla **destra** di tale numero entrerà un **1** o uno **0**, che risulta parcheggiato in una particolare cella di memoria **RAM** chiamata **CARRY**, presente all'interno del microprocessore.
Il valore del **carry** è **1** solo quando l'ultima operazione eseguita ha un riporto, in altre parole quando il risultato è superiore a **255**.
Ad esempio, se il valore nell'accumulatore è **250**, dopo aver eseguito l'istruzione:

```
        addi  a,100
```

il valore del **carry** è **1**.

Tornando all'istruzione **RLC**, se all'interno dell'**accumulatore** è presente il numero **binario 00000100** e nel **carry** è presente un **1**, scrivendo l'istruzione:

```
        rlc   a
```

ritroverete nell'accumulatore questo numero **binario**:

00001001

Se nel **carry** fosse parcheggiato uno **0**, nell'accumulatore ritrovereste questo numero **binario**:

00001000

La funzione **RLC** potrebbe risultarvi utile per accendere in sequenza a **ciclo continuo** dei diodi led. Infatti il numero che **perdete** sulla sinistra entra nel **carry** per rientrare poi sulla **destra**.

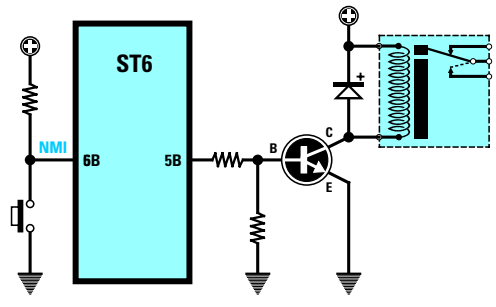


Fig.14 Se scrivete il programma con l'etichetta "nmi_int" riportato nel paragrafo RETI (vedi sopra), quando il piedino 5 dell'NMI viene posto a livello logico 0, anche sull'uscita 5B ritroverete un livello logico 0 ed il relè si disecciterà.

SLA

Anche questa istruzione serve per **spostare** verso sinistra tutte le **cifre** di un **numero binario** presente nell'**accumulatore**, con la sola differenza che, non utilizzando il **carry**, sulla destra entra sempre uno **0** ed il numero che fuoriesce a **sinistra** va **perso**.
L'istruzione **SLA** può essere utile per ottenere una **moltiplicazione** per **2**.
Ad esempio, se nell'**accumulatore** è presente questo numero **binario**

00010111

che corrisponde al numero **decimale 23**, scrivendo:

```
        sla   a
```

nell'accumulatore troverete questo diverso numero **binario**:

00101110

che corrisponde al numero **decimale 46** (equivalente a **23 x 2**).
Il numero massimo che potete **moltiplicare** per 2 è **127**.

STOP

Questa istruzione serve per interrompere l'esecuzione di un programma e per spegnere lo stadio oscillatore del **clock**.
E' un'istruzione che si usa raramente.

WAIT

Questa istruzione serve per interrompere l'esecuzione di un programma, con la sola differenza rispetto alla precedente, che non si spegne lo stadio oscillatore del **clock**.

SUB

Questa istruzione viene utilizzata per **sottrarre** dal numero presente nell'**accumulatore** il numero contenuto in una **variabile**.
Se nell'**accumulatore** è presente il numero **183** e volete sottrargli il numero decimale **53**, dovrete inserire in una variabile, chiamata ad esempio **pippo**, il numero **53** scrivendo questa istruzione:

```
      ldi   a, 183      ; numero nell'accumulatore
      ldi   pippo, 53   ; numero inserito nella variabile
      sub   a, pippo    ; sottrai da a il valore di pippo
```

Eseguita questa operazione nell'accumulatore avrete **183-53 = 130**.

SUBI

Questa istruzione viene utilizzata per **sottrarre** dal numero presente nell'**accumulatore** un numero da voi inserito nella riga di questa istruzione.
Se nell'**accumulatore** è presente il numero **183** e gli volete sottrarre il numero **53**, dovrete scrivere l'istruzione in questo modo:

```
      ldi   a,183       ; numero nell'accumulatore
      subi  a,53        ; sottrai da a il numero 53
```

Il risultato presente nell'accumulatore sarà dopo questa istruzione **130**.
Come per la precedente, anche con questa istruzione non potete sottrarre un numero **maggiore** a quello presente nell'**accumulatore**.

CONTINUA nel PROSSIMO NUMERO

Sebbene con gli articoli e gli esempi pubblicati sulle riviste N.172/173 e 174 siate già riusciti a **programmare** e a **cancellare** senza difficoltà il microprocessore **ST6**, questo non è ancora sufficiente per passare alla **fase** più propriamente **pratica**, iniziare cioè a scrivere dei programmi, perché occorre prima conoscere come vanno correttamente utilizzati:

- la funzione **watchdog**
- i piedini delle **porte** come **ingressi**
- i piedini delle **porte** come **uscite**
- la funzione **interrupt**
- la funzione **A/D converter**
- la funzione **timer**

IMPARARE a programmare

NOTA IMPORTANTE

Negli articoli precedenti (vedi riviste N.172/173 e 174) vi abbiamo consigliato di scrivere tutte le istruzioni in **minuscolo** sottolineando nel frattempo che non cambia assolutamente nulla se utilizzate la forma **maiuscola**.
Abbiamo tuttavia notato che la tipografia quando trascrive questi testi in **minuscolo** compie più frequentemente **errori tipografici**; infatti confonde la lettera **i** con la lettera **l** e viceversa, la lettera **r** per la **n**, e spesso trascrive così vicino le due lettere **r n** da farle sembrare una **m**.
Proprio per evitare questo tipo di errori da adesso in poi scriveremo le righe dei programmi un po' in **maiuscolo** ed un po' in **minuscolo**.
Ripetiamo nuovamente che ogni **riga** di programma è composta da **quattro blocchi** chiamati **Etichetta - Istruzione - Operando ; Commento** (vedi fig.1), che dovrete tenere distanziati gli uni dagli altri con un carattere di **tabulazione**.
In questo modo avrete tutte le istruzioni perfettamente incolonnate ed in caso di necessità potrete controllarle meglio.
Non è obbligatorio inserire l'ultimo **blocco**, quello del **commento**, ma se lo aggiungete dovete ricordarvi di farlo precedere sempre da un **punto e virgola**.
Nei programmi che troverete di seguito ogni **blocco** è stato evidenziato da un rettangolo **colorato** in azzurro.
Così se ad esempio in una riga di programma **manca** l'**etichetta**, troverete al suo posto un **rettangolo** colorato senza alcuna **scritta** al suo interno.
Questo per ricordarvi che per avere tutte le istruzioni incolonnate dovete lasciare un carattere di **tabulazione** prima di scrivere il secondo blocco, quello cioè dell'**istruzione**.

WATCHDOG

Il **watchdog** è un contatore pilotato dalla frequenza di **clock** del quarzo, che, iniziando dal numero che avete inserito nel registro **wdog** (è sempre bene inserire un numero alto, ad esempio **255**), conta a rovescio fino ad arrivare a **0**.
Quando raggiunge lo **0**, il watchdog **resetta** automaticamente il microprocessore, che in questo modo non potrà più **proseguire** con il programma che stava eseguendo.
Il **watchdog** impedisce che eventuali **disturbi** presenti sulla tensione di rete dei **220 volt** o generati da altri fonti, entrando involontariamente nel microprocessore lo predispongano per eseguire funzioni **non previste**.
Tanto per portare un esempio, supponiamo di aver programmato un **ST6** per far eseguire ad un **tornio automatico** delle viti **filettate** lunghe **10 millimetri**.
Può verificarsi il caso in cui un **disturbo**, entrando nel micro, lo predisponga per **togliere** la filettatura

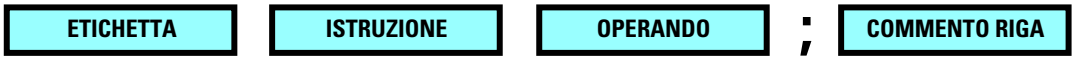


Fig.1 Ogni riga di programma è composta da quattro blocchi che occorre tenere distanziati da uno "spazio". Potrete anche non inserire l'ultimo blocco del COMMENTO, ma se lo utilizzate, dovrete farlo precedere da un "punto e virgola". Se nella riga di un programma mancasse l'ETICHETTA, dovrete lasciare uno SPAZIO prima di scrivere l'ISTRUZIONE.

i microprocessori ST6

Poiché ci siamo ripromessi di insegnarvi a programmare correttamente un ST6, è nostra intenzione spiegarvi in modo molto semplice e con numerosi esempi tutti i passi che bisogna compiere, per evitare che commettiate quei comuni ed involontari piccoli errori, che potrebbero impedire il regolare funzionamento del microprocessore.

o per fare delle viti lunghe **10 centimetri**.
Per **evitare** che, in assenza di **disturbi**, il contatore del **watchdog** possa raggiungere lo **0** e quindi **resettare** il microprocessore mentre sta eseguendo le istruzioni del programma, dovrete sempre ricordarvi di **inserire** ogni **20 - 30 - 40 righe** di programma questa scritta:

```
        LDI   wdog,255
```

Conviene inserire questa istruzione dopo un'**etichetta** che faccia ripetere alcune righe di programma diverse volte, perché è in questi casi che il contatore del **watchdog** può più facilmente **scaricarsi**, cioè raggiungere lo **0**.
Se ciò avviene, il microprocessore si **resetta**, in altre parole non può più proseguire con le successive istruzioni ed il programma riparte dall'inizio.
Molti programmatori principianti non trovando un'esauriente spiegazione e nemmeno nessun esempio su come utilizzarla, non inserivano questa istruzione, e quando il programma si **bloccava** perché il **watchdog** arrivava a **0**, non sapendo trovare un'altra spiegazione, cambiavano l'**ST6** ritenendolo **difettoso** oppure cercavano nel programma un **errore**, che in realtà non esisteva.
A chi ci ha interpellato per queste **anomalie** abbiamo chiesto se vicino ad un'etichetta **ripeti** o ad una funzione che il programma eseguiva diverse volte era stata riportata la riga **LDI wdog,255**, e quasi sempre abbiamo ricevuto una risposta negativa.
Dopo aver spiegato che il **watchdog** si può paragonare ad una **pila ricaricabile** e, come tale, ogni tanto occorre **ricaricarla** per evitare che arrivi a **0 volt**, tutti hanno capito l'importanza di questa funzione e dopo averla inserita nel loro programma gli inconvenienti lamentati sono spariti.

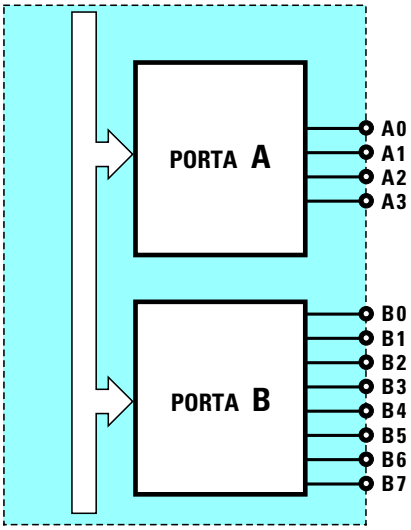
Negli **esempi** che trovate in questo articolo troverete spesso questa istruzione:

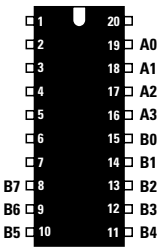
```
ripeti  LDI   wdog,255   ; carica il watchdog
        .....            ; programma
        JP    ripeti     ; salta a ripeti
```

che in pratica serve a **ricaricare** questa "pila" sul numero massimo che possiamo inserire, cioè **255**.

TABELLA N.1 per ST62/E10 - ST62/T10 - ST62/E20 - ST62/T20

porta	A0	A1	A2	A3	B0	B1	B2	B3	B4	B5	B6	B7
piedino	19	18	17	16	15	14	13	12	11	10	9	8





ST 62 E 10

ST 62 T 10

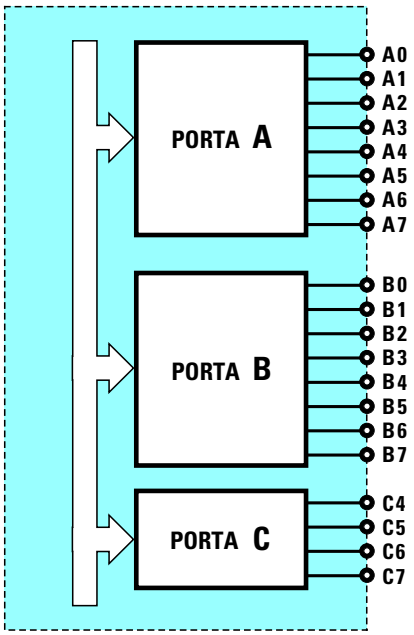
ST 62 E 20

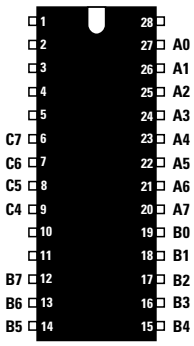
ST 62 T 20

Fig.2 All'interno dei microprocessori ST62 della serie 10-20 sono presenti due PORTE chiamate A-B. Nella porta A troviamo solo quattro piedini Ingressi/Uscita siglati A0-A1-A2-A3, mentre nella porta B troviamo otto piedini siglati da B0 a B7.
Nella Tabella sopra riportata abbiamo indicato a quale piedino dell'ST6 corrispondono i piedini di queste due PORTE.

TABELLA N.2 per ST62/E15 - ST62/T15 - ST62/E25 - ST62/T25

porta	A0	A1	A2	A3	A4	A5	A6	A7	B0	B1	B2	B3	B4	B5	B6	B7	C4	C5	C6	C7
piedino	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	9	8	7	6





ST 62 E 15

ST 62 T 15

ST 62 E 25

ST 62 T 25

Fig.3 All'interno dei microprocessori ST62 della serie 15-25 sono presenti tre PORTE chiamate A-B-C.
Nelle due porte A-B sono presenti otto piedini Ingressi/Uscita siglati da A0 a A7 e da B0 a B7, mentre nella porta C troviamo solo quattro piedini siglati C4-C5-C6-C7.
Nella Tabella sopra riportata abbiamo indicato a quale piedino dell'ST6 corrispondono i piedini di queste tre PORTE.

LE PORTE sui piedini del MICROPROCESSORE

All'interno di tutti i microprocessori **ST6** sono presenti due oppure tre **porte** siglate **A-B-C** i cui terminali, contraddistinti dalle sigle **A0 - A1 - A2** ecc. e **B0 - B1 - B2** ecc., fanno capo ai piedini del microprocessore (vedi figg.2-3).
Nelle **Tabelle N.1** e **N.2** riportiamo i numeri dei piedini a cui sono collegati i terminali delle **porte** presenti nei diversi integrati **ST6**.
Conoscere a quale **piedino** risultano collegati i terminali di queste **porte** è molto importante, perché nella **riga** del programma non viene mai indicato il **numero** del piedino, ma la sola **sigla** della **porta**, cioè **A0 - A2 - A3** o **B5 - B6 - B7** ecc.

COME SETTARE LE PORTE

Per **settare** il piedino prescelto come **ingresso senza pull-up** - **ingresso con interrupt - ingresso analogico** - **ingresso pull-up** oppure come **uscita open collector - uscita push-pull**, dobbiamo inserire nei registri **pdir - popt - port** degli **0** o degli **1**, disponendoli come abbiamo riportato nella **Tabella N.3**.

Esempio: Ammesso di voler predisporre tutti i piedini della **porta A** come **ingressi** tipo **pull-up**, dovremmo necessariamente scrivere queste tre righe:

```
LDI   pdir_a,00000000B
LDI   popt_a,00000000B
LDI   port_a,00000000B
```

Se volessimo invece predisporre tutti i piedini della **porta A** come **uscita** in **push-pull** dovremmo necessariamente scrivere queste tre righe.

```
LDI   pdir_a,11111111B
LDI   popt_a,11111111B
LDI   port_a,00000000B
```

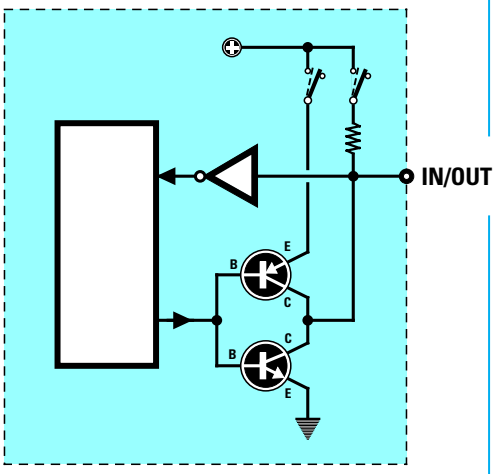


Fig.4 Inserendo nei tre registri PDIR - POPT - PORT degli 0 o degli 1 (vedi Tabella N.3), noi possiamo predisporre il piedino di una qualsiasi Porta a funzionare come Ingresso oppure come Uscita.

Ricordatevi che dopo il nome dei tre registri, **pdir - popt - port**, bisogna sempre indicare il tipo di **porta** che si vuole utilizzare scrivendo:

_a se la porta è la **A**
_b se la porta è la **B**
_c se la porta è la **C**

Non scrivete **-a**, in altre parole non utilizzate il segno della **sottrazione**, ma sempre il segno del **sottolineato**, cioè **_a**.
La lettera **B**, posta dopo l'ultima cifra di destra di ogni riga, serve ad indicare che si tratta di un numero **binario**.

TABELLA N.3 per predisporre gli ingressi e le uscite

Registri	INGRESSI				USCITE	
	con pull-up	senza pull-up	con interrupt	per segnali analogici	open collector	uscita push-pull
pdir	0	0	0	0	1	1
popt	0	0	1	1	0	1
port	0	1	0	1	0	0

Fig.5 Per predisporre il piedino di una Porta a funzione come Ingresso oppure come Uscita, bisogna scrivere nei tre registri PDIR - POPT - PORT gli 0 o gli 1 nell'ordine riportato in questa Tabella. Nell'articolo trovate molti esempi su come settare queste Porte.

NOTA IMPORTANTE: Non tutti i piedini delle **porte** si possono settare come **ingresso** per **segnali analogici**, quindi per evitare **errori** ricordatevi che di tutti i microprocessori riportati sia nella **Tabella N.1** sia nella **Tabella N.2**, non potrete mai utilizzare come **ingressi analogici** i piedini **A0 - A1 - A2 - A3**.
Queste porte potranno invece essere **settate** per tutti gli altri tipi di ingresso, cioè **con pull-up - senza pull-up - con interrupt**.
Al contrario tutti i piedini di tutte le porte possono essere **settati** come **uscite**, tenendo presente che l'**uscita** più comunemente utilizzata è il **push-pull**, che si ottiene ponendo a **1-1-0** i tre registri **pdir**, **popt** e **port**.

Per concludere facciamo presente che l'**ingresso** più comunemente utilizzato è quello **con pull-up** che si ottiene ponendo a **0-0-0** i tre registri **pdir**, **popt** e **port**.
E' inoltre sempre consigliabile **settare** come **ingressi** tutti i piedini che non vengono usati, perché in questo modo non si corre il rischio di provocare involontariamente dei **cortocircuiti** che potrebbero mettere fuori uso il microprocessore.
Infatti se un piedino che non viene utilizzato viene **settato** come **uscita** e poi viene collegato involontariamente a **massa**, anche tramite una resistenza di basso valore, può capitare che per un **errore** nel programma questo piedino cambi il suo stato logico da **0** a **1**, ed in questo caso in uscita si potrebbe ritrovare una tensione **positiva** di **5 volt** che potrebbe provocare dei **cortocircuiti**.
Per questo motivo, come noterete dagli esempi, **tutti** i piedini delle **porte A - B - C** che non vengono utilizzati li abbiamo settati come **ingressi** in **pull-up**.

Posizione dei PIEDINI nel NUMERO BINARIO

Per poter predisporre i piedini delle **porte** come **ingressi** o come **uscite** si utilizza un numero **binario** composto da otto cifre.
Molti tra voi si staranno chiedendo come si fa a capire qual è il piedino **A0 - A1 - A2** ecc. oppure **B0 - B1 - B2** ecc., in quanto anche se abbiamo indicato il tipo di porta con **_a**, con **_b** o con **_c**, non abbiamo mai precisato il suo terminale **0-1-2-3-4-5-6-7**.
Inoltre non abbiamo ancora specificato come ci si deve comportare con i microprocessori della **Tabella N.1**, dove la **porta A** ha solo quattro piedini siglati **A0 - A1 - A2 - A3** o con quelli della **Tabella N.2**, dove la **porta C** ha invece quattro piedini siglati **C4 - C5 - C6 - C7**.

Ogni **terminale** di queste **porte** viene definito da uno degli **otto** numeri posti dopo la **virgola**, ricordando che il terminale **7** è la prima cifra a **sinistra** ed il terminale **0** è l'ultima cifra a **destra**, come qui sotto riportato:

TABELLA N.4

cifra	1°	2°	3°	4°	5°	6°	7°	8°
piedino	A7	A6	A5	A4	A3	A2	A1	A0
piedino	B7	B6	B5	B4	B3	B2	B1	B0
piedino	C7	C6	C5	C4				

Quindi se volessimo caricare un **1** nel registro **pdir** di **A7** dovremmo scrivere:

```
LDI   pdir_a,10000000B
```

mentre se volessimo caricare un **1** nel registro **pdir** di **A4** dovremmo scrivere:

```
LDI   pdir_a,00010000B
```

ed ancora se volessimo caricare un **1** nel registro **pdir** di **A0** dovremmo scrivere:

```
LDI   pdir_a,00000001B
```

Per completare il **settaggio** di ogni porta si devono aggiungere le altre **due** righe di programma, dove compaiono i registri **popt** e **port**, mettendo degli **0** o degli **1** come riportato nella **Tabella N.3**.

Ammesso che si voglia **settare** il piedino **5** della **porta B** come **uscita** in **push-pull** si dovrà scrivere:

```
LDI   pdir_b,00100000B
LDI   popt_b,00100000B
LDI   port_b,00000000B
```

Infatti la sequenza per settare un piedino in **uscita push-pull** è **1-1-0**, e, come abbiamo riportato nella **Tabella N.4**, il piedino **B5** è la terza cifra a partire da sinistra.
Come abbiamo già avuto modo di dire, **tutti** i piedini delle **porte A - B - C** che non vengono utilizzati devono comunque essere settati come **ingressi** in **pull-up (0 - 0 - 0)**.

Nei microprocessori che hanno solo quattro piedini per la **porta A** (**A0 - A1 - A2 - A3**), si devono comunque riportare sempre tutte le otto cifre del **numero binario**.
Quindi se volessimo predisporre come **uscite pu-**

sh-pull i piedini della porta **A** di un **ST62E10**, dovremmo scrivere:

```
LDI  pdir_a,00001111B
LDI  popt_a,00001111B
LDI  port_a,00000000B
```

Allo stesso modo, se volessimo predisporre come **uscite push-pull** tutti i quattro piedini dei microprocessori che hanno anche la **porta C** (**C4 - C5 - C6 - C7**) dovremmo scrivere:

```
LDI  pdir_c,11110000B
LDI  popt_c,11110000B
LDI  port_c,00000000B
```

Ogni piedino di ogni **porta** può essere singolarmente **settato** come **ingresso** o come **uscita** e persino in maniera differente.
Ad esempio, possiamo **settare** il piedino **B1** della **porta B** come **uscita push-pull**, il piedino **B2** come **ingresso pull-up**, il piedino **B3** come **ingresso analogico** ed il piedino **B4** come **uscita open collector**.

ESEMPI di SETTAGGIO piedini

Esempio N.1 = Disponendo di un **ST6** del tipo riportato nella **Tabella N.1**, ad esempio un **ST62E10**, vorremmo programmare i piedini **A0 - A1** come **ingressi senza pull-up** ed i piedini **B4 - B5 - B6 - B7** come **uscite** in **push-pull**.

Soluzione: Per scrivere queste istruzioni esaminiamo prima di tutto la **Tabella N.3**, per sapere qual è la cifra, se **0** o **1**, che dobbiamo mettere nei tre registri **pdir - popt - port**.
Sapendo che per ogni riga che definisce il settaggio dei piedini dobbiamo sempre mettere **8 cifre** e che il piedino **7** è definito dalla prima cifra a **sinistra** e lo **0** dall'ultima cifra a **destra**, possiamo scrivere le nostre istruzioni:

```
LDI  pdir_a,00000000B
LDI  popt_a,00000000B
LDI  port_a,00000011B

LDI  pdir_b,11110000B
LDI  popt_b,11110000B
LDI  port_b,00000000B
```

legenda:

LDI = significa **carica** sul registro.

pdir - popt - port = sono i tre registri necessari per **settare** una porta.

_a e **_b** = sono le **porte A** e **B** ed il numero che segue dopo la **virgola** indica quale degli **otto** piedini, **7-6-5-4-3-2-1-0**, vogliamo **settare** come **ingresso** o come **uscita**.

B = questa lettera posta sull'estrema destra indica che il **numero** a **otto** cifre è un **Binario**.

Esempio N.2 = Disponendo di un **ST6** del tipo riportato nella **Tabella N.2**, ad esempio un **ST62E15**, vorremmo programmare tutti i piedini delle **porte A-B-C** come **ingressi pull-up**.

Soluzione: Controlliamo nella **Tabella N.3** se dobbiamo mettere uno **0** o un **1** nei tre registri **pdir - popt - port** per poterli settare come **ingressi pull-up**, quindi poiché in tutti va inserito uno **0** scriviamo:

```
LDI  pdir_a,00000000B
LDI  popt_a,00000000B
LDI  port_a,00000000B

LDI  pdir_b,00000000B
LDI  popt_b,00000000B
LDI  port_b,00000000B

LDI  pdir_c,00000000B
LDI  popt_c,00000000B
LDI  port_c,00000000B
```

Come avrete notato, anche sulla porta **C** abbiamo messo tutti **0** sebbene questa abbia solo quattro piedini siglati **C4 - C5 - C6 - C7**.

Esempio N.3 = Disponendo di un **ST6** del tipo riportato nella **Tabella N.2** vorremmo programmare il piedino **B2** come **ingresso pull-up**, il piedino **B1** come **ingresso analogico** ed il piedino **C7** come **uscita push-pull**.

Soluzione: Controlliamo innanzitutto nella **Tabella N.3** come vanno settati i tre registri **pdir - popt - port** per avere un **ingresso pull-up**, un **ingresso analogico** ed una **uscita push-pull**, quindi sapendo che per ogni riga di programma dobbiamo

sempre mettere **8 cifre** e che il piedino **7** è definito dalla prima cifra a **sinistra** e lo **0** dall'ultima cifra a **destra**, scriviamo:

```
        LDI   pdir_a,00000000B
        LDI   popt_a,00000000B
        LDI   port_a,00000000B

        LDI   pdir_b,00000000B
        LDI   popt_b,00000010B
        LDI   port_b,00000010B

        LDI   pdir_c,10000000B
        LDI   popt_c,10000000B
        LDI   port_c,00000000B
```

Come indicato dalla **Tabella N.3**, nei tre registri relativi alla porta **B** abbiamo messo **0-0-0** per il piedino **B2** e **0-1-1** per il piedino **B1**, mentre nelle tre righe relative alla **porta C**, per il piedino **C7** abbiamo posto **1-1-0**.

Qualcuno si starà già chiedendo in quale applicazione pratica possiamo utilizzare un piedino **settato** come **ingresso**. Anche se in seguito troverete alcuni **esempi** su questo argomento, possiamo anticiparvi subito che potete usarlo per vedere se sull'**ingresso** entra una tensione positiva oppure per stabilire se questa cambia di stato logico da **0** a **1** e viceversa, o ancora per **convertire** una tensione **analogica** in una **digitale** ecc.

Esempio N.4 = Abbiamo realizzato lo schema di fig.6 da utilizzare per un antifurto, quindi vorremmo che si **eccitasse** un relè ogni volta che l'interruttore viene pigiato.

Soluzione: La prima operazione da compiere è quella di **settare** il piedino della porta che vogliamo utilizzare come **ingresso** ed il piedino della porta che vogliamo utilizzare come **uscita**.
Come **ingresso** abbiamo deciso di scegliere il piedino **A1** e come **uscita** il piedino **A2**.

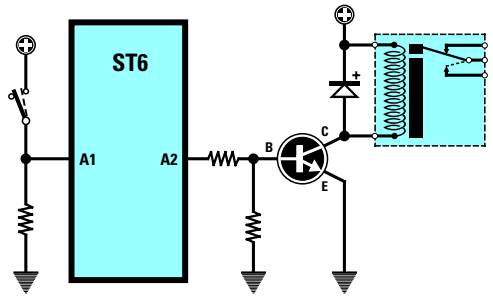


Fig.6 Schema da utilizzare per far eccitare il relè con il programma dell'ESEMPIO N.4.

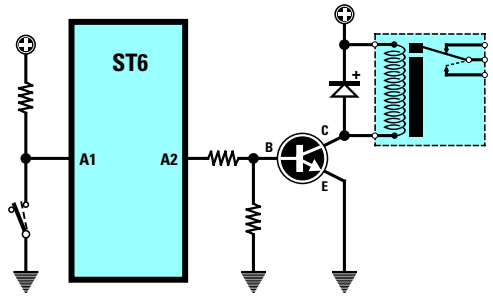


Fig.7 Schema da utilizzare per far eccitare il relè con il programma dell'ESEMPIO N.5.

Dalla **Tabella N.3** controlliamo come settare i registri **pdir - popt - porta** per far diventare questi piedini **ingressi** o **uscite**, dopodiché possiamo scrivere tutte le righe del programma indicato a fine pagina.
Nella riga **ripeti** abbiamo ricaricato il **watchdog**, poi con l'istruzione **JRS** abbiamo detto al programma di saltare all'etichetta **eccita** solo quando sul piedino **A1** riscontra una tensione di **5 volt**.
Se non riscontra questa condizione, il microprocessore passa ad eseguire la riga **JRR** e da qui salta all'etichetta **spegni**, che **diseccita** il relè se prima risultava eccitato, o lo lascia diseccitato se si trova già in questa condizione.

```
        LDI   pdir_a,00000100B  ; con queste tre righe abbiamo settato la porta A1 come
        LDI   popt_a,00000100B  ; ingresso e la porta A2 come uscita
        LDI   port_a,00000000B  ;
ripeti  LDI   wdog,255          ; ricarica il watchdog
        JRS   1,port_a,eccita   ; se in A1 ci sono 5 Volt salta a eccita
        JRR   1,port_a,spegni   ; se in A1 ci sono 0 Volt salta a spegni
eccita  SET   2,port_a          ; setta a 5 volt l'uscita di A2
        JP    ripeti            ; salta all'etichetta ripeti
spegni  RES   2,port_a          ; setta a 0 volt l'uscita di A2
        JP    ripeti            ; salta all'etichetta ripeti
```

Esempio N.5 = Abbiamo realizzato lo schema di fig.7 e vorremmo che, quando l'interruttore **corto-circuita** verso massa il piedino d'ingresso **A1**, si **eccitasse** il relè collegato sull'uscita **A2**.
Soluzione: La prima operazione da compiere è quella di **settare** il piedino **A1** come **ingresso** ed il piedino **A2** come **uscita**.
In questo caso, dovendo ottenere una funzione opposta a quella dell'**esempio N.4**, dovremo scrivere il programma come indicato di seguito:

```
        LDI   pdir_a,00000100B   ; con queste tre righe abbiamo settato la porta A1 come
        LDI   popt_a,00000100B   ; ingresso e la porta A2 come uscita
        LDI   port_a,00000000B   ;
ripeti  LDI   wdog,255           ; ricarica il watchdog
        JRR   1,port_a,eccita    ; se in A1 ci sono 0 Volt salta a eccita
        JRS   1,port_a,spegni    ; se in A1 ci sono 5 Volt salta a spegni
eccita  SET   2,port_a           ; setta a 5 volt l'uscita di A2
        JP    ripeti             ; salta all'etichetta ripeti
spegni  RES   2,port_a           ; setta a 0 volt l'uscita di A2
        JP    ripeti             ; salta all'etichetta ripeti
```

Esempio N.6 = Ammettiamo di voler predisporre tutti i piedini della porta **A** e della porta **B** come **uscite** in **push-pull**.

Soluzione: Per ottenere questa condizione dobbiamo scrivere solo **1** per i due registri **pdir - popt** e solo **0** per il terzo registro **port** (vedi **Tabella N.3**).

```
        LDI   pdir_a,11111111B
        LDI   popt_a,11111111B
        LDI   port_a,00000000B

        LDI   pdir_b,11111111B
        LDI   popt_b,11111111B
        LDI   port_b,00000000B
```

Se la porta **A** disponesse di solo **4 piedini** (vedi microprocessori della **Tabella N.1**) dovremmo scrivere **1** solo sui piedini delle porte utilizzate.

Esempio N.7 = Supponiamo di voler predisporre i piedini **A0 - A1 - A2 - A3** e **B0 - B1** come **ingressi pull-up** e i piedini **A4 - A5 - A6 - A7** e **B2 - B3 - B4 - B5 - B6 - B7** come **uscite push-pull**.
In questo caso dovremo scrivere:

```
        LDI   pdir_a,11110000B
        LDI   popt_a,11110000B
        LDI   port_a,00000000B

        LDI   pdir_b,11111100B
        LDI   popt_b,11111100B
        LDI   port_b,00000000B
```

Esempio N.8 = Abbiamo un microprocessore del tipo riportato nella **Tabella N.2**, con a disposizione le tre porte **A-B-C**, e vorremmo predisporre tutte i piedini della porta **A-B** come **ingressi pull-up** e quelli della porta **C** come **uscite push-pull**.
In questo caso dovremo scrivere:

```
        LDI   pdir_a,00000000B
        LDI   popt_a,00000000B
        LDI   port_a,00000000B

        LDI   pdir_b,00000000B
        LDI   popt_b,00000000B
        LDI   port_b,00000000B

        LDI   pdir_c,11110000B
        LDI   popt_c,11110000B
        LDI   port_c,00000000B
```

INTERRUPT

L'**Interrupt** serve per "interrompere" momentaneamente l'esecuzione delle istruzioni principali affinché il microprocessore possa eseguire altre istruzioni, che si trovano inserite tra una delle cinque **etichette** qui sotto riportate e la scritta **reti**.

```
ad_int          ;  etichetta
        .....   ;  programma da eseguire
        reti    ;  fine interrupt

tim_int         ;  etichetta
        .....   ;  programma da eseguire
        reti    ;  fine interrupt

BC_int          ;  etichetta
        .....   ;  programma da eseguire
        reti    ;  fine interrupt
```

```
A_int       ;  etichetta
            .....   ;  programma da eseguire
            reti    ;  fine interrupt

nmi_int     ;  etichetta
            .....   ;  programma da eseguire
            reti    ;  fine interrupt
```

Nota: la funzione di **interrupt** viene abilitata ed eseguita dal microprocessore solo se nel programma è stata scritta la seguente istruzione:

```
            SET     4,ior
```

Dopo aver scritto le righe di programma da eseguire con la funzione **interrupt** è necessario completarle con la scritta **reti**, perché quando il microprocessore arriva ad eseguire l'istruzione **reti** ritorna al programma principale nel punto in cui era stato **momentaneamente** interrotto.
L'**interrupt** viene utilizzato per far eseguire al microprocessore un'istruzione che in quel momento è **più importante** di quella che stava eseguendo. Ad esempio, se avessimo programmato un microprocessore per **aprire** e **chiudere** il cancello scorrevole del nostro giardino, dovremmo utilizzare la funzione **interrupt** applicata alla **fotocellula** per fare in modo che in fase di **chiusura** se un bambino o il nostro cane attraversa improvvisamente la **fotocellula**, il cancello si blocchi.

Etichetta ad_int

Questa etichetta serve per riconoscere quando il convertitore **A/D** ha terminato la conversione. Normalmente non si usa **mai** perché in sua sostituzione si preferisce scrivere questa riga di programma.

```
attendi     JRR     6,adcr,attendi
```

Etichetta tim_int

Questa etichetta viene utilizzata nelle funzioni **timer**. In pratica quando il registro **tcr** (vedi paragrafo **Timer**) arriva a **0**, il microprocessore esegue tutte le istruzioni che sono state scritte tra **tim_int** e **reti**.

Etichetta BC_int

Questa etichetta viene utilizzata per fare eseguire tutte le istruzioni che abbiamo scritto tra **BC_int** e **reti** quando su uno dei piedini da noi scelto delle porte **B** o **C** la tensione cambia di stato, in altre parole quando passa dal **livello logico 0** al **livello logico 1** (fronte di **salita**) o quando passa dal **livello logico 1** al **livello logico 0** (fronte di **discesa**).

Nota: per la funzione **ingresso con interrupt** possiamo abilitare **un solo piedino** di una delle due porte **B** o **C**.

Esempio N.9 = Supponiamo di voler **settare** il piedino **B2** come ingresso **con interrupt** che intervenga sul **fronte di salita** e tutti gli altri piedini della **porta B** come ingresso in **pull-up**. Come prima operazione controlliamo nella **Tabella N.3** come vanno **settati** i registri **pdir - popt - port** per l'ingresso **con interrupt**, dopodiché possiamo scrivere il programma indicato "**Esempio n.9**".

PROGRAMMA per Esempio n.9

```
            LDI     pdir_b,00000000B  ;  abbiamo settato il piedino B2 come ingresso interrupt
            LDI     popt_b,00000100B  ;  e tutti gli altri come ingressi pull-up
            LDI     port_b,00000000B  ;
            SET     4,ior             ;  serve per abilitare l'interrupt
            SET     5,ior             ;  serve per sentire il fronte di salita
BC_int      ;                         ;  etichetta d'inizio interrupt
            .....                     ;  istruzioni da eseguire con l'interrupt
            RETI                      ;  istruzione fine interrupt
```

PROGRAMMA per Esempio n.10

```
            LDI     pdir_b,00000000B  ;  abbiamo settato il piedino B7 come ingresso interrupt
            LDI     popt_b,10000000B  ;  e tutti gli altri come ingressi pull-up
            LDI     port_b,00000000B  ;
            SET     4,ior             ;  serve per abilitare l'interrupt
            RES     5,ior             ;  serve per sentire il fronte di discesa
BC_int      ;                         ;  etichetta d'inizio dell'interrupt
            ....                      ;  istruzioni da eseguire con l'interrupt
            RETI                      ;  istruzione fine interrupt
```

Esempio N.10 = Supponiamo di voler **settare** il piedino **B7** come ingresso **con interrupt** che intervenga sul **fronte di discesa** e tutti gli altri piedini della **porta B** come ingresso in **pull-up**. Come prima operazione controlliamo nella **Tabella N.3** come vanno **settati** i registri **pdir - popt - port** per l'ingresso **con interrupt**, dopodiché possiamo scrivere il programma indicato "**Esempio n.10**".

Etichetta A_int

Questa etichetta viene utilizzata per fare eseguire tutte le istruzioni che abbiamo scritto tra **A_int** e **reti** quando sul piedino della porta **A** da noi scelto la tensione passa dal **livello logico 1** al **livello logico 0** (fronte di **discesa**).

Nota: per la funzione **ingresso con interrupt** possiamo abilitare **un solo piedino** della porta **A**.

Esempio N.11 = Ammettiamo di voler **settare** il piedino **A5** come ingresso **con interrupt** che intervenga quando il **livello logico 1** cambia a **livello logico 0**. Come prima operazione controlliamo nella **Tabella N.3** come dobbiamo **settare** i registri **pdir - popt - port** per l'ingresso **con interrupt**, dopodiché possiamo scrivere il programma indicato "**Esempio n.11**".

Etichetta nmi_int

Questa etichetta viene utilizzata per fare eseguire tutte le istruzioni che abbiamo scritto tra **nmi_int** e **reti** quando sul piedino siglato **NMI** (piedino **5** di tutti gli **ST6**) la tensione passa dal **livello logico 1** al **livello logico 0** (fronte di **discesa**). Poiché questo piedino è sempre **abilitato**, non è necessario scrivere nel programma l'istruzione:

```
        SET   4,ior
```

Esempio N.12 = Vogliamo che pigiando il pulsante **P1** (vedi fig.8) si accendano i **diodi led** applicati sui piedini **A0 - A1 - A2 - A3**.

Come prima operazione **settiamo** i piedini **A0 - A1 - A2 - A3** come **uscite push-pull** prelevando dalla **Tabella N.3** i dati da inserire nelle righe **pdir - popt - port** e a questo punto possiamo scrivere il programma indicato "**Esempio n.12**".

NOTA IMPORTANTE: se nelle righe del programma dell'**interrupt** fossero presenti delle istruzioni che utilizzano l'accumulatore **A**, ad esempio:

```
        CPI   a,10   ; confronta A con il numero 10
        ADDI  a,10   ; somma ad A il numero 10
        LDI   a,10   ; metti in A il numero 10
```

bisognerà inserire in una **variabile**, che potremo chiamare **salva**, il valore dell'accumulatore **A** subito dopo l'etichetta dell'**interrupt** e, prima di terminare con l'istruzione **RETI**, lo dovremo reinserire nell'accumulatore **A**.
In questo modo quando il microprocessore tornerà ad eseguire il programma principale, nell'accumulatore si avrà lo stesso valore che c'era prima dell'interrupt.

PROGRAMMA per Esempio n.11

```
        LDI   pdir_a,00000000B   ; abbiamo settato il piedino A5 come ingresso interrupt
        LDI   popt_a,00100000B   ; e tutti gli altri come ingressi pull-up
        LDI   port_a,00000000B   ;
        SET   4,ior              ; serve per abilitare l'interrupt
A_int   ....                     ; etichetta d'inizio dell'interrupt
        ....                     ; istruzioni da eseguire con l'interrupt
        RETI                     ; istruzione fine interrupt
```

PROGRAMMA per Esempio n.12

```
        LDI   pdir_a,00001111B   ; con queste tre righe abbiamo settato come uscite
        LDI   popt_a,00001111B   ; push-pull i piedini A0 - A1 - A2 - A3
        LDI   port_a,00000000B   ;
mni_int ....                     ; etichetta d'inizio dell'interrupt
        SET   0,port_a           ; accendi il led sul piedino A0
        SET   1,port_a           ; accendi il led sul piedino A1
        SET   2,port_a           ; accendi il led sul piedino A2
        SET   3,port_a           ; accendi il led sul piedino A3
        RETI                     ; istruzione fine interrupt
```

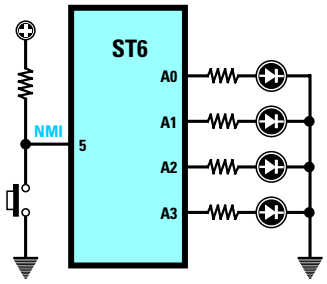


Fig.8 Affinché, pigiando il pulsante posto sul piedino "NMI", si accendano tutti i led applicati sui piedini A0-A1-A2-A3, potrete utilizzare il programma riportato nell'Esempio N.12 o quello dell'Esempio N.13.

Esempio N.13 = Nell'esempio N.12 abbiamo riportato un programma che accendeva contemporaneamente **quattro diodi led** pigiando il pulsante collegato sul piedino **NMI** (vedi fig.8). Usando l'accumulatore **A** si ottiene lo stesso risultato, ma bisogna modificare il programma come indicato nell'"**Esempio n.13**".

A/D CONVERTER

All'interno di ogni microprocessore **ST6** è presente un **A/D converter**, cioè un circuito in grado di convertire una tensione **analogica** compresa tra **0** e **5 volt** in un **numero decimale** compreso tra **0** e **255**.
Per ottenere questa condizione occorre **settare** il piedino della porta in cui viene applicata questa **tensione** come **ingresso analogico**.
Ricordatevi che in tutti i **microprocessori** della serie **ST6** (vedi **Tabelle N.1 - N.2**) **non possono** essere mai utilizzati come **ingressi** per **segnali analogici** i piedini **A0 - A1 - A2 - A3**.
La **massima tensione** che si può applicare su questi ingressi non deve mai **superare** i **5 volt positivi**.
Il valore **numerico** che otteniamo applicando una tensione al piedino che abbiamo settato come **ingresso analogico** si può calcolare con questa formula:

numero decimale = (volt x 255) : 5

Pertanto se applichiamo sul piedino d'ingresso una tensione di **4 volt** otteniamo un **numero decimale** di:

(4 x 255) : 5 = 204

Ricordatevi che l'**A/D converter** fornisce in uscita soltanto dei **numeri interi** quindi se nel risultato sono presenti dei decimali, questi vengono arrotondati al numero intero più prossimo.
Ad esempio, se la tensione di **4 volt** dovesse scendere a **3,98 volt**, sull'uscita non otterremmo:

(3,98 x 255) : 5 = 202,98

ma il numero più prossimo, cioè **203**.

Se la tensione di **4 volt** dovesse salire a **4,04 volt**, sull'uscita non otterremmo:

(4,04 x 255) : 5 = 206,04

ma il numero più prossimo, cioè **206**.

Anche con l'arrotondamento del numero si ottiene sempre un'elevata precisione in quanto la differenza risulta di sole poche **decine** di **millivolt**.
Ad esempio, prendendo sempre la tensione di **4 volt**, per ogni variazione di **0,01 volt** otterremo questi **numeri decimali**:

3,96 volt = 202
3,97 volt = 202
3,98 volt = 203
3,99 volt = 203
4,00 volt = 204
4,01 volt = 205
4,02 volt = 205
4,03 volt = 206
4,04 volt = 206

PROGRAMMA per Esempio n.13

```
        LDI   pdir_a,00001111B  ; con queste tre righe abbiamo settato come uscite
        LDI   popt_a,00001111B  ; push-pull i piedini A0 - A1 - A2 - A3
        LDI   port_a,00000000B  ;
mni_int                         ; etichetta d'inizio dell'interrupt
        LD    salva,a           ; copia nella variabile salva il valore di A
        LDI   a,00001111B       ; carica in A il numero binario 00001111
        LD    port_a,a          ; copia il valore A nel registro della porta a
        LD    a,salva           ; copia nell'accumulatore A il valore di salva
        RETI                    ; istruzione fine interrupt
```

Conoscendo il **numero decimale** è possibile calcolare il valore della tensione in **volt** utilizzando questa formula:

volt = (decimale x 5) : 255

Quindi il **numero decimale 203** corrisponde ad un valore di tensione pari a:

(203 x 5) : 255 = 3,98 volt

con una differenza di **0,01 volt** in più o in meno.
Come abbiamo già detto, un ingresso **settato** per **segnali analogici** può servire soltanto per misurare delle **tensioni continue** che non superino i **5 volt**.
Se la tensione risultasse maggiore, occorrerà ridurla con dei **partitori resistivi**, come in pratica accade in tutti i **tester analogici** che, pur disponendo di uno strumento da **1 volt fondo scala**, possono misurare tensioni anche di **250 - 300 volt**.
E' inoltre possibile misurare delle **tensioni alternate**, se si provvede prima a **raddrizzarle**.
Un ingresso **analogico** può servire per misurare delle **temperature**, delle variazioni di **luce**, degli **ohm** oppure la **reattanza** dei condensatori o delle impedenze, ed anche la **corrente** assorbita da un circuito o la **potenza** di un amplificatore.

IMPORTANTE: Poiché all'interno dei microprocessori **ST6** è presente un **solo A/D converter**, solo **un piedino** può essere adibito a questa funzione. Se per errore vengono settati come **ingressi** per **segnali analogici due piedini**, questi verranno posti in **cortocircuito** ed in questo modo verrà danneggiato il microprocessore.

Esempio N.14 = Vogliamo realizzare un circuito che **accenda** un **diodo led** quando la tensione applicata sul piedino prescelto supera i **3 volt**. Per l'**ingresso** si potrebbe decidere di utilizzare il piedino **B1** e come **uscita** il piedino **A0** (vedi fig.9).

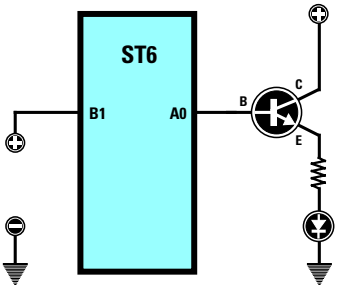


Fig.9 Per accendere il diodo led applicato sul piedino A0 quando la tensione sul piedino B1 supera i 3 Volt, dovrete scrivere il programma posto a fine pagina. Leggere attentamente la soluzione dell'Esempio N.14.

Come si deve scrivere il programma perché il micro esegua questa funzione?
Soluzione: Dalla **Tabella N.3** controlliamo come dobbiamo settare i registri **pdir - popt - port** per predisporre **B1** come **ingresso analogico** e **A0** come **uscita** in **push-pull**.
Successivamente calcoliamo il **numero decimale** di **3 volt** che risulta:

(3 x 255) : 5 = 153

A questo punto possiamo scrivere il nostro programma.

PROGRAMMA per Esempio n.14

```
        LDI   pdir_a,00000001B  ; in queste prime tre righe abbiamo settato la porta
        LDI   popt_a,00000001B  ; A0 come uscita in push-pull
        LDI   port_a,00000000B  ;
        LDI   pdir_b,00000000B  ; in queste tre righe abbiamo settato la porta
        LDI   popt_b,00000010B  ; B1 come ingresso analogico
        LDI   port_b,00000010B  ;
ripeti  LDI   wdog,255          ; carichiamo il watchdog
        LDI   adcr,00110000B    ; provvedi a convertire da analogico a digitale
attendi JRR   6,adcr,attendi    ; attendere che avvenga la conversione A/D
        LD    a,addr            ; carica nell'accumulatore A il numero digitale
        CPI   a,153             ; compara il valore di A con il numero 153
        JRNC  accendi           ; se A è maggiore di 153 salta all'etichetta accendi
        JRC   spegni            ; se A è minore di 153 salta all'etichetta spegni
        JP    ripeti            ; salta all'etichetta ripeti del watchdog
accendi SET   0,port_a          ; setta l'uscita del piedino A0 a 5 volt
        JP    ripeti            ; salta all'etichetta ripeti del watchdog
spegni  RES   0,port_a          ; setta l'uscita del piedino A0 a 0 volt
        JP    ripeti            ; salta all'etichetta ripeti del watchdog
```

Esempio N.15 = Vogliamo accendere una fila di **5 diodi led**, ma in modo che con **1 volt** si accenda un **solo led**, con **2 volt** si accendano **2 led**, con **3 volt** si accendano **3 led** ecc., fino a far accendere tutti i **5 diodi led** quando la tensione raggiunge i **5 volt**.
Come **ingresso analogico** abbiamo deciso di scegliere il piedino **A7** e come **uscite** i piedini da **B0** a **B4**.

Soluzione: La prima operazione che dobbiamo compiere è quello di calcolare il **numero decimale** corrispondente ai valori di tensione di **1 - 2 - 3 - 4 - 5 volt** usando la formula che già conosciamo:

(1 x 255) : 5 = 51 numero decimale di 1 Volt
(2 x 255) : 5 = 102 numero decimale di 2 Volt
(3 x 255) : 5 = 153 numero decimale di 3 Volt
(4 x 255) : 5 = 204 numero decimale di 4 Volt
(5 x 255) : 5 = 255 numero decimale di 5 Volt

A questo punto possiamo programmare il piedino **A7** come **ingresso analogico** ed i piedini **B0 - B1 - B2 - B3 - B4** come **uscite push-pull** (vedi **Tabella N.3**).

Se anziché accendere tutta la fila dei **diodi led** volessimo accendere **un solo** diodo led per volta, cioè prima quello su **B0**, poi quello su **B1 - B2 - B3 - B4**, dovremmo modificare tutte le righe delle **etichette LED** mettendo un **1** solo sul piedino a cui è collegato il led che vogliamo accendere e degli **0** sui piedini a cui sono collegati i led che vogliamo spegnere.

Nell'esempio che si trova a fine pagina dovremmo riscrivere le sole righe **LED2 - LED3 - LED4 - LED5** in questo modo:

```
LED2  LDI  port_b,00000010B
LED3  LDI  port_b,00000100B
LED4  LDI  port_b,00001000B
LED5  LDI  port_b,00010000B
```

PROGRAMMA per Esempio n.15

```
        LDI   pdir_a,00000000B   ; in queste tre righe abbiamo settato
        LDI   popt_a,10000000B   ; il piedino A7 come ingresso analogico
        LDI   port_a,10000000B   ;
        LDI   pdir_b,00011111B   ; in queste righe abbiamo settato
        LDI   popt_b,00011111B   ; i piedini da B0 a B4 come uscite
        LDI   port_b,00000000B   ;
ripeti  LDI   wdog,255           ; carichiamo il watchdog
        LDI   adcr,00110000B     ; provvedi a convertire da analogico a digitale
attendi JRR   6,adcr,attendi     ; attendere che avvenga la conversione A/D
        LD    a,addr             ; carica nell'accumulatore A, il numero digitale
        CPI   a,255              ; compara il valore di A con il numero 255
        JRNC  LED5               ; se A è uguale a 255 salta all'etichetta LED5
        CPI   a,204              ; compara il valore di A con il numero 204
        JRNC  LED4               ; se A è maggiore di 204 salta all'etichetta LED4
        CPI   a,153              ; compara il valore di A con il numero 153
        JRNC  LED3               ; se A è maggiore di 153 salta all'etichetta LED3
        CPI   a,102              ; compara il valore di A con il numero 102
        JRNC  LED2               ; se A è maggiore di 102 salta all'etichetta LED2
        CPI   a,51               ; compara il valore di A con il numero 51
        JRNC  LED1               ; se A è maggiore di 51 salta all'etichetta LED1
        JP    LED0               ; se A è minore di 51 salta all'etichetta LED0
LED0    LDI   port_b,00000000B   ; non accendere nessun diodo led
        JP    ripeti             ; salta all'etichetta ripeti del watchdog
LED1    LDI   port_b,00000001B   ; accendi il led sul piedino B0
        JP    ripeti             ; salta all'etichetta ripeti del watchdog
LED2    LDI   port_b,00000011B   ; accendi i led sui piedini B0 - B1
        JP    ripeti             ; salta all'etichetta ripeti del watchdog
LED3    LDI   port_b,00000111B   ; accendi i led sui piedini B0 - B1 - B2
        JP    ripeti             ; salta all'etichetta ripeti del watchdog
LED4    LDI   port_b,00001111B   ; accendi i led sui piedini B0 - B1 - B2 - B3
        JP    ripeti             ; salta all'etichetta ripeti del watchdog
LED5    LDI   port_b,00011111B   ; accendi i led sui piedini B0 - B1 - B2 - B3 - B4
        JP    ripeti             ; salta all'etichetta ripeti del watchdog
```

Questa modifica potrebbe risultare utile se invece di accendere dei **diodi led** volessimo **eccitare** cinque diversi **relè** per ogni diverso valore di tensione, ad esempio, il **relè1** quando la tensione raggiunge **1 volt**, il **relè 2** quando la tensione raggiunge i **2 volt** ecc.

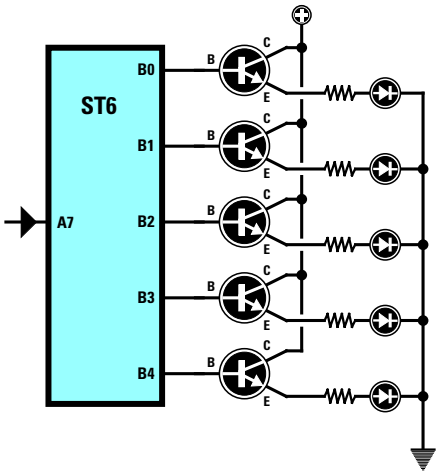


Fig.10 Volendo realizzare un Level-Meter che accenda una fila di 4 led all'aumentare della tensione continua, applicata sul piedino A7 settato come Ingresso, dovrete utilizzare il programma dell'Esempio N.15 riportato sulla pagina in basso a sinistra.

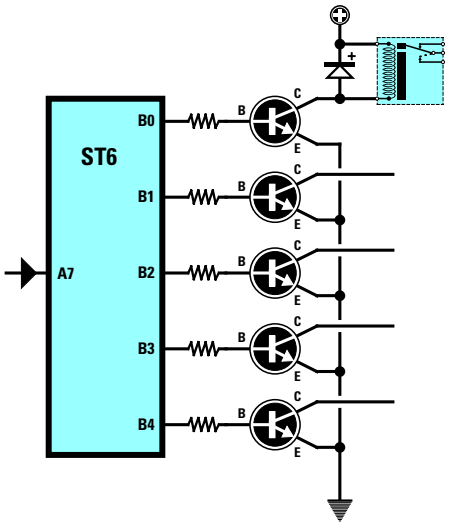
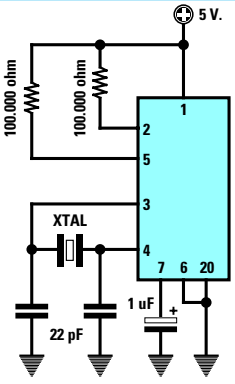
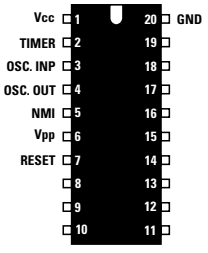


Fig.11 Se vorrete eccitare dei relè dovrete modificare lo schema come qui sopra riportato. Per motivi di spazio abbiamo inserito il relè solo sul primo transistor in alto. Modificando il programma potrete utilizzare come uscite anche tutti gli 8 piedini della Porta B.

Fig.12 Per far funzionare un qualsiasi micro ST6 della serie 10-20 dovrete collegare sui piedini indicati un Quarzo da 8 MHz, due condensatori ceramici da 22 pF, un elettrolitico da 1 microF e due resistenze da 100.000 ohm.



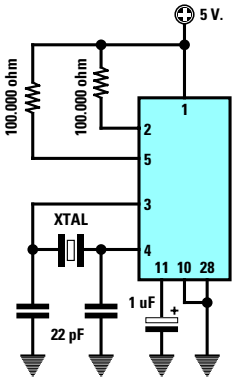


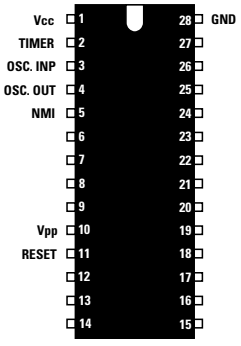
ST 62 E 10

ST 62 T 10

ST 62 E 20

ST 62 T 20





ST 62 E 15

ST 62 T 15

ST 62 E 25

ST 62 T 25

Fig.13 Per far funzionare un qualsiasi micro ST6 della serie 15-25 dovrete collegare sui piedini indicati un Quarzo da 8 MHz, due condensatori ceramici da 22 pF, un elettrolitico da 1 microF e due resistenze da 100.000 ohm.

TIMER

Il **timer** è un **contatore** collegato al **quarzo** del microprocessore tramite un **prescaler** ed un **divisore x 12** (vedi fig.14).
Per la funzione **timer** dobbiamo settare due registri, uno chiamato **tcr** e l'altro **tscr**.
Nel registro **tcr** (timer counter register) dovremo inserire un **numero decimale** compreso tra **1** e **255**.
Il **tcr** partendo da questo numero **conterà all'indietro** e quando arriverà al numero **0** automaticamente eseguirà tutte le istruzioni comprese tra l'etichetta **tim_int** e l'istruzione **reti**.
Ammesso che si voglia inserire nel registro **tcr** il numero **255** dovremmo scrivere questa riga di programma:

```
        LDI   tcr,255
```

Nel registro **tscr** (timer status control register), che controlla il **prescaler**, dovremo inserire un **numero binario** come riportato nella **Tabella N.5**.

TABELLA N.5 Registro tscr

01011000	divide x	1
01011001	divide x	2
01011010	divide x	4
01011011	divide x	8
01011100	divide x	16
01011101	divide x	32
01011110	divide x	64
01011111	divide x	128

Ammesso di voler far dividere il **prescaler** per **128** dovremo scrivere questa riga di programma:

```
        LDI   tscr,01011111B
```

Alla fine della riga non dobbiamo dimenticarci di inserire una **B**, perché questo è un **numero binario**.
Per calcolare il **tempo** in **secondi** possiamo usare questa formula:

secondi = (12 x tscr x tcr) : Xtal in Hz

Ammesso che si usi un quarzo da **8 MHz** (pari a **8.000.000 Hz**), che nella riga **tscr** si sia inserito il numero **128** e nella riga **tcr** il numero **255**, otterremo un tempo **massimo** di:

(12 x 128 x 255) : 8.000.000 = 0,0489 secondi

che corrispondono a **48,9 millisecondi**.

Tempi così ridotti potrebbero servire soltanto per realizzare dei **generatori di onde quadre**, ma certo non dei **timer** dove normalmente occorrere raggiungere dei tempi di **minuti** o **ore**.
Per ottenere dei **tempi molto lunghi** possiamo usare degli accorgimenti come per esempio ricorrere all'uso di altre variabili.

Esempio N.16 = Vorremmo prelevare dal piedino **A7** degli **impulsi** di **1 millisecondo**, quindi vorremmo sapere come impostare il programma.

Soluzione: Come prima operazione convertiamo i **millisecondi** in **secondi** dividendoli per **1.000** e così otteniamo:

1 : 1.000 = 0,001 secondo

Quindi calcoliamo quale numero dobbiamo mettere nel **registro tcr** con la formula:

tcr = [(Xtal Hz : 12) : tscr] x secondi

Tenete presente che il numero del **tcr** non deve mai risultare **maggiore** di **255** quindi se questo si

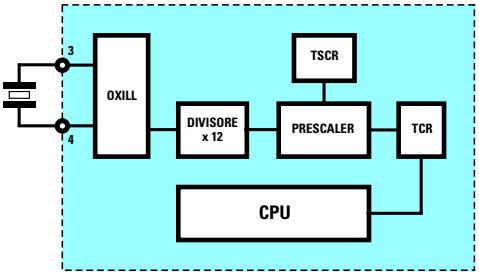


Fig.14 All'interno di ogni microprocessore c'è un Timer che preleva dall'oscillatore quarzato la frequenza generata e la DIVIDE subito x 12. Per ricavare tempi diversi si deve agire solo sui due registri chiamati TCSR e TCR. Nell'articolo trovate due Esempi, uno ha il numero 16 e l'altro il 17.

verificasse dovrete **aumentare** il numero di divisione del **prescaler**.
Questo numero, che possiamo prelevare dalla **Tabella N.5**, è **1 - 2 - 4 - 8 - 16 - 32 - 64 - 128**.
Ad esempio se scegliamo per **tscr** il numero **2** otteniamo un **tcr** di:

[(8.000.000 : 12) : 2] x 0,001 = 333,33 tcr

poiché questo numero è **maggiore** di **255** dovremo scegliere per **tscr** un numero **maggiore**, ad esempio **4** e così otterremo:

[(8.000.000 : 12) : 4] x 0,001 = 166,66 tcr

Poiché il microprocessore lavora solo con **numeri interi**, dovremo arrotondarlo sul numero più prossimo che nel nostro caso è **167**.
Noi abbiamo scelto per **tscr** il numero **4**, ma potevamo anche scegliere **8 - 16 - 32 ecc.** tenendo comunque presente che più alto è il numero del **tscr** minore risulterà la **precisione** sul tempo.
Disponendo dei valori richiesti cioè:

tscr = 4
tcr = 167

possiamo calcolare il **tempo** con la formula:

secondi = (12 x tscr x tcr) : Xtal in Hz

ottenendo:

(12 x 4 x 167) : 8.000.000 = 0,001002 secondi

che corrispondono a:

0,001002 x 1.000 = 1,002 millisecondi

Il programma che dobbiamo scrivere per prelevare dal piedino **A7** questi impulsi è visibile in fondo alla pagina.

Esempio N.17 = Vogliamo prelevare dal piedino **A7** delle **onde quadre** che abbiano una frequenza di **1.200 Hz**, quindi vogliamo sapere come impostare il programma.

Soluzione: Come prima operazione dobbiamo calcolare il **tempo** in **secondi** corrispondente alla frequenza di **1.200 Hz** e per ottenere questo dato usiamo la formula:

secondi = (1 : Hz) : 2

Nel nostro caso otteniamo:

(1 : 1.200) : 2 = 0,0004 secondi

Quindi calcoliamo qual è il numero che dobbiamo mettere nel **registro tcr** con la formula:

tcr = [(Xtal Hz : 12) : tscr] x secondi

Poiché nella formula manca il valore del **tscr**, consultiamo la **Tabella N.5** scegliendo uno di questi numeri **1 - 2 - 4 - 8 - 16 - 32 - 64 - 128**.
Facciamo presente che il valore di **tcr** che ricaveremo da questa formula non dovrà mai superare il numero **255** quindi se risultasse maggiore dovremo usare un **tscr** maggiore, cioè **4 - 8 - 16** ecc. (vedi **Tabella N.5**).
Nel nostro esempio abbiamo scelto per **tscr** il numero **2** perché otteniamo un valore **minore** di **255**.

[(8.000.000 : 12) : 2] x 0,0004 = 133,33 tcr

PROGRAMMA per Esempio n.16

```
        LDI   pdir_a,10000000B  ; queste tre righe servono per settare
        LDI   popt_a,10000000B  ; il piedino A7 come uscita in push-pull
        LDI   port_a,00000000B  ;
        SET   4,ior             ; abilita l'interrupt quando il tcr diventa 0
        LDI   tcr,167           ; numero 167 calcolato per il tcr
        LDI   tscr,01011010B    ; numero binario per un fattore di divisione di 4 (Tabella N.5)
main    LDI   wdog,255          ; ricarichiamo il watchdog
        JP    main              ; salta all'etichetta main
tim_int LDI   wdog,255          ; ricarichiamo il watchdog
        LDI   tcr,167           ; ricarichiamo 167 nel tcr per ripetere gli impulsi
        LDI   tscr,01011010B    ; questa riga fa ripartire il contatore
        SET   7,port_a          ; fa uscire dal piedino A7 un impulso a 5 volt
        RES   7,port_a          ; riporta il piedino A7 a 0 volt
        RETI                    ; ritorna al programma main
```

Poiché il microprocessore lavora solo con **numeri interi** dovremo arrotondarlo sul numero più prossimo che nel nostro esempio è **133**.
Disponiamo così di tutti i dati richiesti:

Per prelevare dal piedino **A7** delle **onde quadre** che abbiano una frequenza di **1.200 Hz**, dovremo scrivere il programma come visibile qui sotto.

tscr = 2
tcr = 133

PROGRAMMA per Esempio n.17

```
        LDI   pdir_a,10000000B  ; queste tre righe ci servono per settare
        LDI   popt_a,10000000B  ; il piedino A7 come uscita in push-pull
        LDI   port_a,00000000B  ;
        SET   4,ior             ; abilita l'interrupt
        LDI   tcr,133           ; carica nel tcr il numero 133
        LDI   tscr,01011001B    ; numero binario per un fattore di divisione di 2 (Tabella N.5)
main    LDI   wdog,255          ; ricarica il watchdog
        JP    main              ; salta all'etichetta main
tim_int                         ; etichetta dell'interrupt
        LDI   wdog,255          ; ricarica il watchdog
        LDI   tcr,133           ; ricarica 133 nel tcr per continuare
        LDI   tscr,01011001B    ; questa riga fa ripartire il contatore
        JRR   7,port_a,salita   ; se A7 è a 0 salta all'etichetta salita
        RES   7,port_a          ; se A7 è a 1 cambia il livello logico a 0
        JP    continua          ; salta all'etichetta continua
salita  SET   7,port_a          ; metti il piedino A7 a livello logico 1
continua RETI                   ; fine dell'interrupt
```

BUS per

Con il microprocessore ST6 si possono realizzare un'infinità di circuiti, come ad esempio orologi, contasecondi, timer, antifurto, controlli numerici per macchine utensili, termostato, piccoli robot, comandi per luci, termometri, inoltre si possono scrivere delle parole sui display LCD, convertire dei calcoli ecc., e qui ci fermiamo perché volendo elencare tutto ci vorrebbero non poche pagine della rivista.

Anche se nelle riviste precedenti vi abbiamo spiegato le istruzioni per scrivere un programma, per chi è all'inizio queste informazioni potrebbero risultare ancora **insufficienti**.

Infatti chi volesse realizzare un orologio a **display** potrebbe trovarsi in difficoltà nel far apparire i numeri delle **ore** e dei **minuti**. Se poi in sostituzione di un display a sette segmenti si volesse utilizzare un display **LCD**, non si saprebbe quali modifiche apportare al software.

Chi vuole **eccitare** un relè ad una determinata **ora** si chiederà invece quale istruzione gli permette di ottenere questa condizione.

Anche ammesso di aver scritto tutte le istruzioni richieste, il lettore desidera giustamente sapere se il **suo programma** è in grado di eseguire senza errori tutte le funzioni per cui è stato scritto, ha cioè bisogno di **testarlo** e per questo gli servono delle **schede di test** universali.

Sono proprio queste che prenderemo in esame con

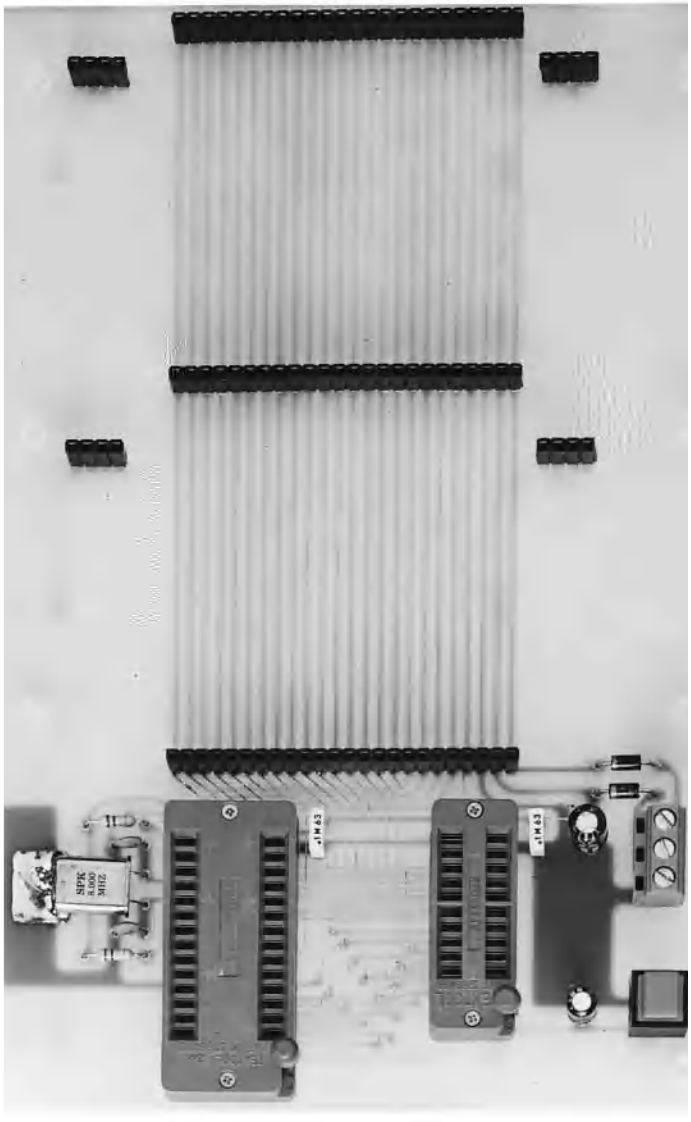


Fig.1 Foto del bus progettato per testare i micro ST6.
I due costosi zoccoli textool possono essere sostituiti con due economici zoccoli per integrati, come visibile nel disegno pratico di fig.3.

TESTARE i micro ST6

Il primo problema che si presenta a quanti desiderano iniziare a scrivere del software personalizzato per i microprocessori ST6 della SGS, è quello di poter controllare il programma per verificare che esegua le funzioni richieste. Per aiutarvi abbiamo progettato delle schede sperimentali, che saranno particolarmente utili agli Istituti Tecnici se usate come supporto al consueto materiale didattico.

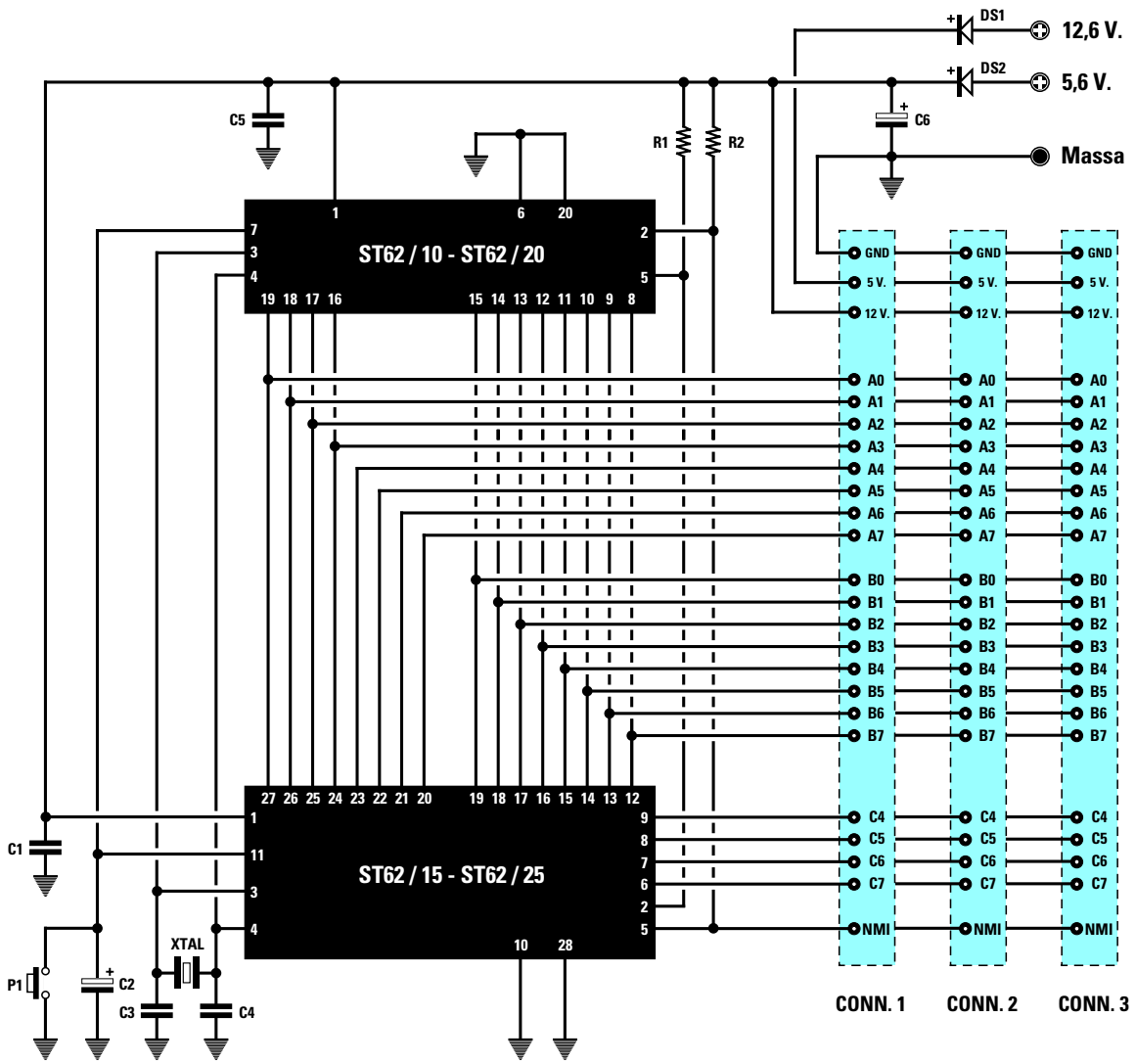
il nostro articolo.
A parte vi forniamo un **dischetto** contenente alcuni programmi completi di esaurienti **commenti**, che potranno servirvi per realizzare orologi, timer, contasecondi, antifurto ecc., e con riportate tutte le modifiche che si possono apportare.
Come sempre i maligni penseranno che il nostro obiettivo sia solo quello di vendere al lettore un **dischetto**, ma essi non considerano che riempire **10 pagine** della rivista con sole righe di programma **non risulta** per nulla gradito a coloro ai quali **non** interessa l'**ST6**.
Inoltre non pensano che nel trascrivere i programmi sulla rivista si possono verificare degli **errori di stampa**, ed altri errori possono commetterli gli stessi lettori nel ricopiare le istruzioni.
Disponendo di un **dischetto** con programmi già testati, il lettore potrà subito metterli in funzione e poi modificarli secondo le proprie esigenze.
Se con le modifiche apportate il programma darà qualche **errore**, sarà sempre possibile fare un controllo con il **programma originale** per verificare dove è stato commesso l'errore.

IMPORTANTE

La **SGS** ci ha comunicato che cesserà di produrre i microprocessori **riprogrammabili** tipo **ST62E10** e tipo **ST62E15**, entrambi da **2 K** di **memoria**, perché tutte le Industrie chiedono e preferiscono utilizzare i **riprogrammabili** da **4 K** di **memoria** anche se più costosi.
I microprocessori **non riprogrammabili** tipo **ST62T10 - ST62T15** con **2 K** di memoria rimarranno invece sempre in produzione.
Pertanto **fuori produzione** andranno i tipi:

ST62E10 che verranno sostituiti dagli **ST62E20**
ST62E15 che verranno sostituiti dagli **ST62E25**

Poiché le dimensioni e la piedinatura dei due mo-



ELENCO COMPONENTI LX.1202

R1 = 100.000 ohm 1/4 watt
R2 = 100.000 ohm 1/4 watt
C1 = 100.000 pF poliestere
C2 = 1 mF elettr. 63 volt
C3 = 22 pF ceramico
C4 = 22 pF ceramico
C5 = 100.000 pF poliestere
C6 = 100 mF elettr. 35 volt
XTAL = quarzo 8 MHz
DS1 = diodo 1N.4007
DS2 = diodo 1N.4007
CONN.1-2-3 = connettori 24 poli
P1 = pulsante

Fig.2 Schema elettrico del circuito bus progettato per testare i programmi per i microprocessori ST6. Anche se nel bus sono presenti due zoccoli, dovrete sempre utilizzarne uno SOLO alla volta, quindi prima di inserire un micro in uno zoccolo dovrete togliere quello presente sull'altro zoccolo.
Il bus andrà alimentato con lo stadio di alimentazione visibile nelle figg.4-7 cercando di non invertire i due fili dei 5,6 e 12,6 volt.

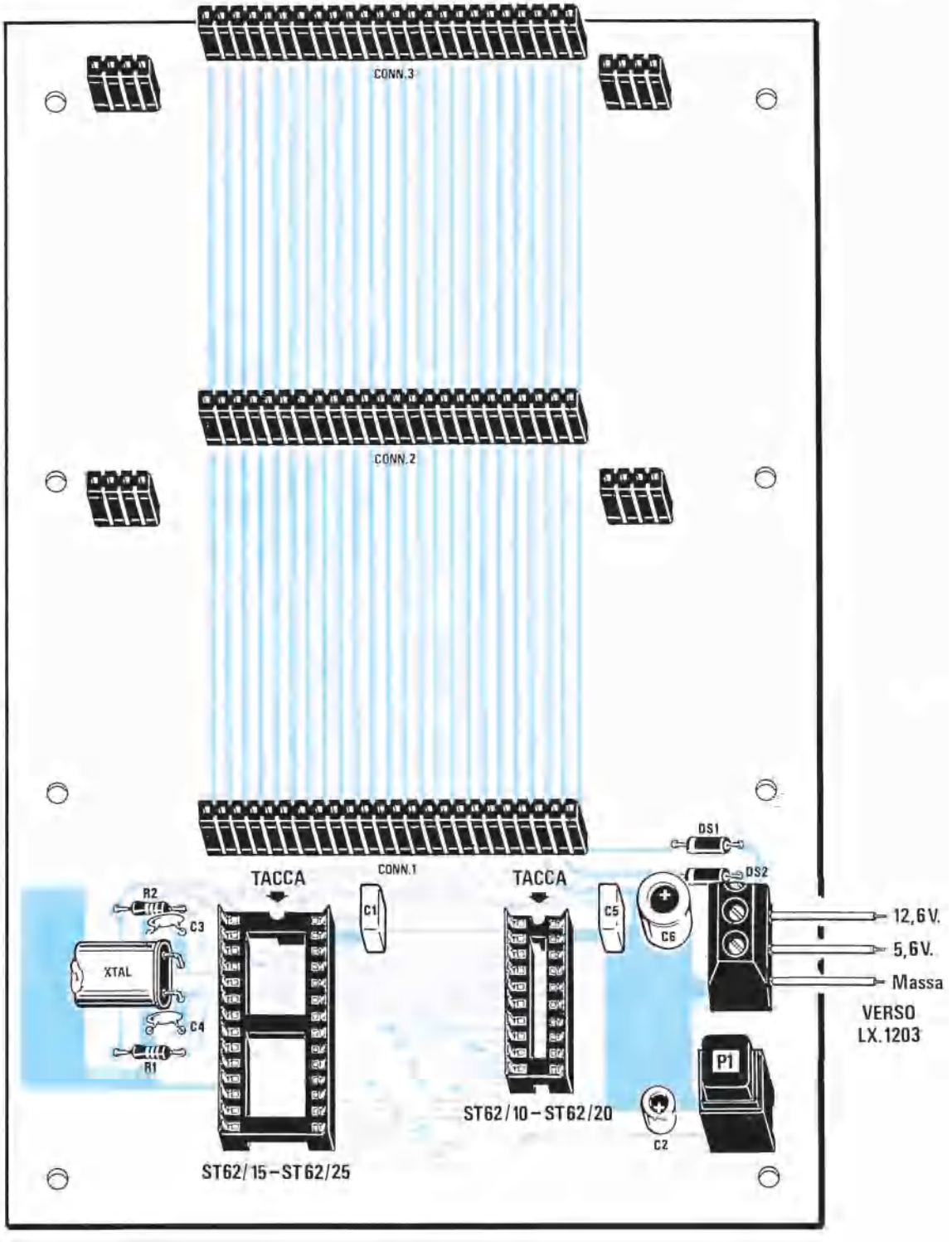


Fig.3 Schema pratico di montaggio del bus. Potete sostituire i due comuni zoccoli per i microprocessori ST6 con i più comodi, ma costosi textool (vedi fig.1).

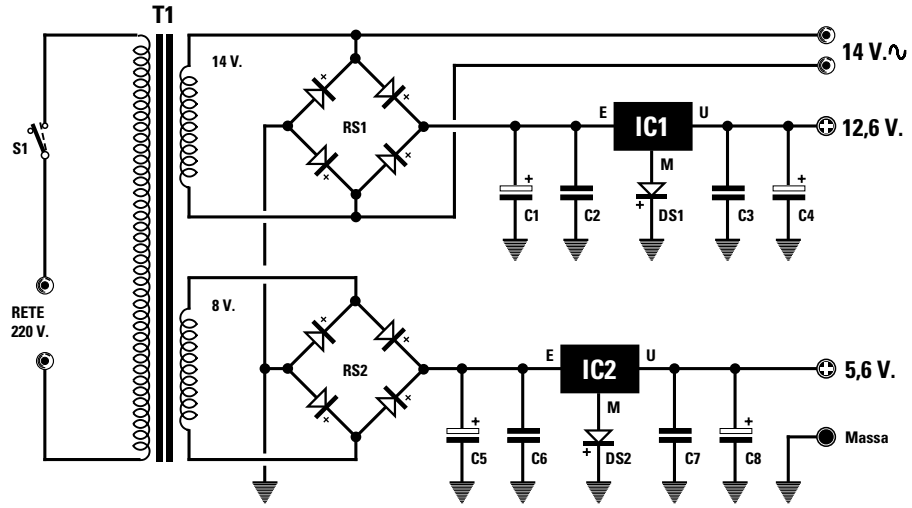


Fig.4 Schema elettrico dello stadio di alimentazione da utilizzare per il bus.

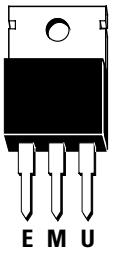
delli risultano identiche sarà possibile sostituirli senza problemi.
Il motivo per cui tutti preferiscono i microprocessori **riprogrammabili** da **4 K** è ovvio.
I microprocessori **riprogrammabili** servono e vengono utilizzati unicamente per **testare** i programmi e quindi possono essere **cancellati** e riutilizzati per provare altri programmi.
Dopo aver verificato che il programma funziona, si può definitivamente trasferirlo sui microprocessori **non riprogrammabili** tipo **ST62T10 - ST62T20** o **ST62T15 - ST62T25**.
Poiché un microprocessore **riprogrammabile** viene riutilizzato un **centinaio** di volte, si preferisce acquistarne uno da **4 K**, perché può essere usato sia per i programmi che occupano **1 - 1,5 - 2 K** sia per quelli che occupano **2,5 - 3 - 4 K**.

ELENCO COMPONENTI LX.1203

C1 = 2.200 mF elettr. 35 volt
C2 = 100.000 pF poliestere
C3 = 100.000 pF poliestere
C4 = 100 mF elettr. 35 volt
C5 = 2.200 mF elettr. 35 volt
C6 = 100.000 pF poliestere
C7 = 100.000 pF poliestere
C8 = 100 mF elettr. 35 volt
DS1 = diodo 1N.4007
DS2 = diodo 1N.4007
RS1 = ponte raddr. 100 V. 1 A.
RS2 = ponte raddr. 100 V. 1 A.
IC1 = uA.7812
IC2 = uA.7805
T1 = trasformatore 25 watt (T025.01)
sec. 14 V. 1 A. - 8 V. 1 A.
S1 = interruttore

SCHEMA ELETTRICO scheda BUS

Per testare i programmi abbiamo realizzato una scheda in cui si possono utilizzare sia i microprocessori da 20 piedini sia quelli da 28 piedini.
Per questa scheda abbiamo inoltre realizzato un **bus** che porta tutti i segnali del microprocessore ai connettori femmina sui quali potrete inserire diversi tipi di schede, ad esempio con dei **display** a **7 segmenti**, oppure con un **display LCD** o con dei **relè** o ancora con dei **Triac**.



uA 7805
uA 7812

Fig.5 Connessioni dei due integrati stabilizzatori di tensione.

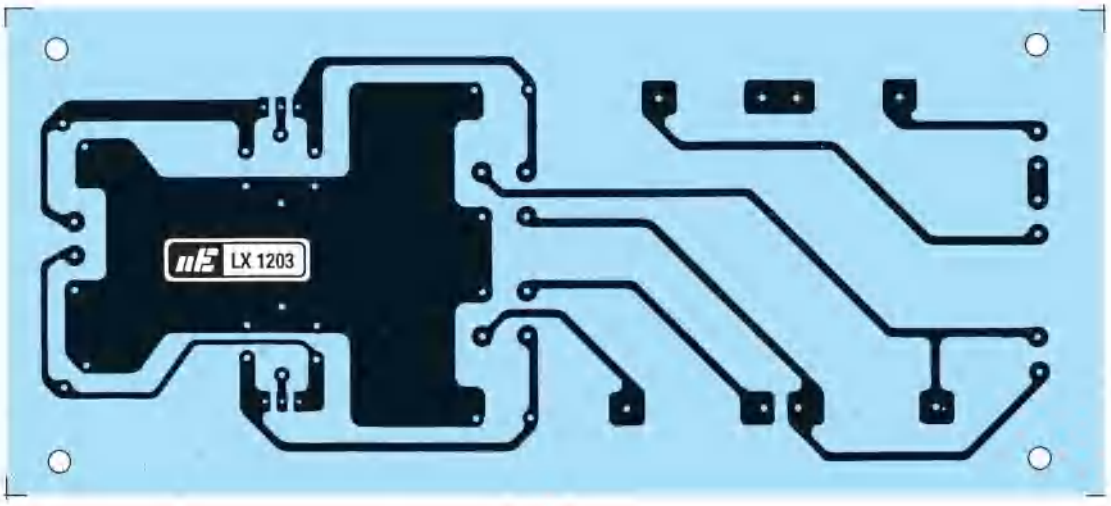


Fig.6 Disegno del circuito stampato dello stadio di alimentazione siglato LX.1203 visto dal lato rame. Nota: il disegno è stato leggermente ridotto per farlo rientrare nella pagina; le sue misure reali sono Lunghezza = mm 160 ed Altezza = mm 70.

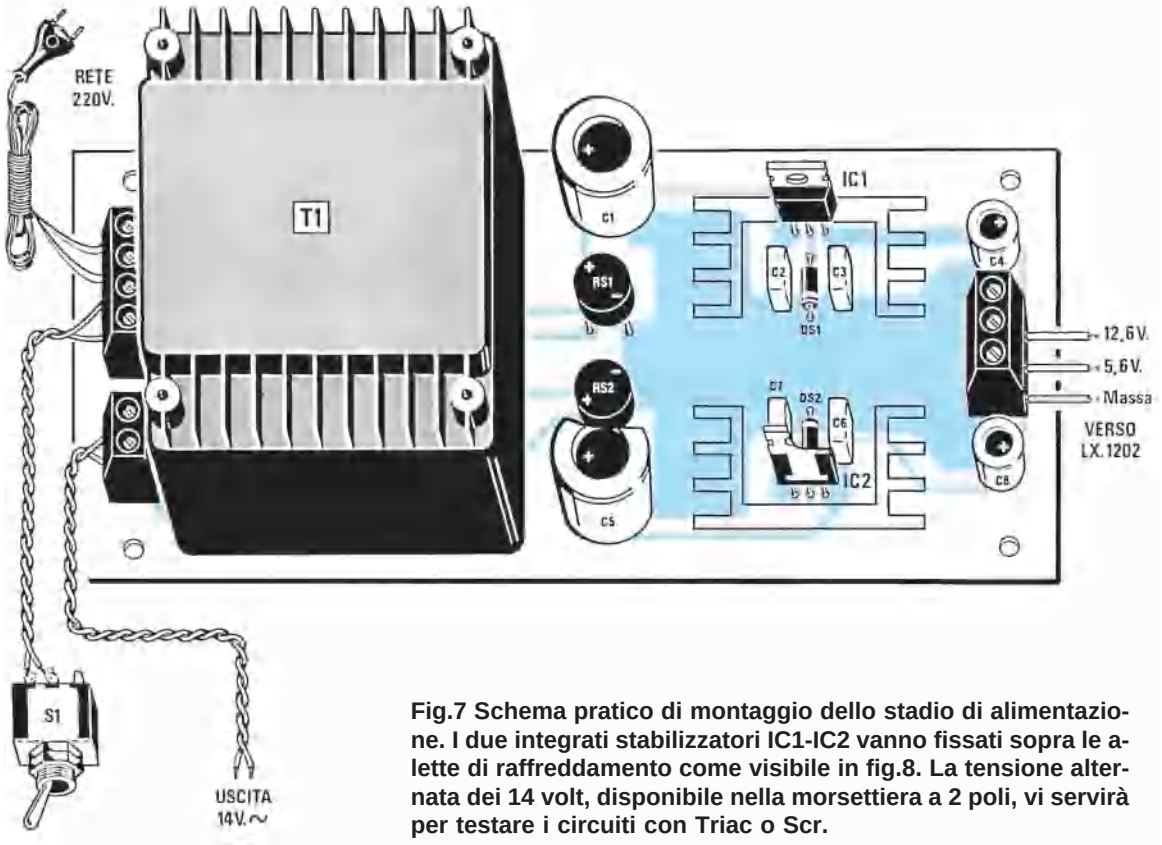


Fig.7 Schema pratico di montaggio dello stadio di alimentazione. I due integrati stabilizzatori IC1-IC2 vanno fissati sopra le alette di raffreddamento come visibile in fig.8. La tensione alternata dei 14 volt, disponibile nella morsettiera a 2 poli, vi servirà per testare i circuiti con Triac o Scr.

Per la descrizione dello schema elettrico, visibile in fig.2, iniziamo proprio dai due zoccoli, che, ovviamente, dovranno essere utilizzati **solo uno** per volta.

Dunque se avete già inserito un microprocessore in uno zoccolo, per inserirne un altro nel secondo zoccolo dovrete **togliere** il primo.
Tutti i terminali dei due zoccoli sono collegati tra loro in modo da sfruttare per entrambi lo stesso **quarzo** per il **clock**, lo stesso pulsante di **reset** ed ovviamente la stessa alimentazione.
Anche tutte le porte d'ingresso/uscita sono collegate in **parallelo** e qui è necessario ricordare che nello zoccolo che riceverà i microprocessori **ST62E10 - ST62E20** la **porta A** inizia da **A0** e termina ad **A3**, la **porta B** inizia da **B0** e termina a **B7**, e che manca la **porta C**.
Nello zoccolo che riceverà i microprocessori **ST62E15 - ST62E25** risultano presenti tutte le **porte** da **A0** ad **A7** e da **B0** a **B7**, e la **porta C** inizia da **C4** e termina a **C7**.

Tutti gli **ingressi/uscite** raggiungono i connettori femmina **CONN.1 - CONN.2 - CONN.3** nei quali andranno inserite le **schede sperimentali**.
Sempre sugli stessi **connettori** risultano presenti le piste di alimentazione, cioè **5,6 volt positivi - 12,6 volt positivi** e la **massa**.
I diodi al silicio **DS1 - DS2**, posti in serie sui due ingressi di alimentazione, sono stati inseriti per evitare di danneggiare il **micro** nel caso venisse applicata su questi terminali una polarità opposta a quella richiesta.
Poiché questi diodi introducono una tensione di circa **0,6 volt**, applicando sull'ingresso una tensione di **5,6 volt** e **12,6 volt**, in uscita si otterranno esattamente **5 volt** e **12 volt**.

REALIZZAZIONE PRATICA

Sul circuito stampato siglato **LX.1202** dovete montare i pochi componenti visibili in fig.3.
Come noterete, nel kit abbiamo inserito due zoccoli, uno da **20** e l'altro da **28** piedini, ma qui dobbiamo aprire una piccola parentesi.
Poiché questo progetto verrà utilizzato anche da piccole e medie Industrie per **testare** i loro microprocessori, noi consigliamo di utilizzare, in sostituzione degli zoccoli inseriti nel kit, degli zoccoli **textool** provvisti di una levetta di bloccaggio (vedi fig.1).
Usando questi zoccoli risulterà più semplice e veloce inserire e togliere i microprocessori, ma questo vantaggio costerà 55.000 lire in più.

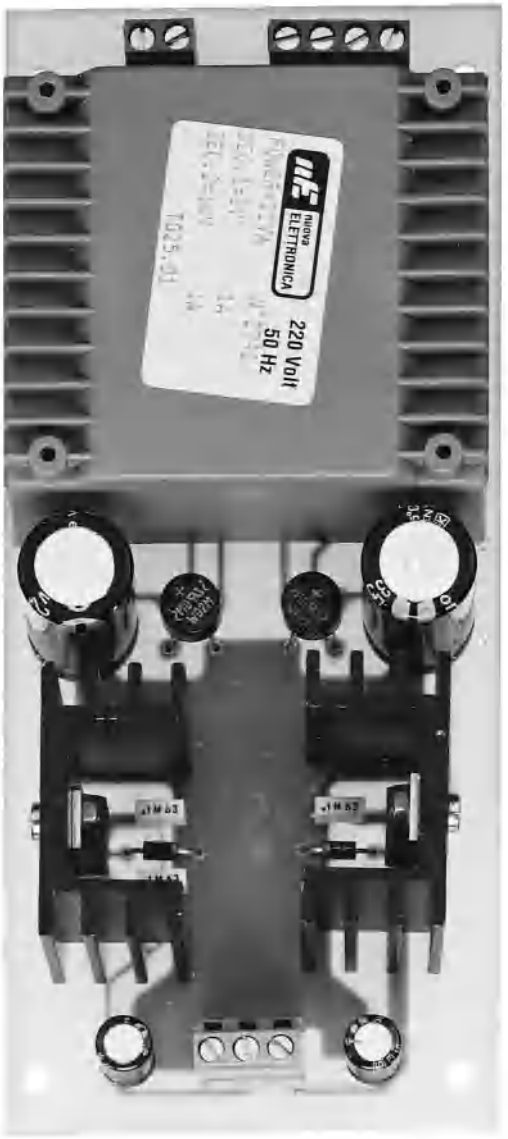


Fig.8 Come si presenta a montaggio ultimato lo stadio di alimentazione che vi fornisce tutte le necessarie tensioni per alimentare il nostro bus.
Vi consigliamo di racchiudere l'alimentatore dentro un mobile plastico e a questo proposito vi indichiamo il mobile codificato MTK06.22 del costo di € 6,97

Come visibile in fig.3, sullo stampato dovrete montare i due zoccoli, il pulsante di **reset**, il **quarzo**, i **connettori** per le schede sperimentali ed i pochi componenti passivi, cioè resistenze, condensatori ed i due diodi al silicio, rispettando il verso della **fascia bianca** riportata sul loro corpo.
I quattro connettori femmina a **4 terminali**, che sullo stampato risultano isolati dal circuito, serviranno come **punto di appoggio** per le schede sperimentali che inserirete nel **bus**.
Questi connettori vi serviranno anche per evitare di inserire le schede sperimentali in senso inverso al richiesto.

ALIMENTAZIONE

Le tensioni richieste per alimentare questo **bus** devono risultare, come già anticipato, di **5,6** e di **12,6 volt**.
Come visibile in fig.4, le due tensioni prelevate dai due secondari del trasformatore **T1** vengono raddrizzate da **RS1** ed **RS2**, poi stabilizzate a **5,6 volt** dall'integrato **uA.7805** (vedi **IC2**) e a **12,6 volt** dall'integrato **uA.7812** (vedi **IC1**).
Per prelevare sull'uscita di questi due integrati una tensione di **5,6 volt** anziché di **5 volt** ed una tensione di **12,6 volt** anziché di **12 volt**, abbiamo inserito tra il terminale **M** e la **massa** un diodo al silicio (vedi **DS1 - DS2**).
Dallo stesso trasformatore si preleva anche una **tensione alternata** di circa **14 volt**, che potrà servire per testare i circuiti che utilizzano dei **Triac**.
Anche se per queste prove è possibile applicare sulla loro uscita una tensione di **220 volt alternati**, noi ve lo **sconsigliamo**, perché se inavvertitamente toccaste con le mani le **piste** del circuito stampato potrebbe risultare **molto pericoloso**.

REALIZZAZIONE PRATICA ALIMENTATORE

Sul circuito stampato monofaccia siglato **LX.1203** monterete tutti i componenti visibili in fig.7 cercando come sempre di rispettare la polarità dei terminali dei diodi al silicio, dei ponti raddrizzatori e dei condensatori elettrolitici.
Come potete osservare anche dalle foto, i due integrati stabilizzatori vanno fissati sopra due alette di raffreddamento.
Per le tensioni d'uscita inserite una **morsettiera** a **3 poli** dalla quale potrete prelevare le tensioni **stabilizzate** di **12,6 volt - 5,6 volt** più il filo della **massa**, ed una **morsettiera** a **2 poli** dalla quale potrete prelevare una tensione **alternata** di circa **14 volt** che vi servirà per collaudare i programmi che "pilotano" i diodi **Triac**.
Vi conviene racchiudere l'alimentatore dentro un qualsiasi mobile e a tal proposito vi consigliamo il mobile siglato **MTK06.22.**
Per le tensioni d'uscita utilizzate dei fili di diverso **colore** così da poter subito stabilire il valore della tensione presente e non correre il rischio di invertirli quando li collegherete alla morsettiera del **bus**.
Tanto per fare un esempio, per la **massa** potrete scegliere il colore **nero**, per i **5,6 volt** il colore **giallo** o **marrone** e per i **12,6 volt** il colore **rosso** o **arancio**.
Per i due fili dell'**alternata** potrete usare due fili **bianchi** oppure di un colore completamente diverso da quello scelto per le altre uscite.

Per le schede **sperimentali** vi rimandiamo all'articolo pubblicato su questa stessa rivista.

COSTO DI REALIZZAZIONE

Il solo Bus siglato LX.1202 completo di circuito stampato, quarzo, zoccoli normali, connettori, pulsante, cioè tutti i componenti visibili in fig.3 (sensa textool) .....€ 25,80

Costo del solo stampato LX.1202 .....€ 17,04
Costo di uno zoccolo textool a 20 piedini € 19,63
Costo di uno zoccolo textool a 28 piedini € 28,41

Il solo stadio di alimentazione siglato LX.1203 completo di circuito stampato, due integrati stabilizzatori, alette di raffreddamento, due ponti raddrizzatori, cordone di alimentazione e relativo trasformatore .....€ 25,80

Costo del solo stampato LX.1203 .....€ 4,34

Vi consigliamo di racchiudere lo stadio di alimentazione dentro un mobile plastico e a tale scopo vi proponiamo il modello MTK06.22.

Costo del mobile MTK06.22 .....€ 6,97

Ai prezzi riportati andranno aggiunte le sole spese di spedizione a domicilio.

Inserendo queste due schede nel **bus** siglato **LX.1202**, riportato su questo numero, e collocando nel suo zoccolo un **microprocessore ST6**, che voi stessi potrete programmare **copiando** uno dei programmi di **esempio** presenti nel dischetto che vi forniremo, potrete ottenere **orologi - contasecondi - timer - cronometri - contaimpulsi** ecc.
Poiché i nostri programmi di **esempio** si possono facilmente modificare, riuscirete in breve tempo a capire come scriverne altri per ottenere funzioni che noi attualmente non abbiamo previsto.
Queste schede risulteranno quindi utilissime per **testare** tutti i programmi che scriverete, perché vedrete dal **vivo** se appaiono i **numeri** desiderati e se i **relè** si eccitano o si diseccitano nei tempi previsti.
A queste schede ne seguiranno via via altre che utilizzeranno i display **LCD** ed i diodi **TRIAC**.
Entrerete così in possesso di un valido banco di **test** per tutti i tipi di programmi per **microprocessori ST6**.

Sul piedino **22** di **data** vanno inviati dei **bit seriali**, che l'integrato **M.5450** convertirà in **bit paralleli** necessari per accendere i **segmenti** dei quattro display.
La sequenza dei **bit** necessari per accendere i **segmenti** dei display deve essere preceduta da un **bit di start** (vedi fig.1).
In pratica all'integrato giunge una sequenza di **37 bit** come qui sotto riportato:

1 bit di **Start**
32 bit per accendere i **display**
2 bit per accendere i **diodi led**
2 bit di fine **caricamento**

I segmenti dei quattro **display** e dei due **diodi led** si accendono soltanto quando all'integrato sono giunti tutti i **37 bit**, vale a dire tutta la sequenza sopra riportata compresi gli ultimi **2 bit** di fine **caricamento**.

SCHEDA TEST per ST6

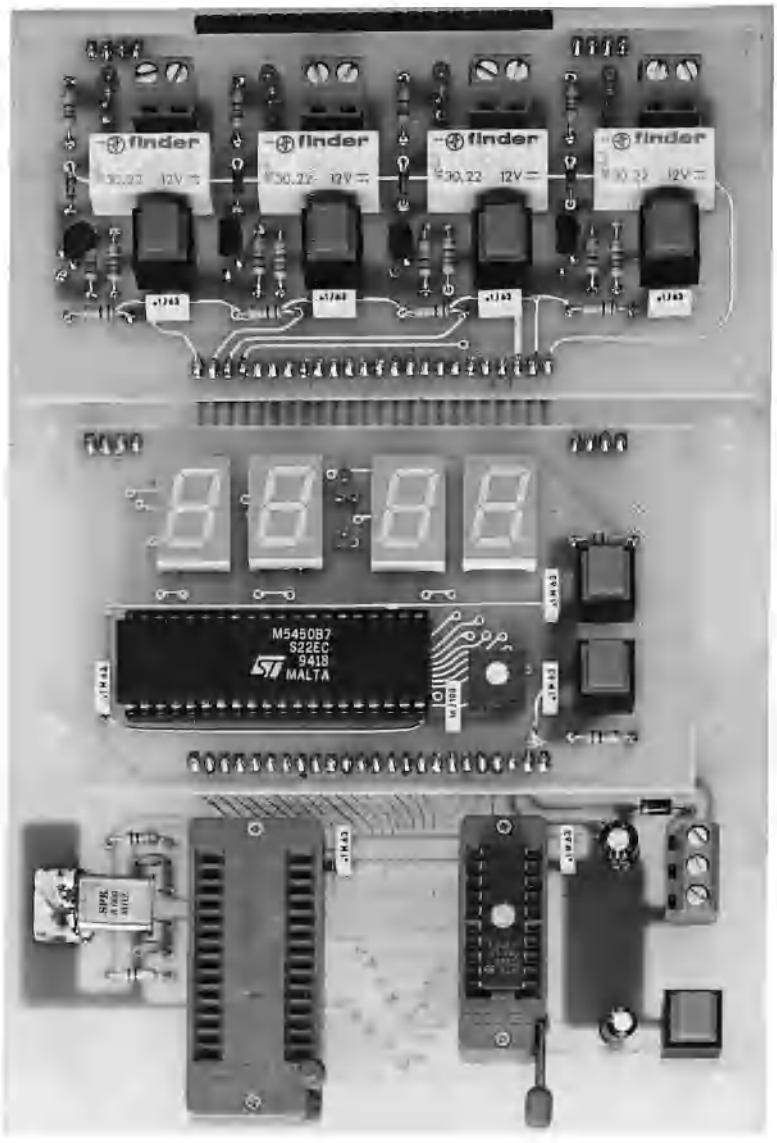
SCHEDA DISPLAY

Lo schema elettrico di questa scheda, riportato in fig.6, è molto semplice, perché utilizza il solo integrato **M.5450** (vedi **IC1**) necessario per pilotare **4 display**.
Nel caso realizzaste un **orologio**, i due **pulsanti** presenti nel circuito vi potranno servire per mettere a punto le **ore** ed i **minuti**, mentre i due **diodi led** potrebbero visualizzare i **secondi** oppure potreste impiegarli per altre funzioni da assegnare tramite software.
Il trimmer siglato **R1** serve soltanto per variare la **luminosità** dei display.
Poiché desideriamo che il lettore sappia come si riesca a far accendere un qualsiasi numero sui **4 display** collegando due soli fili all'integrato **M.5450** (vedi piedini **22-21**), dobbiamo a questo punto spiegare come vanno gestiti questi piedini.
Sul piedino **21** di **clock** va applicata una **frequenza** ad onda quadra che faremo generare dall'ST62 inserendo nel programma le due istruzioni **Set-Res** (vedi righe 119-120 nel programma **DISPLAY.ASM** presente nel dischetto).

Se guardate la fig.1, in cui sono riportati tutti i **7 segmenti** di un display contrassegnati da una **lettera**, potrete ricavare dalla **Tabella N.1** i **bit** che devono giungere al **piedino 22** dell'integrato per accendere i vari **segmenti**.
Il **bit**, come già sapete, è una cifra **binaria** che può assumere un **livello logico 0** oppure un **livello logico 1**.
Per accendere i **segmenti** interessati, i **bit** che entrano sul **piedino 22** devono avere un **livello logico 1**.
E' poi l'integrato che provvede a commutare le sue uscite a **livello logico 0** in modo che i **segmenti** ed i **diodi led** risultino alimentati.

Quindi se volessimo accendere il numero **7** sul display **N.4**, sull'ingresso di **IC1** dovremmo far giungere la sequenza di bit visibile in fig.2.
Se invece volessimo accendere il numero **7** sul display **N.1**, dovremmo far giungere sull'ingresso di **IC1** la sequenza di bit visibile in fig.3.
Nel dischetto che vi forniremo abbiamo inserito un programma chiamato **DISPLAY** per mostrarvi come si possa accendere qualsiasi **numero** in uno dei **quattro display**.

Potrete utilizzare queste due schede come orologio, contasecondi, timer, cronometro, contaimpulsi, e se questo non bastasse potrete eccitare, nei tempi da voi desiderati, dei relè per pilotare una cicalina o per alimentare una qualsiasi apparecchiatura elettrica.



SCHEDA RELÈ

Lo schema elettrico di questa scheda, riportato in fig.9, utilizza solo **4 relè** pilotati da altrettanti **transistor**.
In teoria avremmo potuto inserire ben **20 relè** utilizzando così tutte le porte **A - B - C**, ma poiché questa scheda viene inserita nel **bus** assieme alla scheda dei **display**, che già utilizza le **porte B0 - B1 - B2 - B3**, non potevamo servirci della stessa **porta** per accendere un display ed eccitare un relè.
I vari relè si eccitano quando sulle **porte B4 - B5 - B6 - B7** è presente un **livello logico 1**, che, polarizzando le Basi dei transistor, li portano in conduzione.
A relè **eccitato** si accende il **diodo led** applicato

TABELLA N. 1

DISPLAY N. 4

Bit	piedino IC1	segmento display
9	10	A
10	9	B
11	8	C
12	7	D
13	6	E
14	5	F
15	4	G
16	3	punto

DISPLAY N. 3

Bit	piedino IC1	segmento display
1	18	A
2	17	B
3	16	C
4	15	D
5	14	E
6	13	F
7	12	G
8	11	punto

DISPLAY N. 2

Bit	piedino IC1	segmento display
25	33	A
26	32	B
27	31	C
28	30	D
29	29	E
30	28	F
31	27	G
32	26	punto

DISPLAY N. 1

Bit	piedino IC1	segmento display
17	2	A
18	40	B
19	39	C
20	38	D
21	37	E
22	36	F
23	35	G
24	34	punto

ai suoi capi, quindi se sulla **morsettiera d'uscita** applicheremo una lampadina o un motorino, alimentati con una qualsiasi tensione esterna sia in **continua** sia in **alternata**, la lampada si **accenderà** ed il **motorino** inizierà a **girare**.
I pulsanti **P1 - P2 - P3 - P4**, collegati alle **porte A0 - A1 - A2 - A3**, sono stati inseriti per diseccitare o eccitare **manualmente** uno dei quattro relè.

REALIZZAZIONE PRATICA DISPLAY

Sul circuito stampato siglato **LX.1204** monterete i pochi componenti visibili in fig.7.
Per iniziare consigliamo di inserire lo **zoccolo** per l'integrato **IC1**, poi, dal lato opposto dello stampato, inserite il connettore **maschio** ad 1 fila provvisto di **24 terminali** e gli altri due connettori maschi, sempre ad 1 fila, provvisti di **4 terminali**, che in seguito vi serviranno per innestare questa scheda sui **connettori femmina** della scheda **bus** siglata **LX.1202**.
Dopo aver stagnato questi componenti, potete inserire le due resistenze, i quattro condensatori, il trimmer **R1** ed i due pulsanti **P1 - P2**.
Proseguendo nel montaggio inserite i due **diodi led** rivolgendo il terminale **più lungo**, cioè l'**Anodo**, verso sinistra.
Se invertirete questo terminale, i diodi led non si accenderanno.
Per completare il montaggio dovrete saldare direttamente sul circuito i quattro **display**, controllando che il lato in cui è presente il **punto decimale** risulti rivolto verso il basso, cioè verso l'integrato **IC1**.
Dopo aver inserito tutti i componenti, dovete inserire nel suo zoccolo l'integrato **M.5450** rivolgendo la sua tacca di riferimento ad **U** verso sinistra, come risulta visibile in fig.7.

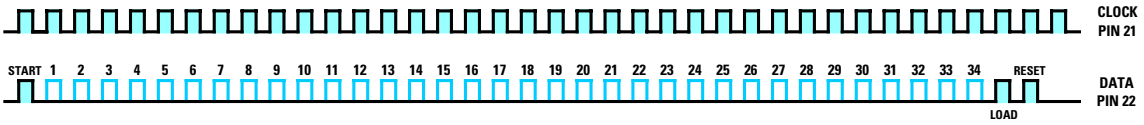
REALIZZAZIONE PRATICA RELÈ

Sul circuito stampato siglato **LX.1205** monterete tutti i componenti visibili in fig.10.
Per il montaggio vi consigliamo di iniziare inseren-

DIODO LED

Bit	piedino IC1	diodi led
34	24	DL2
33	25	DL1

Con queste tabelle potrete sapere quali bit devono giungere sul piedino 22 dell'integrato M.5450 (vedi fig.4) per accendere i sette segmenti dei quattro display ed i due diodi led.



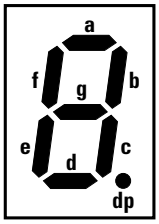


Fig.1 Sul piedino 22 dell'integrato M.5450 giunge un primo bit di Start. A questo seguono 32 bit per accendere i segmenti dei display, 2 bit per accendere i led e 2 bit di fine caricamento. I sette segmenti del display sono identificati da una lettera (vedi A-B-C-D-E-F-G) quindi per visualizzare il numero 7 dovrete alimentare i segmenti A-B-C e per visualizzare il numero 3 dovrete alimentare i segmenti A-B-G-C-D.

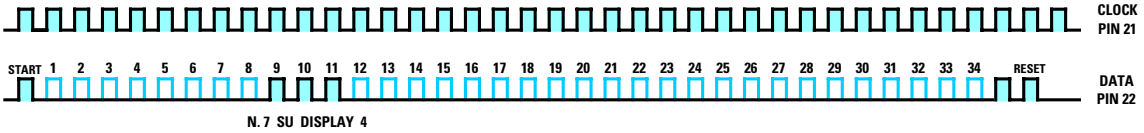


Fig.2 Per accendere il numero 7 sul display 4 dovrete far giungere sull'integrato M.5450 questa sequenza di bit seriali. Con i bit 9-10-11 verranno alimentati i soli segmenti A-B-C del display n.4 (vedi Tabella posta sulla sinistra).

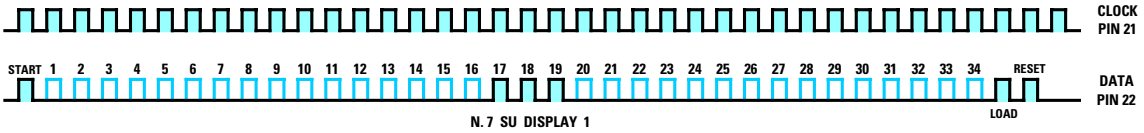


Fig.3 Per accendere il numero 7 sul display 1 dovrete far giungere sull'integrato M.5450 questa sequenza di bit seriali. Con i bit 17-18-19 verranno alimentati i soli segmenti A-B-C del display n.1 (vedi Tabella posta sulla sinistra).

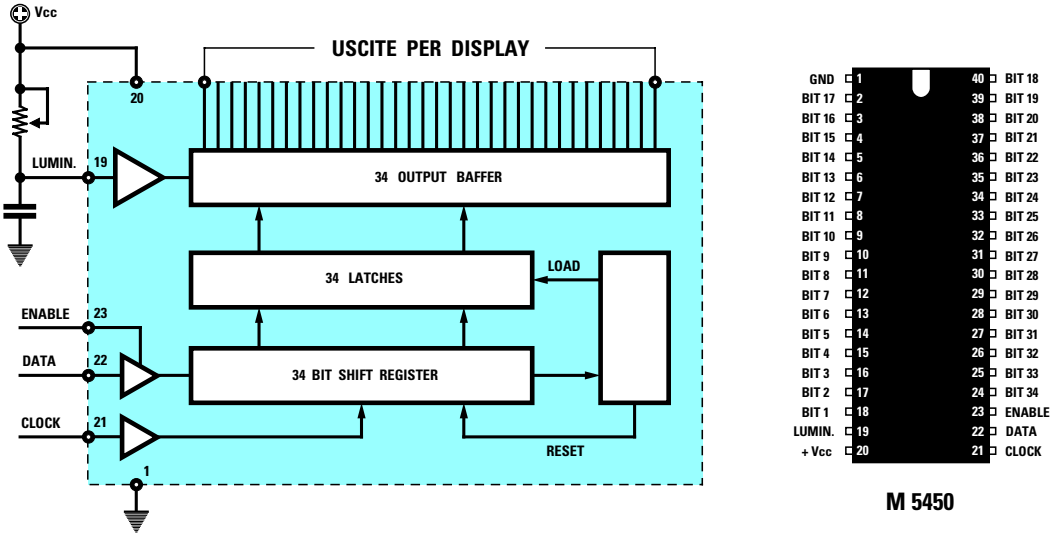


Fig.4 Schema a blocchi dell'integrato M.5450 e connessioni dei piedini sullo zoccolo viste da sopra. Entrando con un segnale seriale nel piedino 22 di questo integrato voi potrete accendere i segmenti dei 4 display (vedi fig.6).

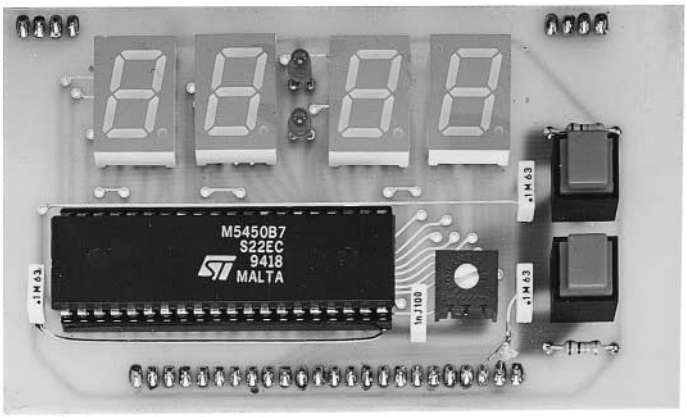


Fig.5 Foto della scheda display notevolmente rimpicciolita per motivi di spazio.

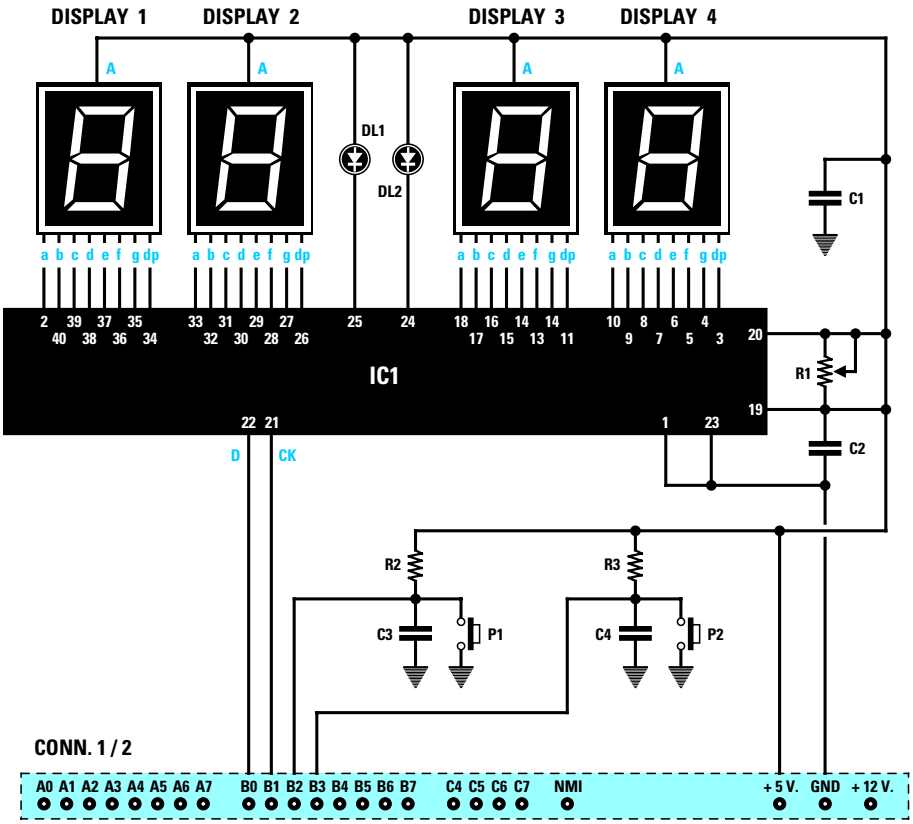


Fig.6 Schema elettrico della scheda display. I segmenti dei display si accendono quando il piedino d'uscita dell'integrato si porta a livello logico 0.

ELENCO COMPONENTI LX.1204

R1 = 50.000 ohm trimmer
R2 = 10.000 ohm 1/4 watt
R3 = 10.000 ohm 1/4 watt
C1 = 100.000 pF poliestere
C2 = 1.000 pF poliestere
C3 = 100.000 pF poliestere
C4 = 100.000 pF poliestere
DL1 = diodo led
DL2 = diodo led
DISPLAY1-4 = display tipo BS.A501
IC1 = M.5450
P1-P2 = pulsanti
CONN.1/2 = connettore 24 poli

do, dal lato opposto a quello dei componenti, il connettore **maschio** ad 1 fila provvisto di **24 terminali** e gli altri due connettori maschi, sempre ad 1 fila, provvisti di **4 terminali**, che in seguito vi serviranno per innestare questa scheda sui **connettori femmina** della scheda **bus** siglata **LX.1202**.
Ora voltate lo stampato e sulla parte inferiore del circuito inserite tutte le **resistenze**, poi i **diodi al silicio** non dimenticando di rivolgere il lato contornato da una **fascia nera** verso le morsettiere di uscita.
Proseguendo nel montaggio inserite tutti i **condensatori**, poi tutti i **transistor** rivolgendo la parte **piatta** del loro corpo verso sinistra come visibile nello schema pratico di fig.10.
Completata questa operazione, potete inserire i quattro **pulsanti**, i quattro **relè** e le quattro **morsettiere** a **2 poli**.
Per ultimi montate i **diodi led** non dimenticando di rivolgere il terminale **più lungo** dell'**Anodo** verso i relè.
Le quattro morsettiere presenti nello stampato sono collegate ai **contatti** dei relè, perciò quando il relè si **ecciterà** il contatto si chiuderà e pertanto lo potrete utilizzare come **interruttore** per accendere **lampadine**, alimentare **motorini** o **trasformatori** oppure dei servorelè a **220 volt**.

INSERIMENTO SCHEDE nel BUS

Potete inserire queste schede nel **bus** siglato **LX.1202** indifferentemente su uno dei due connettori **femmina** presenti sullo stampato.

L'ultimo connettore **femmina**, posto sulla parte superiore dello stampato **LX.1202**, è stato previsto nell'eventualità che vogliate collegare un vostro personale circuito stampato oppure per prolungare il **bus**.

I PROGRAMMI

Il nuovo dischetto che vi forniremo è in pratica lo stesso che abbiamo distribuito in precedenza (vedi rivista N.172/173), con la differenza che oltre ai programmi:

CONTA.ASM
LED.ASM
LOTTO.ASM
STANDARD.ASM

abbiamo aggiunto questi nuovi files:

RELE.ASM
DISPLAY.ASM
OROLOGIO.ASM
CRONOMET.ASM
TEMPOR.ASM
TIMER.ASM

Ogni riga di programma è stata completata da un **commento** che spiega la funzione dell'istruzione e quindi permette di sapere come modificare il programma per fargli compiere una funzione diversa da quella per cui era stato scritto.

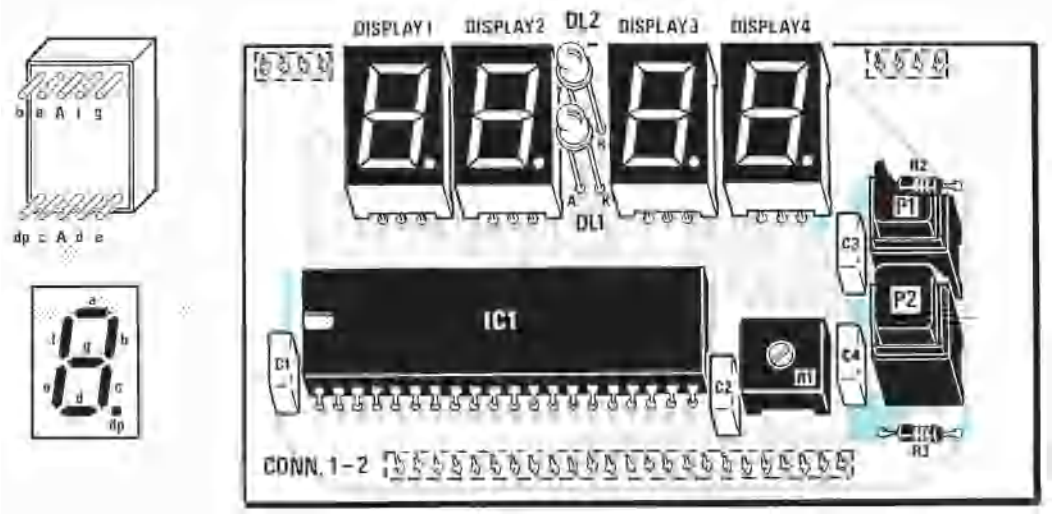


Fig.7 Schema pratico di montaggio della scheda display siglata LX.1204. Quando inserite i display rivolgete il loro "punto decimale" verso l'integrato IC1.

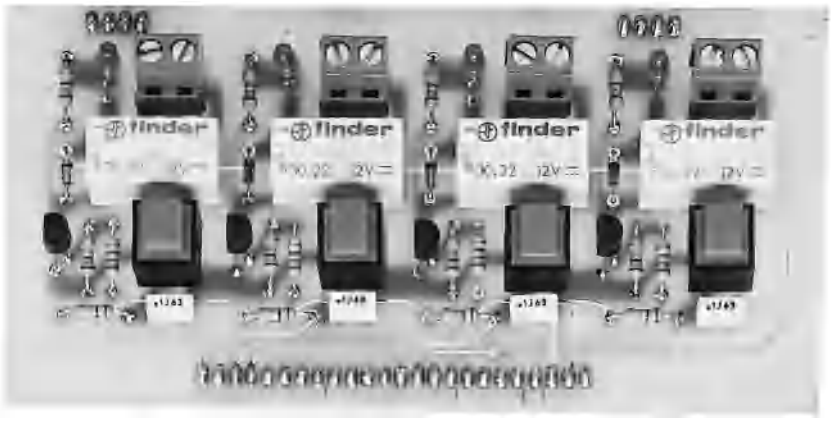


Fig.8 Foto ridotta della scheda relè.

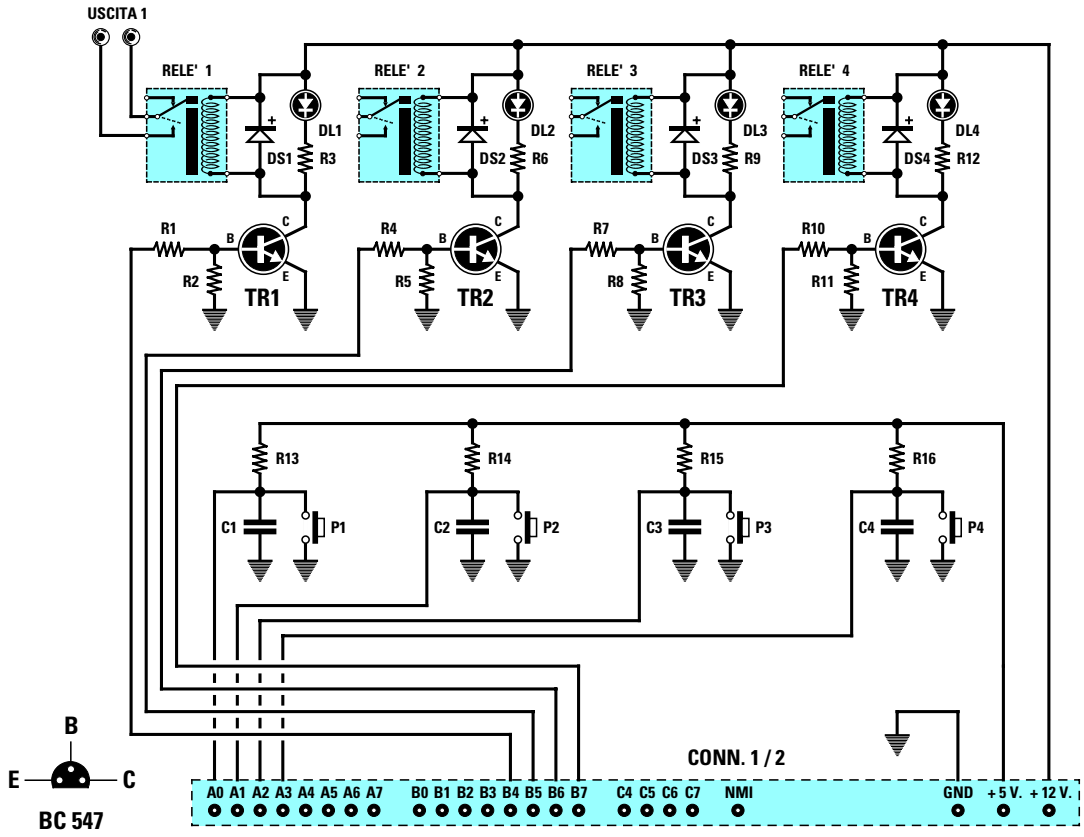


Fig.9 Schema elettrico della scheda relè siglata LX.1205 e connessioni del transistor BC.547 viste da sotto. Potrete utilizzare i relè per alimentare motorini o lampade.

ELENCO COMPONENTI LX.1205

R1 = 2.200 ohm 1/4 watt
R2 = 10.000 ohm 1/4 watt
R3 = 680 ohm 1/4 watt
R4 = 2.200 ohm 1/4 watt
R5 = 10.000 ohm 1/4 watt
R6 = 680 ohm 1/4 watt
R7 = 2.200 ohm 1/4 watt
R8 = 10.000 ohm 1/4 watt
R9 = 680 ohm 1/4 watt
R10 = 2.200 ohm 1/4 watt
R11 = 10.000 ohm 1/4 watt
R12 = 680 ohm 1/4 watt
R13 = 10.000 ohm 1/4 watt
R14 = 10.000 ohm 1/4 watt
R15 = 10.000 ohm 1/4 watt
R16 = 10.000 ohm 1/4 watt
C1-C4 = 100.000 pF poliestere
DS1-DS4 = diodi 1N.4150
DL1-DL4 = diodi led
TR1-TR4 = NPN tipo BC.547
RELE'1-4 = relè 12 volt
P1-P4 = pulsanti
CONN.1/2 = connettore 24 poli

Tanto per fare un esempio, se nel programma **TIMER** volete variare i tempi di eccitazione dei **relè**, troverete spiegato in quale riga va modificata l'istruzione e quale numero occorre inserire.
Se volete che il programma **TIMER** ecciti un **relè** per far **suonare** un campanello o per accendere una **caldaia** ad una precisa ora, vi verrà spiegato quale riga modificare per migliorare in base alle vostre personali esigenze la funzionalità del programma.

TRASFERIMENTO file nell'HARD-DISK

Una volta inserito il dischetto nel drive floppy, per trasferire tutti i suoi files nell'Hard-Disk dovete scrivere:

C:\>A: poi Enter
A:\>INSTALLA poi Enter

Non usate mai l'istruzione **Copy** del **Dos** o altri comandi analoghi del **PCshell - PCtools - Norton** o il **File Manager** di **Windows**, perché il programma **INSTALLA** presente nel dischetto provvede a **scompattare** automaticamente i files inseriti.
Dopo aver pigiato Enter apparirà la scritta:

Directory C:\ST6

Se il computer vi comunica che questa directory **esiste già**, non preoccupatevi e premete due volte il tasto **S**.
A questo punto inizia la **scompattazione** dei files (vedi fig.11) ed al termine dell'operazione vedrete apparire sul monitor la scritta:

Buon divertimento

Premendo un tasto qualsiasi, sul monitor apparirà la scritta:

C:\ST6>

Nel dischetto che vi abbiamo preparato, abbiamo incluso un semplice **Editor** che vi sarà molto utile per **visualizzare** tutte le righe di ogni programma, e, se lo desiderate, per **modificarle**.
Potrete usare lo stesso **Editor** per **scrivere** nuovi programmi, ed anche per **assemblarli** prima di **trasferirli** nella memoria del microprocessore **ST6** tramite il nostro programmatore siglato **LX.1170** presentato sulla rivista **N.172/173**.

Tutte le istruzioni per utilizzare l'**Editor** sono state già descritte nella rivista N.172/173, in particolare nell'articolo sul **Circuito Test** a pag.56, quindi vi consigliamo di rileggere attentamente questo articolo.

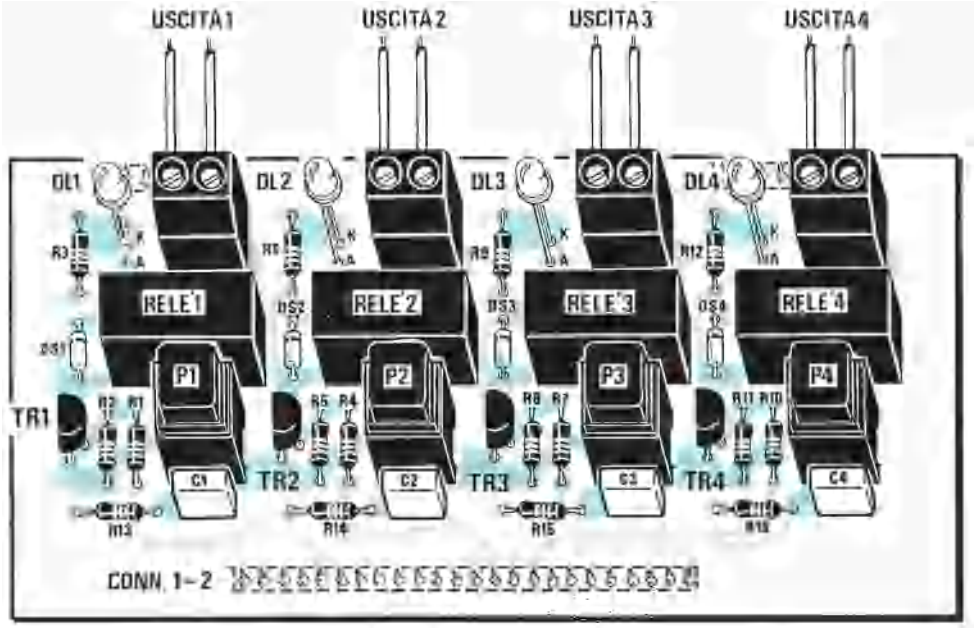


Fig.10 Schema pratico di montaggio della scheda LX.1205. Potrete modificare il programma di base che troverete nel dischetto in modo da adattarlo alle vostre esigenze.

Nelle pagine seguenti ci limiteremo infatti a descrivervi le istruzioni più importanti.

DUPLICARE un FILE con NOME diverso

Anziché **modificare** i files che trovate nel dischetto, vi consigliamo di ricopiarli nell'**Hard-Disk** con un nome **diverso**, e di apportare su questo **nuovo** file le modifiche che riterrete opportune.
In questo modo avrete sempre a disposizione, in caso di bisogno, il programma originale.
Ammesso che vogliate **duplicare** il file **DISPLAY**, dovrete innanzitutto scegliere un **nome** che non abbia più di **8 caratteri**, ad esempio **DPLPROVA**.
Quando siete nel **menu** principale del nostro **Editor**:

premete **ALT+F**
poi pigiate **D**

apparirà la scritta **C:\ST6>**
A questo punto potete scrivere l'istruzione:

C:\ST6>Copy DISPLAY.ASM DPLPROVA.ASM

poi premete Enter.
Dopo pochi secondi apparirà la scritta:

1 file copiato

Per ritornare al **menu** principale dovrete scrivere:

C:\ST6>EXIT poi Enter

Ora con il tasto funzione **F3** potrete richiamare il file **DPLPROVA** sul quale potrete apportare tutte le modifiche che riterrete opportune.

PER MODIFICARE un PROGRAMMA

Premendo il tasto funzione **F3**, quando siete nel **menu** principale, appare sul monitor una finestra con l'intero elenco dei **programmi**.
Premendo prima il tasto **TAB** e poi i tasti **freccia** potrete portare il cursore sui nomi dei programmi.
Quando il **cursore** si trova sul programma che volete modificare, premete **Enter** ed apparirà il **listato** completo del programma selezionato.

Per **cancellare una sola parola**
pigiate i tasti **CTRL+T**.

Per **cancellare un'intera riga**
pigiate i tasti **CTRL+Y**.

Dopo aver **corretto**, **modificato**, **cancellato** una **riga** o una **parola** dovrete **ricordarvi** di premere il tasto funzione **F2** per **memorizzare** tutte le modifiche che avete effettuato.

Fig.11 Tutti i files presenti nel dischetto verranno automaticamente scompattati quando li memorizzerete nell'Hard-Disk.

ASSEMBLAGGIO del PROGRAMMA

Un programma **corretto**, **modificato** o **riscritto**, si può trasferire in un micro **ST6** solo dopo averlo **assemblato**.
Dopo aver salvato un programma premendo il tasto funzione **F2**, per **assemblarlo** dovrete procedere come segue:

Premete **ALT+T**
quindi premete **A**

Se nel programma che avete modificato o riscritto non è presente nessun **errore**, dopo pochi secondi comparirà sul monitor la scritta:

*****SUCCESS*****

Se avete commesso qualche **errore**, apparirà in basso a sinistra sul monitor un **numero**, che corrisponde alla riga in cui c'è un **errore**.
Dovrete allora riaprire il file ed andare sulla **riga** che vi è stata segnalata per scoprire quale errore avete commesso.
Terminata la correzione, dovrete nuovamente **salvare** il programma premendo il tasto **F2**, dopodiché potrete ripetere i comandi già descritti per **assemblare** il vostro programma.

PER TRASFERIRE il programma sull'ST6

Dopo aver **assemblato** il programma ed aver ottenuto la scritta *****success*****, potrete trasferirlo nella memoria del microprocessore **ST6**.

Quando siete nel **menu** principale dovrete:

premere **ALT+P**

e dopo pochi secondi comparirà l'intestazione del software delle **SGS** in lingua inglese.
A questo punto, prendete la rivista **N.172/173** (se non l'avete potrete sempre richiederla) poi rileggete quanto riportato nelle **pagg.39-41**, che non riscriviamo perché sarebbe un'inutile ripetizione di quanto abbiamo già spiegato.

TRASFERIRE un FILE sul DISCHETTO

Nel caso voleste **trasferire** un programma già **assemblato** dall'Hard-Disk in un dischetto, ad esempio per darlo ad un amico, dovrete, sempre partendo dal **menu** principale:

premere **ALT+F**
poi premere **D**

Apparirà la scritta **C:\ST6>**
A questo punto inserite il **dischetto** nel **drive**, ed ammesso che abbiate chiamato il programma che volete trasferire **DPLPROVA**, dovrete scrivere questa istruzione:

C:\ST6>Copy DPLPROVA.ASM A:\DPLPROVA.ASM

poi Enter
Quando apparirà la scritta **1 file copiato** dovrete scrivere:

C:\ST6>EXIT poi Enter

TRASFERIRE dal FLOPPY all'HARD-DISK

Per **trasferire** un programma dal dischetto all'Hard-Disk dovrete, quando vi trovate nel **menu** principale**:**

premere **ALT+F**
poi premere **D**

Apparirà la scritta **C:\ST6>**
A questo punto inserite il **dischetto** nel **drive**, ed ammesso che il programma da trasferire si chiami **DPLPROVA**, dovrete scrivere questa istruzione:

C:\ST6>Copy A:\DPLPROVA.ASM DPLPROVA.ASM
poi Enter

Quando comparirà la scritta **1 file copiato** dovrete scrivere:

C:\ST6>EXIT poi Enter

NOTA IMPORTANTE

All'inizio di tutti i programmi che vi forniamo troverete questa istruzione:

.ORG 880h

che serve per i soli microprocessori **ST6** con **2K** di memoria, cioè i:

ST62/E10 - ST62/E15 - ST62/T10 - ST62/T15

Se utilizzerete dei microprocessori **ST6** con **4K** di memoria, cioè i:

ST62/E20 - ST62/E25 - ST62/T20 - ST62/T25

dovrete sostituire il numero **880h** con il numero **080h**, quindi dovrete scrivere:

.**ORG 080h**

Pertanto se usate un **ST6** da **2K** di memoria, dovrete necessariamente scrivere all'inizio del programma **880h**, perché se scriverete **080h**, non risultando presenti in questo **ST6** queste celle di memoria, non riuscirete a trasferire nessun programma, ed il programmatore lo segnalerà.
Se usate un **ST6** da **4K** di memoria dovrete scrivere all'inizio del programma **080h** per poter utilizzare tutta la sua memoria.
Facciamo presente che se avete un programma da **2K**, che ovviamente inizierà con l'indirizzo di memoria **880h**, lo potrete tranquillamente trasferire con lo stesso indirizzo anche in un **ST6** da **4K**.
In questo caso partendo dall'indirizzo **880h** rimarranno inutilizzate tutte le celle di memoria da **080h** a **87Fh**.

COSTO DI REALIZZAZIONE

Tutti i componenti per realizzare la scheda Display siglata LX.1204 visibile nelle figg.6-7 completa di circuito stampato, integrato M.5450 e 4 display .....€ 18,60

Tutti i componenti per realizzare la scheda Relè siglata LX.1205 visibile nelle figg.9-10 completa di circuito stampato e 4 relè .....€ 19,10

Costo del solo stampato LX.1204 .....€ 4,34
Costo del solo stampato LX.1205 .....€ 5,06

Ai prezzi riportati andranno aggiunte le sole spese di spedizione a domicilio.

Quando un lettore ci scrive che, montato un nostro progetto non riesce a farlo funzionare, presumiamo che abbia commesso un **errore**, poichè prima di pubblicare nella rivista un qualsiasi circuito, è nostra consuetudine farne montare una **decina** di esemplari e se constatiamo che uno di questi risulta **critico** o presenta qualche **anomalia** lo riportiamo in laboratorio per ricercarne le cause e per eliminarle.
Questo modo di procedere lo adottiamo per ridurre al **minimo** le **riparazioni** e per assicurare al lettore il sicuro ed immediato funzionamento di ogni nostro progetto.
A volte si verificano anche delle **strane anomalie** che fanno **arrabbiare** i lettori ed **impazzire** i tecnici della consulenza.

NOTA per il programmatore

Se voi foste un tecnico che da anni usa questo **programmatore per ST6** senza mai riscontrare nessun inconveniente, che conosce tanti amici (esattamente **4.738**) che lo hanno realizzato con successo, usandolo con diversi tipi di computer, e poi trovaste solo **16** lettori che si lamentano perchè il loro montaggio non funziona, cosa rispondereste loro ?
Senz'altro che hanno commesso un errore nel montaggio, ed è questa anche la nostra risposta.
Ciò che ci ha stupito è constatare che a tutti e **16** questi lettori sul monitor appariva lo stesso messaggio, cioè:

Target chip not present or defective!

Subito abbiamo pensato che avessero acquistato dei **microprocessori ST6 difettosi** già all'origine.
Per risolvere tale problema ci siamo fatti inviare questi **ST6** insieme al **programmatore** per controllare entrambi in laboratorio.
Appena arrivati, li abbiamo provati su 8 diversi computer e tutti gli **ST6** che ci sono stati inviati si sono regolarmente programmati, senza **errori**.
Rispediti i **programmatori** a questi lettori, tutti e **16** ci hanno risposto che appariva nuovamente il medesimo **errore**.
Se non ci chiamassimo **Nuova Elettronica**, a questo punto avremmo abbandonato questi **16 lettori** con i loro ST6 difettosi, ma per **serietà** ed anche per **capire** per quale motivo apparisse un simile **errore**, abbiamo chiesto a **2 lettori** residenti a **Ravenna** e a **Ferrara** se fossero disposti a venire a Bologna portando il loro **computer**, perchè volevamo cercare di scoprire la causa di questa anomalia.
Infatti se su **4.738** kit che funzionano in modo perfetto solo **16** si rifiutano di farlo, il difetto può essere causato solo dal **computer** ed infatti grazie a questi **2 lettori** siamo riusciti ad individuarlo.
Abbiamo scoperto che nei loro computer il segnale che entrava nel piedino **4** del **CONN.1** (vedi nello schema elettrico riportato a pag.31 della rivista N.172/173 il segnale **D2** che, tramite la **R7**, giunge al **piedino 1** della porta **IC1/E**) era **strettissimo** oppure aveva un'ampiezza **ridotta**.

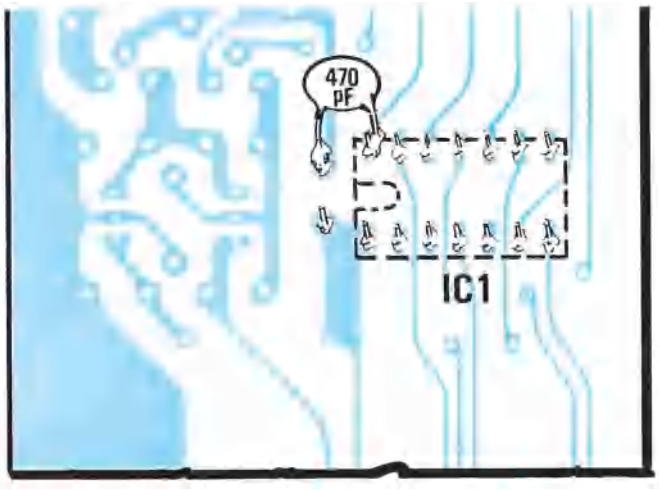
Scartata l'idea di manomettere il computer, abbiamo risolto il problema allargando l'impulso con un piccolo condensatore da **470 picoFarad** ceramico posto tra la resistenza **R7** e la **massa** come visibile nelle figg.1-2.
Immediatamente abbiamo comunicato ai **16 lettori** che non riuscivano a programmare gli **ST6**, di aggiungere sul loro **programmatore** questo condensatore da **470 picoFarad** e questi ci hanno risposto che la scritta:

Target chip not present or defective!

non appare più e con questa semplice modifica ora riescono a programmare qualsiasi **ST6**.

Importante = Se il vostro **programmatore** funziona correttamente non è necessario che inseriate questo condensatore, comunque se qualche volta vi capita di non riuscire a programmare un **ST6**, provate a collocare questo condensatore da **470 pF** tra la **R7** e la **massa** ed il difetto sparirà.
Come potete constatare, quando ci imbattiamo in qualche **anomalia**, facciamo tutto il possibile per eliminarla, ma se i due lettori di **Ravenna** e **Ferrara** non ci avessero portato il loro computer, forse questo caso sarebbe rientrato negli **insoluti**, perchè nessuno poteva supporre che il segnale che usciva dalla loro **presa parallela** fosse **fuori standard**.

Fig.1 Se constatate che il vostro programmatore LX.1170 non sempre riesce a programmare un ST6, potrete risolvere questo problema collegando tra il piedino 1 dell'integrato IC1-E e la massa dello stampato, un condensatore da 470 picoFarad.



LX.1170 per micro ST6

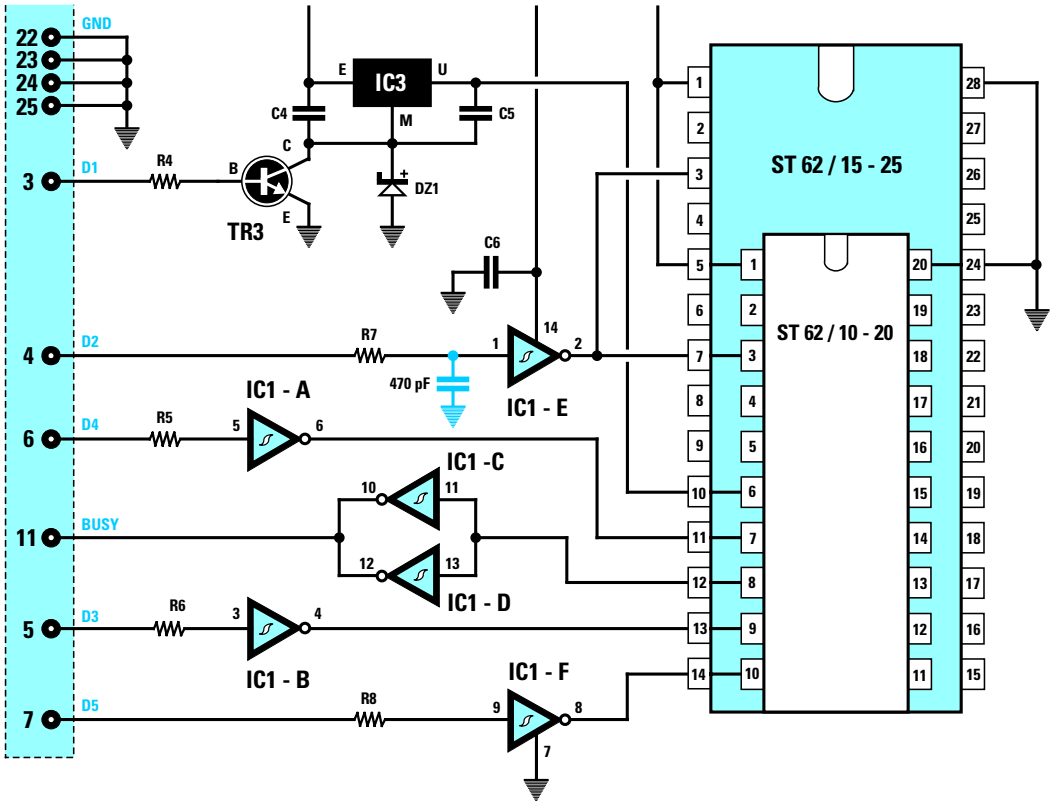


Fig.2 Questo condensatore, collegato dopo la resistenza R7, serve per allargare quegli impulsi che giungono troppo stretti sull'ingresso dell'inverter IC1-E.

A pag.124 della rivista N.175/176 abbiamo riportato il programma N.15 che consente di accendere 5 diodi led variando la tensione d'ingresso da 1 a 5 volt.
Molti lettori sulla base di questo esempio hanno utilizzato i micro **ST62E15** o **ST62E25**, poi hanno cercato di modificare il programma per far accendere 8 diodi led e quando sono andati ad assemblarlo, il computer ha segnalato questo errore:

5-bit displacement overlow

Con tale indicazione il microprocessore segnala che non può fare un **salto** all'**etichetta** richiesta perchè sono troppe le righe che lo separano da essa.
Nel nostro esempio avevamo usato soltanto **5 diodi led**, quindi utilizzando degli altri **diodi led** si sono dovute aggiungere delle altre righe di programma e poichè le istruzioni **JRC - JRNC - JRZ - JRNZ** possono fare solo dei **salti** limitati ad un certo numero di righe, se queste sono maggiori del richiesto appare il messaggio di **errore** menzionato.
In presenza di **salti** molto **lunghi** è necessario modificare l'istruzione presente con quella **inversa**, poi scrivere nella riga successiva l'istruzione **JP** che è in grado di fare un **salto** in qualsiasi punto del programma anche se molto distante.

Le istruzioni andranno quindi modificate:

da **JRC** a **JRNC**
da **JRNC** a **JRC**
da **JRZ** a **JRNZ**
da **JRNZ** a **JRZ**

PROGRAMMA per Esempio n.15

```
        LDI     pdir_a,00000000B    ; in queste tre righe abbiamo settato
        LDI     popt_a,10000000B    ; il piedino A7 come ingresso analogico
        LDI     port_a,10000000B    ;
        LDI     pdir_b,00011111B    ; in queste righe abbiamo settato
        LDI     popt_b,00011111B    ; i piedini da B0 a B4 come uscite
        LDI     port_b,00000000B    ;
ripeti  LDI     wdog,255            ; carichiamo il watchdog
        LDI     adcr,00110000B      ; provvedi a convertire da analogico a digitale
attendi JRR     6,adcr,attendi      ; attendere che avvenga la conversione A/D
        LD      a,addr              ; carica nell'accumulatore A, il numero digitale
        CPI     a,255               ; compara il valore di A con il numero 255
        JRNC    LED5                ; se A è uguale a 255 salta all'etichetta LED5
        CPI     a,204               ; compara il valore di A con il numero 204
        JRNC    LED4                ; se A è maggiore di 204 salta all'etichetta LED4
        CPI     a,153               ; compara il valore di A con il numero 153
        JRNC    LED3                ; se A è maggiore di 153 salta all'etichetta LED3
        CPI     a,102               ; compara il valore di A con il numero 102
        JRNC    LED2                ; se A è maggiore di 102 salta all'etichetta LED2
        CPI     a,51                ; compara il valore di A con il numero 51
        JRNC    LED1                ; se A è maggiore di 51 salta all'etichetta LED1
        JP      LED0                ; se A è minore di 51 salta all'etichetta LED0
LED0    LDI     port_b,00000000B    ; non accendere nessun diodo led
        JP      ripeti              ; salta all'etichetta ripeti del watchdog
LED1    LDI     port_b,00000001B    ; accendi il led sul piedino B0
        JP      ripeti              ; salta all'etichetta ripeti del watchdog
LED2    LDI     port_b,00000011B    ; accendi i led sui piedini B0 - B1
        JP      ripeti              ; salta all'etichetta ripeti del watchdog
LED3    LDI     port_b,00000111B    ; accendi i led sui piedini B0 - B1 - B2
        JP      ripeti              ; salta all'etichetta ripeti del watchdog
LED4    LDI     port_b,00001111B    ; accendi i led sui piedini B0 - B1 - B2 - B3
        JP      ripeti              ; salta all'etichetta ripeti del watchdog
LED5    LDI     port_b,00011111B    ; accendi i led sui piedini B0 - B1 - B2 - B3 - B4
        JP      ripeti              ; salta all'etichetta ripeti del watchdog
```

PROGRAMMA SALTO Esempio n. 15 bis

```
        LDI   pdir_a,00000000B   ; in queste tre righe abbiamo settato
        LDI   popt_a,10000000B   ; il piedino A7 come ingresso analogico
        LDI   port_a,10000000B   ;
        LDI   pdir_b,11111111B   ; in queste righe abbiamo settato
        LDI   popt_b,11111111B   ; i piedini della porta B come uscite
        LDI   port_b,00000000B   ;
ripeti  LDI   wdog,255           ; carichiamo il watchdog
        LDI   adcr,00110000B     ; provvedi a convertire da analogico a digitale
attendi JRR   6,adcr,attendi     ; attendere che avvenga la conversione A/D
        LD    a,addr             ; carica nell'accumulatore A il numero digitale
        CPI   a,255              ; compara il valore di A con 255
        JRC   etich1             ; se A è minore di 255 salta a etich1
        JP    LED8               ; salta a LED8
etich1  CPI   a,224              ; compara il valore di A con 224
        JRC   etich2             ; se A è minore di 224 salta a etich2
        JP    LED7               ; salta a LED7
etich2  CPI   a,192              ; compare il valore di A con 192
        JRC   etich3             ; se A è minore di 192 salta a etich3
        JP    LED6               ; salta a LED6
etich3  CPI   a,160              ; compara il valore di A con 160
        JRC   etich4             ; se A è minore di 160 salta a etich4
        JP    LED5               ; salta a LED5
etich4  CPI   a,128              ; compara il valore di A con 128
        JRC   etich5             ; se A è minore di 128 salta a etich5
        JP    LED4               ; salta a LED4
etich5  CPI   a,96               ; compara il valore di A con 96
        JRC   etich6             ; se A è minore di 96 salta a etich6
        JP    LED3               ; salta a LED3
etich6  CPI   a,64               ; compara il valore di A con 64
        JRC   etich7             ; se A è minore di 64 salta a etich7
        JP    LED2               ; salta a LED2
etich7  CPI   a,32               ; compara il valore di A con 32
        JRC   etich8             ; se A è minore di 32 salta a etich8
        JP    LED1               ; salta a LED1
etich8  JP    LED0               ; salta a LED0
LED0    LDI   port_b,00000000B   ; non accendere nessun diodo led
        JP    ripeti             ; salta all'etichetta ripeti del watchdog
LED1    LDI   port_b,00000001B   ; accendi il led sul piedino B0
        JP    ripeti             ; salta all'etichetta ripeti del watchdog
LED2    LDI   port_b,00000011B   ; accende i led sui piedini B0 - B1
        JP    ripeti             ; salta all'etichetta ripeti del watchdog
LED3    LDI   port_b,00000111B   ; accende i led sui piedini B0 - B1 - B2
        JP    ripeti             ; salta all'etichetta ripeti del watchdog
LED4    LDI   port_b,00001111B   ; accende i led sui piedini B0 - B1 - B2 - B3
        JP    ripeti             ; salta all'etichetta ripeti del watchdog
LED5    LDI   port_b,00011111B   ; accende i led sui piedini B0 - B1 - B2 - B3 - B4
        JP    ripeti             ; salta all'etichetta ripeti del watchdog
LED6    LDI   port-b,00111111B   ; accende i led sui piedini da B0 a B5
        JP    ripeti             ; salta all'etichetta ripeti del watchdog
LED7    LDI   port-b,01111111B   ; accende i led sui piedini da B0 a B6
        JP    ripeti             ; salta all'etichetta ripeti del watchdog
LED8    LDI   port-b,11111111B   ; accende i led sui piedini da B0 a B7
        JP    ripeti             ; salta all'etichetta ripeti del watchdog
```

Nel programma presentato a **pag.124** come "**Esempio N.15**" abbiamo scritto nella **12° riga** questa istruzione:

JRNC LED5 ; se A è uguale a 255 salta all'etichetta LED5

Nel caso in cui si desiderino aggiungere degli altri led, sarà necessario **modificare** il programma come segue:

JRC etich1; se A è minore di 255 salta a etich1
JP LED8; salta a LED8
Etich1; etichetta 1 che proseguirà con il programma

Se ancora tutto questo non vi risulta chiaro, confrontate il primo programma **Esempio N.15** con il secondo programma modificato, che abbiamo chiamato **Esempio N.15 BIS**.

CONVERTITORE A/D

Nella rivista N.175/176 a pag.123 abbiamo scritto che, risultando presente all'interno dell'**ST6** un **solo A/D converter**, potevamo utilizzare come ingresso per **segnali analogici** un **solo** piedino.
In pratica è possibile utilizzare anche **più piedini** come ingressi **analogici**, sempre che si scriva un programma che vada a leggere in **multiplexer** le tensioni presenti su tutti i piedini d'ingresso che abbiamo prescelto per questa funzione.
Per farvi comprendere come un **solo A/D converter** possa leggere le tensioni poste su **diversi** piedini, vi proponiamo qui di seguito un esempio.
Ammesso di possedere un **solo voltmetro** e di voler leggere con questo valori di **tensione** presenti su punti diversi, potremo farlo se sull'ingresso del **voltmetro** applicheremo il **cursore** di un **commutatore rotativo** e sui terminali di commutazione le diverse tensioni che vorremo leggere.
Ruotando il commutatore sulle diverse posizioni, potremo leggere **più tensioni** pur disponendo di un **solo voltmetro**.
Vogliamo comunque ricordarvi che come **ingressi analogici** potremo usare qualsiasi piedino ad eccezione dei soli piedini **A0-A1-A2-A3**.
Nel programma che qui riportiamo come **Programma A/D** vi facciamo vedere come bisognerà scrivere le istruzioni per poter leggere le tensioni presenti sui piedini d'ingresso di **B5-B6-B7**.

PROGRAMMA A/D

```
        LDI   pdir_b,00000000B   ; nelle prime cinque righe
        LDI   popt_b,00000000B   ; abbiamo settato il piedino B7
        LDI   port_b,00000000B   ; come ingresso analogico
        LDI   port_b,10000000B   ;
        LDI   popt_b,10000000B   ;
        LDI   adcr,00110000B     ; provvedi a convertire da analogico a digitale
AD1     JRR   6,adcr,AD1         ; attendere che avvenga la conversione A/D
        LD    a,ADDR             ; copia in A il valore dell'A/D
        LD    VOLT1,a            ; copia in VOLT1 il valore di a
        LDI   pdir_b,00000000B   ; nelle prime cinque righe
        LDI   popt_b,00000000B   ; abbiamo settato il piedino B6
        LDI   port_b,00000000B   ; come ingresso analogico
        LDI   port_b,01000000B   ;
        LDI   popt_b,01000000B   ;
        LDI   adcr, 00110000B    ; provvedi a convertire da analogico a digitale
AD2     JRR   6,adcr,AD2         ; attendere che avvenga la conversione A/D
        LD    a,ADDR             ; copia in A il valore dell'A/D
        LD    VOLT2,a            ; copia in VOLT2 il valore di a
        LDI   pdir_b,00000000B   ; nelle prime cinque righe
        LDI   popt_b,00000000B   ; abbiamo settato il piedino B5
        LDI   port_b,00000000B   ; come ingresso analogico
        LDI   port_b,00100000B   ;
        LDI   popt_b,00100000B   ;
        LDI   adcr,00110000B     ; provvedi a convertire da analogico a digitale
        JRR   6,adcr,AD3         ; attendere che avvenga la conversione A/D
        LD    a,ADDR             ; copia in A il valore dell'A/D
        LD    VOLT3,a            ; copia in VOLT3 il valore di a
```

Se avete già acquistato il kit per **testare** gli ST6 siglato **LX.1202** e le due schede, una con quattro **display** siglata **LX.1204** e l'altra con quattro **relè** siglata **LX.1205** pubblicate nella rivista **N.179**, avrete ricevuto anche un dischetto con codice **DF.1202/3 = DF.1170/3** contenente diversi programmi formato "**.ASM**" che, caricati nell'hard-disk, vi serviranno per gestire le due schede sperimentali apparse nella rivista **N.179** e quella che appare su questo numero con quattro **triac** siglata **LX.1206**.
Prossimamente vi forniremo altre due schede per display **LCD alfanumerici** ed altri nuovi programmi.
Anche se lo abbiamo già precisato negli articoli precedenti, vi ricordiamo che i programmi **.ASM** li potrete trasferire **singolarmente** nella **memoria** di un microprocessore **ST6** solo dopo averli **assemblati**, cioè convertiti in files formato **.HEX**.

I tre programmi **CONTA-LED-LOTTO** che sono **assemblati** in **.HEX** sono già pronti per essere caricati all'interno della memoria dell'ST6.
Tutti gli altri programmi che terminano con **.ASM** li potrete tranquillamente modificare, ampliare e, come già accennato, prima di passarli nella memoria di un **ST6** li dovrete **assemblare** per convertirli in files **.HEX**.

Le modifiche in questi programmi sono sempre necessarie per adattarli alle vostre esigenze. Ad esempio, noi abbiamo predisposto i programmi **TIMER.ASM** e **TEMPOR.ASM** per eccitare un **relè** o un **Triac** in un tempo di **3 minuti** sia contando all'indietro (TEMPOR.ASM) che in avanti (TIMER.ASM) e poichè questo tempo non vi servirà per nessuna delle vostre applicazioni, basterà leggere all'interno del programma i vari **commenti** per sapere quale riga dovrete **correggere** e quale **numero** inserire per modificarla.
Anche in tutti gli altri programmi troverete di lato ad ogni riga un **commento** che vi spiegherà se potete modificarla, sostituirla, o cancellarla.
Le modifiche non fatele mai sul nostro **file**, ma su un identico file che **duplicherete** attribuendogli un **nome** diverso, in modo da avere sempre a disposizione il **file originale** per poterlo consultare o confrontare per scoprire eventuali errori sui files modificati.

SCHEDA con 4 TRIAC

Nel dischetto **DF.1202/3** identico al **DF.1170/3** che da oggi forniamo, troverete questi **18 programmi**:

1^ CONTA.ASM
2^ LED.ASM
3^ LOTTO.ASM
4^ STANDARD.ASM

5^ CRONOMET.ASM
6^ DISPLAY.ASM
7^ LM093.ASM
8^ OROLOGIO.ASM
9^ RELE.ASM
10^ TEMPOR.ASM
11^ TIMER.ASM
12^ TRIAC.ASM
13^ CLOCK.ASM
14^ TIME90.ASM
15^ TEMP90.ASM

16^ CONTA.HEX
17^ LED.HEX
18^ LOTTO.HEX

Nota = I primi **4** programmi ve li avevamo già forniti con il **primo** disco floppy assieme al programmatore per **ST6** (vedi rivista **n.172-173**).

Tutte le istruzioni richieste per poter **duplicare** un **file** le troverete in questo articolo.
Non dovrete **mai modificare** i programmi presenti all'interno del dischetto che terminano con **BAT - EXE - COM - DEV - HEX**.
Sul disco che vi forniremo, oltre ai programmi **.ASM**, è presente anche un **EDITOR** che vi servirà per scrivere dei programmi e per **ASSEMBLARE** i files prima di caricarli sui microprocessori **ST6** tramite il programmatore **LX.1170** (leggere la rivista **N.172-173**).

Una volta comprese le funzioni dei vari **blocchi**, potrete **ampliarli**, **modificarli** oppure **trasferirli** su un vostro programma per poter gestire, secondo la vostra fantasia, queste ed altre **schede sperimentali**.

per microprocessori ST6

Sul precedente numero della rivista vi abbiamo presentato due schede per ST6, una per accendere dei normali display a 7 segmenti ed un'altra per eccitare dei relè. In questo numero vi presentiamo una scheda per eccitare quattro diodi Triac, spiegandovi anche come si possono modificare i programmi da noi forniti.

UN PROMEMORIA

Anche se nel numero **179** della rivista abbiamo spiegato che occorre necessariamente inserire queste schede sperimentali nel **bus** siglato **LX.1202**, molti ci chiedono se e su quale tipo di computer occorra collegarle e se per le prove convenga usare un **ST6 cancellabile** o **non cancellabile**.

- La scheda **bus** siglata **LX.1202** risulta progettata per ricevere tutte le schede **sperimentali** che vi abbiamo presentato e anche le future con display **LCD.** Questa scheda **non andrà** collegata a nessun computer, ma serve per inserirvi l'**ST6** che abbiamo programmato.

- Oltre alla scheda **bus** vi servirà anche il **Programmatore per micro ST6** siglato **LX.1170** pubblicato nella rivista **N.172/173**, che sarà **indispensabile** per poter trasferire il **programma** che sceglierete o che avrete scritto dal computer alla memoria del **microprocessore ST6**.
Dopo aver trasferito il programma nel microprocessore, dovrete togliere quest'ultimo dallo zoccolo Textool del programmatore **LX.1170** ed inserirlo nella scheda **bus** siglata **LX.1202** per poter gestire le schede sperimentali che applicherete su questo **bus**.
Vi ricordiamo che il solo programmatore **LX.1170** andrà collegato alla **porta parallela** del vostro computer, purchè questo sia un **IBM** o un **compatibile** serie **XT - AT - SX - DX** tipo **8088 - 286 - 386 - 486 - Pentium** con qualsiasi **frequenza di clock**, compresi anche i **portatili** con installato il sistema operativo **DOS** dal **3** al **6.2**.
Ripetiamo nuovamente che il **software** per **ST6** non è assolutamente compatibile per computer tipo Commodore, Apple, Amiga, ecc.

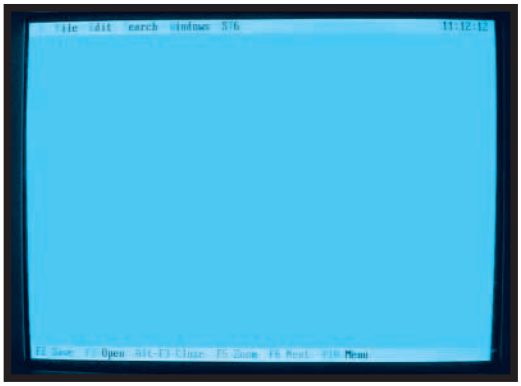


Fig.1 Per duplicare il programma STANDARD.ASM dovrete prima richiamare l'Editor, scrivendo C:\>ST6 poi Enter. Dopodichè scriverete C:\ST6>ST6 e a questo punto potrete premere il tasto Enter.

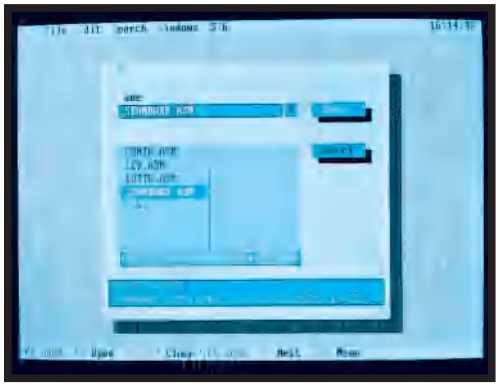


Fig.2 Premete i tasti ALT F poi F3 e, così facendo, vi appariranno tutti i files ASM. Portate il cursore sulla riga STANDARD.ASM, premete Enter e, in tal modo, vi apparirà la finestra di fig.3.

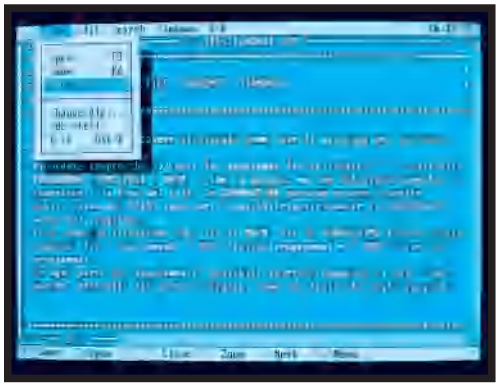


Fig.3 Portate il cursore sulla riga SAVE as ... poi premete Enter. Per duplicare questo file con un nome "diverso" ripremete i due tasti ALT F e, in tal modo, vi apparirà la finestra di fig.4.

- Per le prove **sperimentali** conviene scegliere un microprocessore tipo **ST62E20** con **4 K di Rom** (il micro **ST62E10** è stato messo fuori produzione dalla **SGS**), perchè, anche se risulta molto **costoso**, lo potrete **cancellare** e riutilizzare diverse centinaia di volte per **memorizzare** degli altri nuovi e diversi programmi.

- Per **cancellare** uno di questi microprocessori potrete usare la lampada ad **ultravioletti** siglata **LX.1183** presentata sulla rivista **N.174.**

- Se volete usare i più economici microprocessori tipo **ST62T10** o **ST62T20** potete farlo, ma poichè questi **non sono cancellabili**, se **sbaglierete** nello scrivere un programma, dovrete buttarli ed acquistarne degli **altri**.
Normalmente i microprocessori **non cancellabili** vengono utilizzati solo dopo aver **testato** più di una volta il micro **cancellabile** tipo **ST62E20**, per avere la certezza che nel **vostro programma** non vi siano degli **errori**.

TABELLA N.1 micro NON CANCELLABILI

Sigla Micro	memoria utile	Ram utile	zoccolo piedini	piedini utili per i segnali
ST62T.10	2 K	64 byte	20 pin	12
ST62T.15	2 K	64 byte	28 pin	20
ST62T.20	4 K	64 byte	20 pin	12
ST62T.25	4 K	64 byte	28 pin	20

TABELLA N.2 micro CANCELLABILI

Sigla Micro	memoria utile	Ram utile	zoccolo piedini	piedini utili per i segnali
ST62E.20	4 K	64 byte	20 pin	12
ST62E.25	4 K	64 byte	28 pin	20

- Quando vorrete scrivere dei **nuovi** programmi dovrete sempre caricare il file **STANDARD.ASM**, perchè questo è il file **sorgente** che definisce la locazione dei **5 registri** del micro e fa il **settaggio** delle periferiche, cioè una inizializzazione completa.

IL SORGENTE STANDARD.ASM

Ritenevamo di aver spiegato abbastanza bene a cosa serva il file **STANDARD.ASM**, ma leggendo i quesiti che ci sono pervenuti in proposito abbiamo capito di non essere stati sufficientemente esaurienti.

Per riparare, ve lo rispiegheremo proponendovi anche qualche esempio.
Quando si scrive un **programma**, occorre sempre iniziare con dei **dati ripetitivi** che non varino mai da un programma ad un altro, quindi per non **riscriverli** ogni volta con il rischio di commettere degli **errori**, ve li ritroverete già tutti **impostati** nel programma **STANDARD.ASM** con una **nota** relativa a cosa dovrete **modificare**.
Ad esempio, quando arriverete al paragrafo **SETTAGGIO INIZIALE**, subito dopo **INIZIO PROGRAMMA** troverete tra le prime righe l'istruzione:

```
.org 0880h
```

Se usate un micro da **2K** di memoria, cioè un **ST62E10 - ST62T10 - ST62E15 - ST62T15**, non dovrete **modificare** questo numero.

Se usate un micro da **4K**, cioè un **ST62E20 - ST62T20 - ST62E25 - ST62T25**, dovrete invece **modificare** questo numero come segue:

```
.org 080h
```

Ammettiamo ad esempio di voler **creare** un nuovo programma chiamato **ALIBABA.**
La prima operazione da effettuare sarà quella di **uscire** da qualsiasi programma in utilizzo, come ad esempio **Windows**, **Pcshell**, **Norton**, ecc., in modo da vedere in alto a sinistra del vostro schermo il solo **Prompt** dei comandi, cioè **C:\>**
A questo punto potrete richiamare il programma **ST6** scrivendolo indifferentemente sia in maiuscolo che in minuscolo:

C:\>CD ST6 poi premete Enter

e in questo modo vi apparirà:

C:\ST6>

poi scrivete **ST6** come sotto riportato:

C:\ST6>ST6 poi premete Enter

e subito vedrete apparire sul monitor del computer la finestra dell'**EDITOR** (vedi fig.1).

A questo punto premete contemporaneamente i tasti **ALT F**, poi **F3** e vi apparirà una seconda finestra con gli elenchi di tutti i **files .ASM** (vedi fig.2). Premete il tasto **Enter** e vedrete che il cursore andrà sul primo file **.ASM** colorandolo di **verde**.

Utilizzando i tasti **freccia** presenti sulla tastiera, portate il **cursore** sulla riga **STANDARD.ASM**, poi premete **Enter** e vedrete apparire sul monitor il **li-**

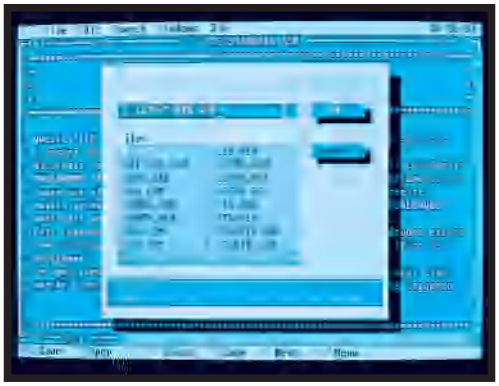


Fig.4 Per cambiare il nome del file STANDARD.ASM con ALIBABA dovrete scrivere C:\ST6\ALIBABA.ASM e poi premere ALT F3. Se desiderate cambiare il listato del file TIMER dovrete scrivere C:\ST6\TIMER.ASM.

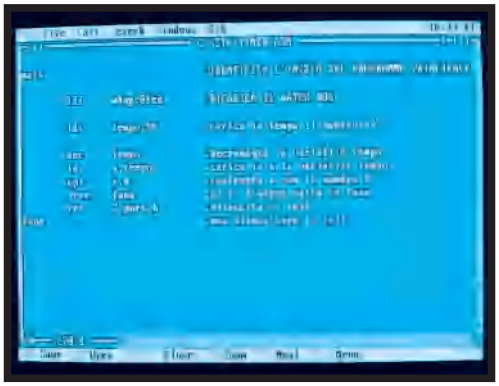


Fig.5 In tutti i listati troverete di lato un "commento condensato" che vi aiuterà a capire quale funzione esplicano le varie righe. Con un po' di esperienza riuscirete molto facilmente a modificarle.

Fig.6 Corretto o riscritto, dovrete "salvare" il nuovo programma premendo il tasto F2, dopodichè lo potrete ASSEMBLARE premendo i tasti ALT T poi A. Se non avete commesso "errori" apparirà SUCCESS.

stato di questo programma che potrete leggere dall'inizio fino alla fine.
A questo punto se premerete contemporaneamente i due tasti **ALT F** vedrete apparire una nuova maschera (vedi fig.3).
Con il tasto freccia giù andate alla riga dove è scritto:

SAVE as.... poi premete Enter

Così facendo vi apparirà una riga con scritto:

C:\ST6\STANDARD.ASM

Poichè intendiamo chiamare il **nuovo** programma **ALIBABA**, questa riga la dovrete riscrivere come segue:

C:\ST6\ALIBABA.ASM poi premete Enter

Nota = I nomi dei **files** non debbono mai avere più di **8 caratteri** escluso ovviamente **.ASM**
Corretta questa riga, premete i tasti **ALT F3** e vi apparirà nuovamente la maschera dell'**EDITOR**.
A questo punto dovrete premere i due tasti **ALT F**, poi **F3** e nella lista dei **files** troverete il nuovo file denominato **ALIBABA.ASM**.

Dopo aver premuto **Enter**, con i tasti delle **frecce** portate il **cursore** sul file **ALIBABA.ASM** e, in questo modo, vi apparirà il **listato duplicato** del file **STANDARD.ASM**, che potrete tranquillamente modificare perchè ora lavorerete sul file **ALIBABA.ASM.**
Se, per ipotesi, tutte le modifiche che apporterete sul file **ALIBABA.ASM** non lo faranno funzionare per qualche **errore** da voi commesso, lo potrete **cancellare** e nuovamente **ricopiare**, utilizzando il file originale **STANDARD.ASM** come vi abbiamo appena spiegato.
Nel programma che in precedenza si chiamava **STANDARD.ASM** e che ora avete chiamato **ALIBABA.ASM** dovrete ricordarvi che la riga:

```
.org  0880h
```

non va modificata se userete dei micro con **2K** di memoria, vale a dire se userete degli **ST62E10**, **ST62E15**, **ST62T10**, **ST62T15**.

Se invece userete dei micro con **4K**, vale dire degli **ST62E20**, **ST62E25**, **ST62T20**, **ST62T25**, questa riga va **modificata** come segue:

```
.org  080h
```

A questo punto dovrete modificare il settaggio di tutte le **porte A-B-C**, cioè dovrete impostarle come **ingressi** o **uscite** a seconda delle esigenze del vostro programma.
Nel caso non sappiate come si faccia a settare le porte, vi consigliamo di leggere l'articolo **"Imparare a programmare i microprocessori ST6"** pubblicato nella rivista n.**175**-**176**.

Terminate tutte le modifiche e scritto il **nuovo programma** che si chiama **ALIBABA.ASM**, lo dovrete **salvare** premendo il tasto **F2**.
Ricordatevi di premere il tasto funzione **F2** tutte le

ELENCO COMPONENTI LX.1206

R1 = 1.000 ohm 1/4 watt
R2 = 100 ohm 1/4 watt
R3 = 1.000 ohm 1/4 watt
R4 = 100 ohm 1/4 watt
R5 = 1.000 ohm 1/4 watt
R6 = 100 ohm 1/4 watt
R7 = 1.000 ohm 1/4 watt
R8 = 100 ohm 1/4 watt
R9 = 220 ohm 1/4 watt
R10 = 220 ohm 1/4 watt
R11 = 220 ohm 1/4 watt
R12 = 220 ohm 1/4 watt
R13 = 10.000 ohm 1/4 watt
R14 = 10.000 ohm 1/4 watt
R15 = 10.000 ohm 1/4 watt
R16 = 10.000 ohm 1/4 watt
C1 = 47.000 pF pol. 400 V.
C2 = 47.000 pF pol. 400 V.
C3 = 47.000 pF pol. 400 V.
C4 = 47.000 pF pol. 400 V.
C5 = 100.000 pF poliestere
C6 = 100.000 pF poliestere
C7 = 100.000 pF poliestere
C8 = 100.000 pF poliestere
TRC1 = triac tipo 500 V. 5 A.
TRC2 = triac tipo 500 V. 5 A.
TRC3 = triac tipo 500 V. 5 A.
TRC4 = triac tipo 500 V. 5 A.
OC1 = fototriac tipo MOC.3020
OC2 = fototriac tipo MOC.3020
OC3 = fototriac tipo MOC.3020
OC4 = fototriac tipo MOC.3020
P1 = pulsante
P2 = pulsante
P3 = pulsante
P4 = pulsante

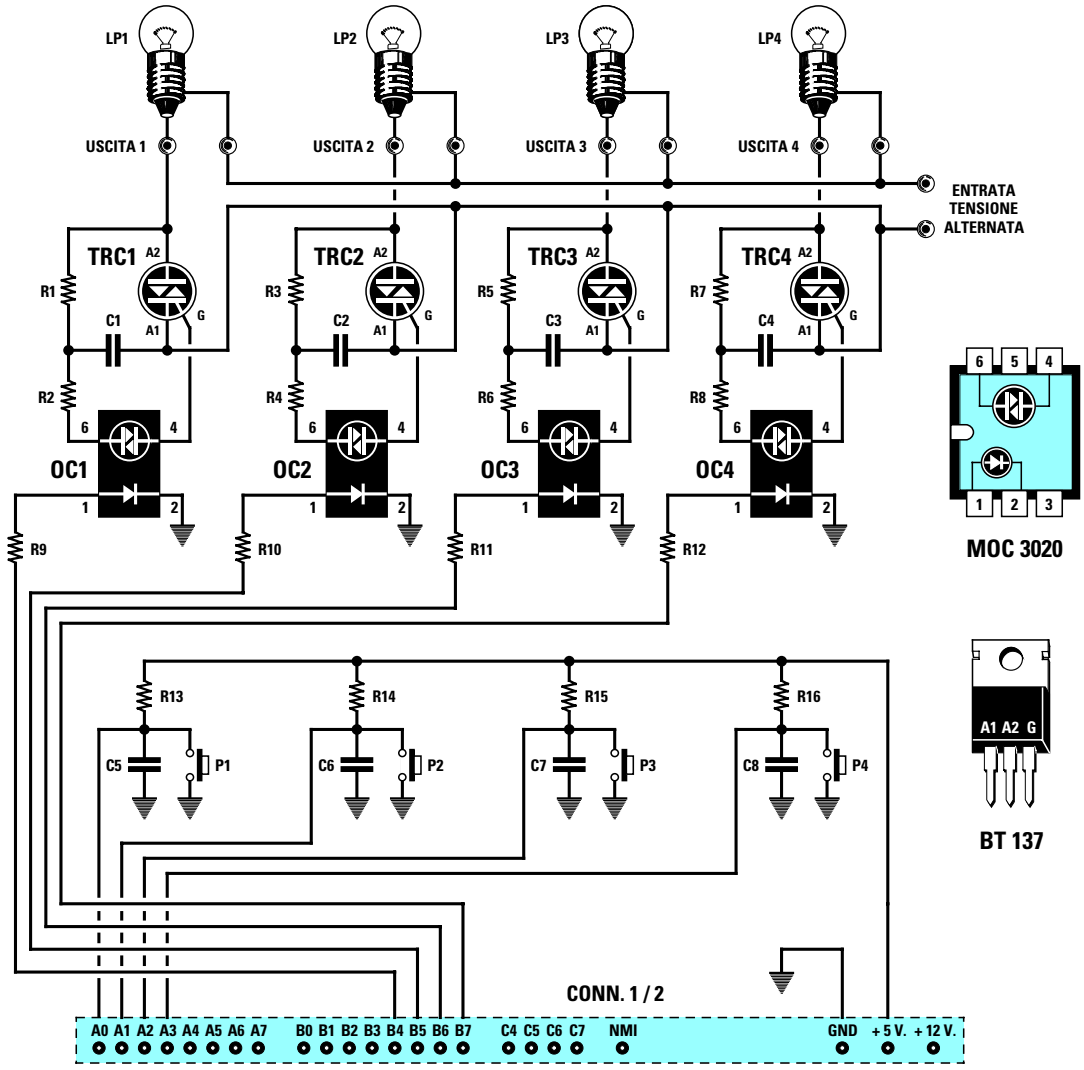


Fig.7 Schema elettrico della scheda Triac siglata LX.1206 e connessioni del fototriac visto da sopra e del triac BT.137. Sulle "uscite" dei Triac dovrete collegare delle lampadine o altre apparecchiature elettriche che funzionino con il valore della tensione alternata che applicherete sulla morsettiera d'ingresso (vedi fig.8).

volte che eseguirete una **variazione** o farete un'**aggiunta** nel programma, diversamente queste non verranno **memorizzate**.
Una volta **memorizzato**, lo dovrete anche **assemblare** premendo i tasti:

ALT T poi il tasto **A** (vedi fig.6)

Se non avrete commesso degli **errori** nello scrivere un'istruzione, sul monitor vi apparirà la scritta:

*****success*****

Se, in sostituzione di questa scritta, vi apparirà un **numero** in basso a sinistra sul monitor, significa che nella **riga** del programma che avete modificato o variato avete commesso un **errore**, ad esempio avete scritto **lp** anzichè **ld.**

Un **errore** che molti commettono è quello di caricare direttamente nel **registro Y** il contenuto del **registro X** scrivendo:

	ld	**y,x**	

Per caricare il contenuto del registro **X** nel registro **Y** dovrete **prima** caricare il registro **X** nell'accumulatore **A**, poi caricare su questo il registro **Y**, quindi questa istruzione andrà scritta su due righe:

	ld	**a,x**	
	ld	**y,a**	

Tutti i files una volta assemblati diventeranno dei **.HEX**, cioè convertiti in **esadecimale**, perchè questo è il solo linguaggio che il microprocessore è in grado di interpretare.
Vi ricordiamo che i files **.HEX** non potranno più essere modificati.

SCHEMA ELETTRICO

Lo schema elettrico di questa scheda per **Triac** è visibile in fig.7.
In teoria, potevamo sfruttare tutte le porte **A-B-C** del microprocessore **ST6** come **uscite** ed in tal modo potevamo inserire in questa scheda ben **20 Triac** con il micro da **28** piedini e **12 Triac** con il micro da **20** piedini.

In pratica, non potremo mai farlo, perchè dobbiamo tenere impegnate diverse porte **A** e **B** per gestire anche le altre schede che inseriremo nel **bus** assieme alla scheda Triac o Relè.

Quando sui piedini della porta **B** siglati **B4-B5-B6-B7** (vedi fig.7) apparirà un **livello logico 1**, vale a dire una **tensione positiva**, questa polarizzerà il **diodo emittente** presente all'interno dei fotoaccoppiatori siglati **OC1-OC2-OC3-OC4** e di conseguenza si **ecciterà** il Triac ad essi collegato.
Questi quattro **fotoaccoppiatori** li abbiamo utilizzati per separare elettricamente l'uscita del **microprocessore** dalla tensione che applicheremo sulle due boccole visibili a destra indicate con la scritta **Entrata Tensione Alternata**.
Per alimentare i Triac potremo usare qualsiasi tensione **alternata** partendo da un minimo di **4,5 volt** per arrivare ad un massimo di **220 volt**.
Ovviamente sulle **morsettiere** indicate **uscita 1 - uscita 2 - uscita 3 - uscita 4** dovremo applicare delle lampadine, dei motorini in alternata, o qualsiasi altra apparecchiatura elettrica che funzioni con il valore di **tensione** utilizzata per alimentare i **Triac**.
Non utilizzate una tensione **continua** per alimentare i Triac. Se volete usare una tensione **continua** dovrete necessariamente servirvi della **scheda RELÈ** siglata LX.1205 presentata nella rivista N.179.

REALIZZAZIONE PRATICA

Sul circuito stampato siglato **LX.1206** dovrete montare tutti i componenti visibili in fig.8.
Per il montaggio vi consigliamo di saldare sul lato opposto di questo stampato il connettore **maschio** a **1 fila** provvisto di **24 terminali** (vedi **CONN 1-2**) e vicino alle due morsettiere **Uscita 1** e **Uscita 4** i due connettori a **1 fila** provvisti di soli **4 terminali**, che vi serviranno per innestare i **connettori femmina** presenti sulla scheda **bus** siglata **LX.1202**.
Nella parte di stampato visibile in fig.8 inserirete i quattro zoccoli per i **fotoaccoppiatori**, tutte le **resistenze**, poi **condensatori** e le **5 morsettiere** a due poli.
Proseguendo nel montaggio, inserirete i **4 Triac** siglati **TRC1-TRC2-TRC3-TRC4**, rivolgendo la parte **metallica** del loro corpo verso **destra** come visibile nello schema pratico di fig.8.

Completata questa operazione, potrete inserire negli zoccoli i quattro **fotoaccoppiatori** rivolgendo verso **sinistra** il lato del loro corpo provvisto del **piccolo foro** di riferimento.

INSERIMENTO SCHEDA nel BUS

Questa scheda la potrete inserire nel **bus** siglato **LX.1202**, indifferentemente su un qualsiasi connettore **femmina** presente su questo stampato e lo stesso dicasi per la **scheda** dei **display** che utilizzerete per visualizzare i relativi **tempi**.
Sulle morsettiere di destra indicate **Entrata Tensione Alternata** dovrete applicare la tensione che servirà per alimentare i **motorini** o le **lampade** che applicherete sulle quattro **morsettiere d'uscita**.
Per le prime prove vi consigliamo di utilizzare una **tensione alternata** di **14 volt** fornita dall'apposito alimentatore **LX.1203**, presentato sulla rivista **N.179**, collegando alle uscite dei Triac delle normali lampadine da **12 volt** massimo **3 watt.**
Se lo ritenete opportuno potrete anche entrare nella morsettiera di destra con una tensione **alternata** di **220 volt** collegando all'uscita dei Triac delle lampadine da **220 volt**, ma **attenzione**, se userete la tensione di rete ricordatevi di **non toccare mai** le parti metalliche dei Triac e le uscite dei fotoaccoppiatori per evitare che la tensione dei **220 volt** si scarichi sul vostro corpo.
Con questa scheda potrete utilizzare tutti i programmi che abbiamo usato per la **scheda RELÈ** siglata LX.1025. A fine articolo abbiamo riportato come si possono modificare i tempi e le funzioni.

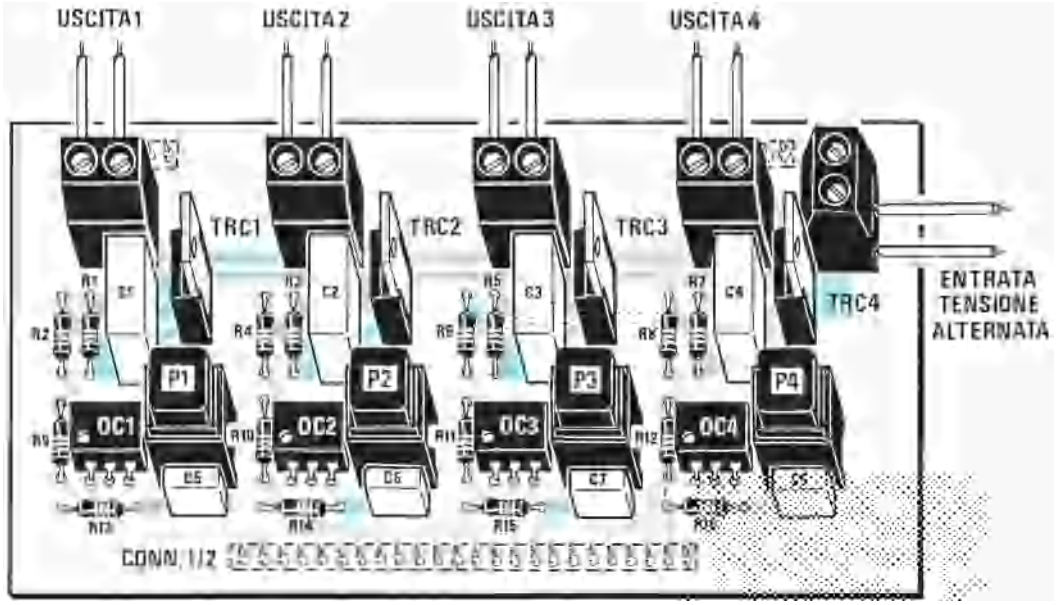


Fig.8 Schema pratico di montaggio della scheda LX.1206. Il circuito stampato che vi forniremo è un "doppia faccia" con fori metallizzati, quindi non cercate mai di allargare questi fori con una punta da trapano perchè, così facendo, asportereste il sottile strato di rame applicato per via galvanica al loro interno e che è necessario per collegare le piste presenti sotto al circuito stampato con quelle presenti sopra.

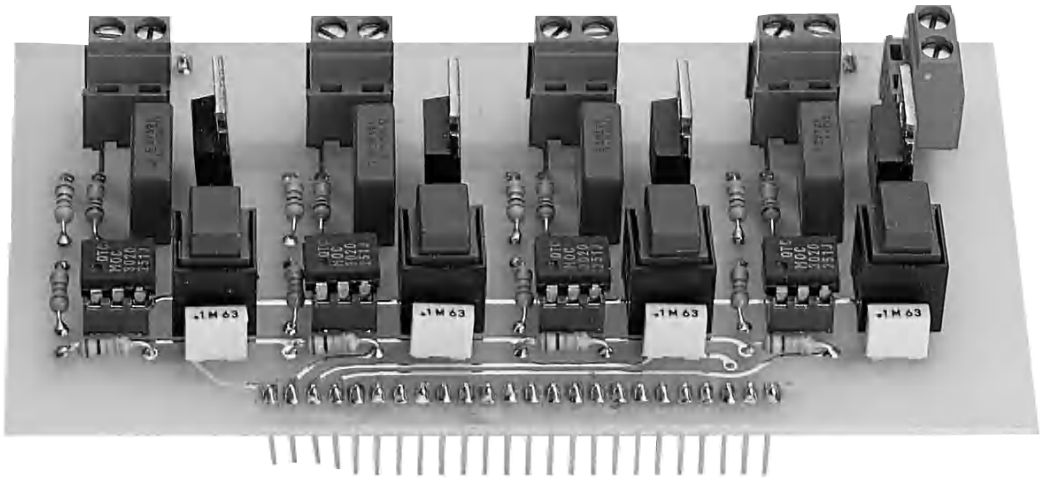


Fig.9 Foto della scheda LX.1206 come si presenterà a montaggio ultimato. Questa scheda andrà posta sul Bus siglato LX.1202 (vedi fig.10) assieme al microprocessore ST6 che avrete già programmato con uno dei programmi necessari per gestirlo. Consigliamo di rileggere le riviste N.172/173 - 174 - N.175/176 - N.179 per sapere come si deve procedere per memorizzare un micro e per settare tutte le PORTE.

INSTALLAZIONE programmi nell'HARD-DISK

Riportiamo in forma condensata quanto già scritto nelle riviste numero **172/173 - 174 - 175/176 - 179**, perchè ad alcuni lettori potrebbe essere sfuggito uno o più di questi numeri in cui sono apparsi i seguenti articoli:

Programmatore per micro ST6
Circuito test per microprocessore ST62E10
Impariamo a programmare i micro ST6
Lampada per cancellare Eprom
Impariamo a programmare i micro ST6
Bus per testare i micro ST6
Scheda test per Relè
Scheda test per display

Facciamo presente che sono ancora reperibili presso la nostra Sede alcune centinaia di copie delle riviste sopra elencate, quindi chi ne fosse sprovvisto e volesse avere la serie completa di articoli inerenti il **microprocessore ST6**, potrà richiedercele fino al loro esaurimento.

Per copiare il dischetto relativo all'**ST6** nell'Hard-Disk di un computer dovrete procedere come segue:

1 - Uscite da qualsiasi programma tipo **Windows - Pcshell - Norton**, ecc.

2 - Quando sul monitor apparirà il prompt **C:\>**, inserite nel drive floppy **A** il dischetto contenente i programmi, poi digitate:

C:\>A: poi premete il tasto Enter

e vi apparirà **A:\>**

A questo punto potrete scrivere:

A:\>installa poi premete Enter

Subito vi apparirà sul monitor la richiesta su quale **directory** volete installare il contenuto del disco.
La directory da noi predefinita è **ST6**, quindi se digiterete **Enter** (vedi fig.11) il programma creerà u-

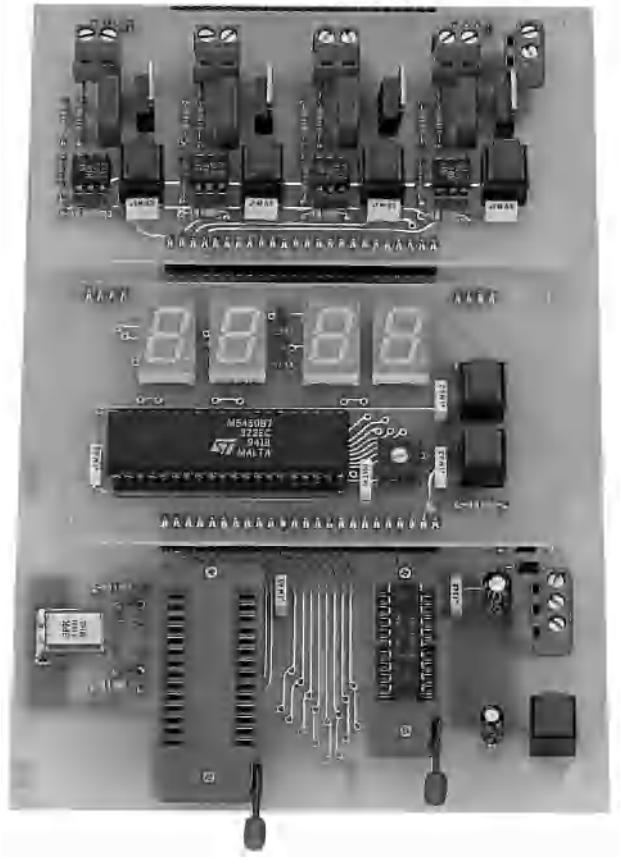


Fig.10 A quanti desiderano seguire questo corso sugli ST6 consigliamo di procurarsi il kit LX.1170 che servirà per trasferire i programmi che abbiamo assemblato da .ASM in .HEX, più il BUS siglato LX.1202 completo del suo alimentatore LX.1203. Questo BUS LX.1202 servirà, prossimamente, per controllare la scheda con un display LCD numerico e la scheda con un display LCD alfanumerico.

na directory con questo nome e copierà nell'**Hard-Disk** tutto il contenuto del dischetto **scompattando** i files.

Se volete trasferire il contenuto su una **diversa** directory, ad esempio **LX1202**, quando sul monitor vi apparirà:

C:\ST6

dovrete **cancellare** il nome della **directory** precedente, cioè **ST6** (vedi fig.11), scrivendo in sua sostituzione **LX1202** come qui sotto riportato:

C:\LX1202 poi premete Enter

Ricordatevi che qualsiasi **nome** di directory sceglierete non dovrete mai superare gli **8 caratteri** e nemmeno utilizzare dei caratteri che il sistema operativo **DOS** non accetta, quali ad esempio **barre**, **punto interrogativo**, **segno di uguale**, ecc.
Se farete questo **errore** il **DOS** ve lo segnalerà immediatamente con la scritta **ERROR** su uno sfondo rosso.

Ad alcuni lettori che ci avevano segnalato di non riuscire a trasferire il contenuto del dischetto nell'**Hard-Disk** perchè sul computer appariva subito la scritta **error** o la **mancanza** di un file, abbiamo consigliato di creare prima una **directory** con il nome **ST6**, poi di trasferire su questa tutto il contenuto del dischetto e da ultimo di **scompattare** il programma procedendo come segue:

Quando sul monitor appare il prompt **C:\>** dovrete scrivere:

C:\>MD ST6 poi premete Enter

In questo modo riapparirà il prompt **C:\>** e a questo punto dovrete inserire nel driver il disco che vi abbiamo fornito con la sigla **DF1170.3** o **DF1202.3** (il contenuto dei due dischetti è identico) e scrivere quanto segue:

C:\>COPY A: *.* C:\ST6 poi premere Enter

Nota = Rispettate gli **spazi**: per agevolarvi abbiamo colorato in **azzurro** le scritte che il computer farà apparire sul monitor, senza colore quelle che dovrete scrivere voi inserendo come **spaziatura** un rettangolo in **azzurro**.

Terminato di copiare il contenuto del dischetto nell'Hard-disk, apparirà nuovamente la scritta **C:\>**, ma non potrete ancora usare il programma **ST6**

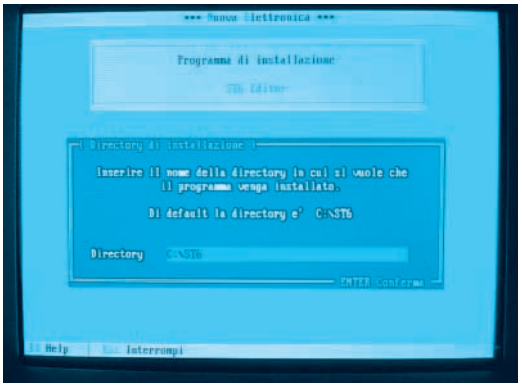


Fig.11 Quando trasferite il programma dal dischetto nell'Hard-Disk, se volete chiamare la directory ST6 in LX1202, quando vi appare questa riga sostituite C:\ST6 con la scritta C:\LX1202 e poi premete Enter.

Fig.12 Immediatamente tutti i files "compattati" si scompatteranno. Ricordatevi che il programma occupa circa 1 MEGA di memoria. Non premete nessun tasto durante la fase di scompattazione.

Fig.13 Solo quando sul monitor apparirà la scritta BUON DIVERTIMENTO potrete premere un qualsiasi tasto per uscire. Per richiamare il programma dovrete scrivere C:\>ST6 poi C:\ST6>ST6 ed Enter.

perchè non l'avete ancora **scompattato**.
Per farlo dovrete scrivere:

C:\>CD ST6 poi premete Enter

Quando vi apparirà **C:\ST6>**, scrivete:

C:\ST6>installa poi premete Enter

Non appena premerete il tasto **Enter**, sul monitor vi apparirà la scritta che questa **directory esiste già**, ma di ciò non preoccupatevi e **premete nuovamente Enter** e quando vi sarà chiesta la **conferma** premete per una **seconda** volta il tasto **Enter**.

Subito vedrete sul monitor tutti i nomi dei files che, abbastanza velocemente, si stanno **scompattando** (vedi fig.12)**.**

I PROGRAMMI

Prima di trasferire tutti i files con l'estensione **.ASM** nella memoria di un micro **ST6**, li dovrete **assemblare** per ottenere un file in estensione **.HEX**
Eseguita questa operazione, nel computer troverete sempre due identici **files**, uno **ASM** ed uno **HEX**.
Se tenterete di trasferire dall'Hard-disk alla memoria di un micro **ST6** un file **ASM**, il computer segnalerà **errore** con questa scritta in inglese:

error can't open file

oppure con:

enter name of source HEX file

Per sapere se il file richiesto risulta convertito nell'estensione **HEX**, potrete procedere come segue.
Quando sul monitor del computer apparirà la finestra dell'**Editor** (vedi fig.1) premete i tasti **ALT F**, poi **F3** e quando sul monitor vi apparirà la fascia con la scritta:

***.ASM**

sostituitela con:

***.HEX** poi premete Enter

e, in questo modo, vi appariranno tutti i files in **.HEX**.

I **kit** che vi serviranno per provare tutte le nostre **schede sperimentali** sono i seguenti:

LX.1170 = Questo kit, pubblicato nella rivista **N.172/173**, serve per trasferire i programmi che abbiamo **assemblato** e convertito da **.ASM** in **.HEX** dall'Hard-disk del computer alla memoria di un **ST6 vergine**. Questo kit va collegato alla presa uscita **parallela** del computer.

LX.1202 = Questo kit, pubblicato nella rivista **N.179**, serve per ricevere tutte le **schede sperimentali** che abbiamo già pubblicato e quelle che pubblicheremo in seguito. Questa scheda, che **non va** collegata al computer, andrà alimentata con il kit **LX.1203**.

LX.1203 = Questo kit, pubblicato nella rivista **N.179**, serve per alimentare la scheda **LX.1202** e tutte le **schede sperimentali** che inserirete in questa stessa scheda.

LX.1204 = Questo kit, pubblicato nella rivista **N.179**, provvisto di **4 display** a **sette segmenti** serve per realizzare dei cronometri-orologi-timer, ecc. Questa scheda va inserita nei connettori presenti nella scheda **LX.1202**.

LX.1205 = Questo kit, pubblicato nella rivista **N.179**, provvisto di **4 relè** serve per alimentare lampade-motorini o accendere qualsiasi apparecchiatura elettronica. Questa scheda va inserita nei connettori presenti nella scheda **LX.1202**.

LX.1206 = Questo kit, pubblicato nella rivista **N.180**, provvisto di **4 Triac** serve per alimentare delle lampade - motorini o altre apparecchiature elettroniche che funzionano con **tensioni alternate**. Questa scheda va inserita nei connettori presenti nella scheda **LX.1202**.

CRONOMET.HEX

Questo programma è un semplice **cronometro**, quindi per visualizzare i tempi occorre inserire nel **bus LX.1202** la **sola** scheda dei display siglata **LX.1204**.
Se nel bus inserirete le schede dei **relè** o dei **triac**, non potrete renderle attive perchè nel programma non è presente nessuna istruzione per gestirle.
Una volta caricato su un micro **ST6** vergine il programma **CRONOMET.HEX** ed inserito nello zoccolo presente sulla scheda bus **LX.1202**, appena

alimenterete il circuito sui **4 display** apparirà il numero:

00:00

Premendo il pulsante **P1** il micro comincerà a contare in avanti ad intervalli di tempo di un **secondo**, quindi sui display vedrete apparire i numeri:

00:01 - 00:02 - 00:03 ecc.

Sui primi due display di sinistra vedrete i **minuti** e sui display di destra i **secondi**.
I **due led** che separano i display dei minuti e dei secondi lampeggeranno con una cadenza di un secondo.
Come noterete, quando si è raggiunto un tempo di **00:59 secondi**, subito dopo si passerà al tempo successivo di **01:00**, cioè **1 minuto** e **00 secondi**.
Il massimo numero che potrete visualizzare sarà quindi di **99 minuti** e **59 secondi**, dopodichè apparirà **00:00**.
Se in fase di conteggio premerete **P1**, il conteggio si **bloccherà** sul tempo raggiunto e premendolo nuovamente questo ripartirà dal numero sul quale si era fermato.
Se invece premerete il pulsante **P2**, il conteggio ripartirà da **zero**, cioè il tempo visualizzato si **azzererà**.

DISPLAY.HEX

Questo programma serve solo per far capire come si possa **visualizzare** il numero desiderato sui **4 display** della scheda siglata **LX.1204**.
Una volta caricato su un micro **ST6** vergine il programma **DISPLAY.HEX** ed inserito nello zoccolo presente sulla scheda bus **LX.1202**, non appena alimenterete il circuito sui **4 display** apparirà il numero:

12:34

e nient'altro.
Questo programma l'abbiamo composto soltanto per farvi capire come si deve scrivere una istruzione per gestire in modo seriale l'integrato **M.5450**.
Le righe da utilizzare per cambiare questo **numero** sono quelle numerate dalla numero **159** alla numero **162** del listato **DISPLAY.ASM**:

```
ldi  bcd3,1  ;159 accende 1 sul display 1
ldi  bcd4,2  ;160 accende 2 sul display 2
ldi  bcd1,3  ;161 accende 3 sul display 3
ldi  bcd2,4  ;162 accende 4 sul display 4
```

Quindi basta cambiare il numero **dopo la virgola** in questi registri per modificare il numero visualizzato.
Poichè questo programma fa molto poco, vi consigliamo di visualizzarlo solo sul monitor e di non memorizzarlo su un **ST6**.

OROLOGIO.HEX

Questo programma è un semplice **orologio.**
Per poter visualizzare le **ore** ed i **minuti** dovrete inserire nel **bus LX.1202** la **sola** scheda dei display siglata **LX.1204**.
Se nel bus inserirete le schede dei **relè** o dei **triac**, non potrete renderle attive, perchè nel programma non è presente nessuna istruzione per gestirle.
Una volta caricato su un micro **ST6** vergine il programma **OROLOGIO.HEX** ed inserito nello zoccolo presente nella scheda bus **LX.1202**, non appena alimenterete il circuito sui **4 display** apparirà il numero:

00:00

I primi due display di **sinistra** segneranno le **ore**, mentre i due di **destra** i **minuti**.
I due **diodi led** che separano i due display lampeggeranno con una cadenza di **1 secondo**.
Come noterete, raggiunte le **ore 23** ed i **59 minuti**, dopo **1 minuto** si passerà alle **24 ore** che verranno visualizzate con **00:00**.
Per mettere a **punto** le **ore** dell'orologio si utilizzerà il pulsante **P2** e, per mettere a punto i **minuti**, il pulsante **P1**.
Facciamo presente che potrete solo **far avanzare** i **numeri** e non **indietreggiare**.

RELE.HEX

Questo programma serve solo per far eccitare dei **relè** o dei **triac**, quindi nella scheda **bus LX.1202** potrete inserire la **sola** scheda dei Relè siglata **LX.1205** o la **sola** scheda dei Triac siglata **LX.1206**.
Se inserirete nel bus anche la scheda dei **display** siglata **LX.1204** non potrete renderla attiva, perchè nel programma non è presente nessuna istruzione per gestirla.
Una volta caricato su un micro **ST6** vergine il programma **RELE.HEX** ed inserito nello zoccolo presente sulla scheda bus **LX.1202**, non appena ali-

menterete il circuito tutti i relè risulteranno **diseccitati**.
Non appena premerete uno o più dei pulsanti da **P1** a **P4** si ecciterà il relè o il triac corrispondente. Premendolo una **seconda** volta, il relè o il triac si **diseccciterà**.

TEMPOR.HEX

Questo programma è un semplice **temporizzatore** con conteggio all'**indietro**.
Nel **bus LX.1202** dovrete inserire, oltre alla scheda display siglata **LX.1204**, anche la scheda Relè siglata **LX.1205**, oppure la scheda Triac siglata **LX.1206**.
Una volta caricato su un micro **ST6** vergine il programma **TEMPOR.HEX** ed inserito nello zoccolo presente nella scheda bus **LX.1202**, non appena alimenterete il circuito, tutti i relè o i triac risulteranno **diseccitati** e sui **4 display** vedrete apparire il numero:

03:00

che indica **03 minuti** e **00** secondi.
Immediatamente, partendo da questo **numero**, il conteggio inizierà a contare all'**indietro** con una cadenza di **un secondo**, quindi sui display vedrete i numeri:

02:59 - **02:58** - **02:57** ecc.

Quando apparirà il numero **00:00**, si ecciterà il solo relè **1** presente nella scheda **LX.1205** oppure il solo triac **1** presente nella scheda **LX.1206**.
Per ricominciare il ciclo basterà premere il pulsante **P1** presente sulla scheda bus **LX.1202**.
Tutti i pulsanti presenti sulla scheda del relè o del triac non risultano **attivati**.

PER cambiare i TEMPI

Per cambiare il tempo da noi prefissato **03:00** basterà modificare i valori impostati sulle righe **239** e **240** nel listato del programma **TEMPOR.ASM**.
Come tempo **massimo** potrete partire da **99 minuti** e **59 secondi**.
Attualmente sulla riga **240** che è la riga dei **minuti** troverete riportato il numero **3**:

```
ldi  minuti,3  ;240 carica 3 minuti
```

quindi se volete partire da **12** minuti basterà semplicemente scrivere **12** al posto di **3** come qui sotto riportato:

```
ldi  minuti,12  ;240 carica 12 minuti
```

Se volete cambiare anche i **secondi** dovrete modificare il numero sulla riga **239** che attualmente è **0**:

```
ldi  secondi,0  ;239 carica 0 secondi
```

Ammesso che ai **12 minuti** già presenti vogliate sommare **28 secondi**, dovrete scrivere nella riga **239** il numero **28** come qui sotto riportato:

```
ldi  secondi,28 ;239 carica 28 secondi
```

Così facendo il conteggio partirà da **12:28** e quando raggiungerà lo **00:00** si **ecciterà** il relè **1** o il triac **1**.

Nota = Anche se nella riga **241** troverete:

```
ldi  ore,0      ;241 carica 0 ore
```

questo parametro **ore** non viene utilizzato, pertanto questa istruzione non dovrete **mai modificarla**.

Se in sostituzione del relè **1** volete eccitare un altro relè ad esempio il relè **2**, o un corrispondente triac, dovrete modificare la riga **262**:

```
loop3  set  4,port_b  ;262 accende RL1
```

sostituendo dopo il **set** il numero **4** con il numero **5** come qui sotto riportato:

```
loop3  set  5,port_b  ;262 eccita RL2
```

Se volete eccitare il relè **RL3**, dovrete sostituire il numero **4** con il numero **6**:

```
loop3  set  6,port_b  ;262 eccita RL3
```

Se volete eccitare il relè **RL4** dovete sostituire il numero **4** con il numero **7**:

```
loop3  set  7,port_b  ;262 eccita RL4
```

Se volete eccitare contemporaneamente tutti e **4** i relè, dovrete aggiungere tutte queste righe:

```
loop3  set  4,port_b  ;262 eccita RL1
       set  5,port_b  ;262.1 eccita RL2
       set  6,port_b  ;262.2 eccita RL3
       set  7,port_b  ;262.3 eccita RL4
```

In pratica, se vi interessa eccitare i soli relè **RL1**, **RL3** ed **RL4** dovrete scrivere queste tre righe:

```
loop3   set   4, port_b   ;262 eccita RL1
        set   6, port_b   ;262.1 eccita RL3
        set   7, port_b   ;262.2 eccita RL4
```

Vi ricordiamo che tutte le volte che modificherete un programma, lo dovrete **salvare** digitando il tasto **F2**, poi lo dovrete **riassemblare** premendo i tasti **ALT T**, poi il tasto **A**, dopodichè lo potrete trasferire nella **memoria** di un **ST6**.

TIMER.HEX

Questo programma è un semplice **temporizzatore** con conteggio in **avanti**.
Nel **bus LX.1202** dovrete inserire oltre alla scheda display siglata **LX.1204**, anche la scheda relè siglata **LX.1205**, oppure la scheda triac siglata **LX.1206**.
Una volta caricato su un micro **ST6** vergine il programma **TIMER.HEX** ed inserito nello zoccolo presente sulla scheda bus **LX.1202**, non appena alimenterete il circuito, tutti i relè o i triac risulteranno **diseccitati** e sui **4 display** vedrete apparire il numero:

00:00

Immediatamente, partendo da questo **numero**, il conteggio inizierà a contare in **avanti** con una cadenza di **un secondo**, quindi sui display vedrete i numeri salire:

00:01 - **00:02** - **00:03** ecc.

e quando raggiungerete il numero **03:00**, corrispondente a **3 minuti** e **00 secondi**, subito si ecciterà il relè **1** presente nella scheda **LX.1205** oppure il solo triac **1** presente nella scheda **LX.1206**.
Per ricominciare il ciclo basterà premere il pulsante **P1** presente sulla scheda bus **LX.1202**.
Tutti i pulsanti presenti sulla scheda dei relè o dei triac non risultano **attivati**.

PER cambiare i TEMPI

Per cambiare il tempo da noi prefissato **03:00** basterà modificare i valori impostati sulle righe **248** e **252** nel listato del programma **TEMPOR.ASM**.
Come tempo massimo potrete raggiungere i **99 minuti** e **59 secondi**.

Attualmente sulla riga **248**, che è la riga dei **secondi**, troverete riportato il numero **0**:

```
        cpi   a,0    ;248 compara con zero
```

e nella riga **252**, che è quella dei **minuti**, troverete riportato:

```
        cpi   a,3    ;252 compara con 3
```

Se volete ad esempio eccitare il relè **RL1** dopo **15 secondi** dall'accensione, basterà inserire nella riga **248**:

```
        cpi   a,15   ;248 compara con 15 secondi
```

ed inserire il numero **0** nella riga **252** dei **minuti**:

```
        cpi   a,0    ;252 compara con 0 minuti
```

Con queste modifiche, quando sul display apparirà il numero **00:15** il relè **RL1** si ecciterà.
Se volete far eccitare il relè dopo **30 minuti** e **5 secondi** basterà inserire il numero **5** nella riga **248**:

```
        cpi   a,5    ;248 compara con 5 secondi
```

ed inserire il numero **30** nella riga **252** dei **minuti**:

```
        cpi   a,30   ;252 compara con 30 minuti
```

Con queste modifiche, quando sul display apparirà il numero **30:05** il relè **RL1** si ecciterà.

Nota = Anche se nella riga **256** troverete riportato:

```
        ldi   ore,0  ;256 carica 0 ore
```

questo parametro **ore** non viene utilizzato, pertanto non dovrete **mai modificare** questa istruzione.

Se in sostituzione del relè **1** volete eccitare un altro relè, ad esempio il relè **2**, o un corrispondente triac, dovrete modificare la riga **259**:

```
loop3   set   4,port_b   ;259 eccita RL1
```

sostituendo dopo il **set** il numero **4** con il numero **5** come qui sotto riportato:

```
loop3   set   5,port_b   ;259 eccita RL2
```

Se volete eccitare il relè **RL3** dovrete sostituire il numero **4** con il numero **6**:

```
loop3  set  6,port_b   ;259 eccita RL3
```

Per eccitare il relè **RL4** dovrete sostituire il numero **4** con il numero **7**:

```
loop3  set  7,port_b   ;259 eccita RL4
```

Per eccitare contemporaneamente tutti e **4** i relè dovrete aggiungere le seguenti righe:

```
loop3  set  4,port_b   ;259 eccita RL1
       set  5,port_b   ;259.1 eccita RL2
       set  6,port_b   ;259.2 eccita RL3
       set  7,port_b   ;259.3 eccita RL4
```

Se vi interessa eccitare i soli relè **RL1**, **RL3** ed **RL4**, dovrete scrivere queste tre righe:

```
loop3  set  4,port_b   ;259 eccita RL1
       set  6,port_b   ;259.1 eccita RL3
       set  7,port_b   ;259.2 eccita RL4
```

Vi ricordiamo che tutte le volte che modificherete un programma, lo dovrete **salvare** premendo il tasto **F2**, poi lo dovrete **riassemblare** premendo i tasti **ALT T**, poi il tasto **A**, dopodichè lo potrete trasferire nella **memoria** di un **ST6**.

TRIAC.HEX

Questo programma è una dimostrazione di come si possano **eccitare** in sequenza dei **Triac** o dei **Relè**.
Nel **bus LX.1202** dovrete inserire la **sola** scheda siglata **LX.1206** o, in sua sostituzione, quella dei relè siglata **LX.1205**.
Una volta caricato su un micro **ST6** vergine il programma **TRIAC.HEX** ed inserito nello zoccolo presente sulla scheda bus **LX.1202**, non appena alimenterete il circuito vedrete eccitarsi in **sequenza**, con un intervallo di **1 secondo**, i quattro **triac** o i quattro **relè** se avrete inserito la scheda **LX.1205**.

Nota = In questo programma i pulsanti **P1-P2-P3-P4** presenti nella scheda dei triac o dei relè non risultano **attivati**, quindi anche se li premerete non accadrà **nulla**.

Questo programma può essere modificato per eccitare i triac in senso inverso a quello indicato, modificando anche i **tempi** oppure facendo accendere delle coppie di triac, ecc., modificando semplicemente il **numero binario** riportato nelle righe **74-76-78-80-82-84-86-88** .

```
     Idi  port_b,10000000b  ;74 eccita il triac TRC4
     Idi  port_b,01000000b  ;76 eccita il triac TRC3
     Idi  port_b,00100000b  ;78 eccita il triac TRC2
     Idi  port_b,00010000b  ;80 eccita il triac TRC1
     Idi  port_b,00100000b  ;82 eccita il triac TRC2
     Idi  port_b,01000000b  ;84 eccita il triac TRC3
     Idi  port_b,10000000b  ;86 eccita il triac TRC4
     Idi  port_b,00000000b  ;88 diseccita tutti i triac
```

Come noterete, dopo la dicitura **port_b** ci sono **otto numeri**, ma quelli che dovrete modificare sono solo i primi **quattro**, cioè **1000 - 0100 - 0010 - 0001**.
Se nella riga **88** sostituirete i quattro **0000** con **1111**, tutti i quattro triac si **ecciteranno** anzichè **disecciarsi**.
Per far **lampeggiare** per una infinità di volte il solo **TRC4**, dovrete scrivere in tutte le righe **74-76-78-80-82-84-86** solo **1000**.
Per accendere contemporaneamente **TRC4-TRC3** dovrete scrivere nelle righe **74-76** il numero **1100.**

PER cambiare i TEMPI

Per modificare i **tempi** di intervallo tra l'accensione di un triac e quella del successivo, occorre modificare la **subroutine** chiamata **delay**.
Anche se prima non le abbiamo riportate, tra una riga e l'altra delle **74-76-78-80-82-84-86-88** troverete una istruzione **call delay** che chiama una **subroutine**:

```
     Idi  port_b,10000000b  ;74 eccita il triac TRC4
     call delay             ;75 esegue ritardo
     Idi  port_b,01000000b  ;76 eccita il triac TRC4
```

questa **subroutine** la troverete nelle righe **59** - **60**.

```
     Idi  x,255             ;59 carica in x 255
del1 Idi  y,255             ;60 carica in y 255
```

Il **massimo** ritardo che potete ottenere è di circa **1 secondo**, perchè disponendo di registri **X-Y** ad **8 bit** non potrete mettere un numero **maggiore** di **255.**
Volendo **ridurre** il ritardo, dovrete inserire dei numeri **minori**, ad esempio per ottenere all'incirca **1/2 secondo** dovrete caricare nella **riga 59** il numero **127** come qui sotto riportato:

```
     Idi  x,127             ;59 carica in x 127
del1 Idi  y,255             ;60 carica in y 255
```

Se volete ottenere **1/4** di **secondo** dovrete cambiare le due righe come segue:

```
      ldi   x,127      ;59 carica in x 127
del1  ldi   y,127      ;60 carica in y 127
```

Se anzichè **ridurli** li voleste **aumentare**, potrete farlo utilizzando questo "trucchetto".
Tutte le istruzioni che trovate inserite tra le righe delle istruzioni dal numero **74** alla **88** con la parola **call delay**, le dovrete scrivere **due-tre-quattro** o più volte, ad esempio:

```
      ldi   port_b,10000000b   ;74 eccita TRC4
      call  delay              ;75 esegue ritardo
      call  delay              ;raddoppia tempo
      call  delay              ;triplica tempo
      ldi   port_b,01000000b   ;76 eccita TRC3
```

Quindi inserendo più o meno righe di **call delay** tra una riga e l'altra potrete variare i tempi di eccitazione tra un triac e l'altro.

CLOCK.HEX

Questo programma **CLOCK** anche se funziona come **orologio** è totalmente diverso dal precedente programma **OROLOGIO**, perchè oltre a visualizzare le **ore** e i **minuti** permette di **eccitare** un **relè** o un **triac** ad un'**ora** prestabilita e di **diseccitarlo** dopo un tempo che voi stessi potrete prefissare modificando alcune righe del programma.
Per farlo funzionare occorre inserire nel **bus LX.1202** la scheda dei display siglata **LX.1204** e quella dei relè siglata **LX.1205**, oppure quella dei triac siglata **LX.1206.**
Prima di spiegarvi quali righe dovrete modificare, consigliamo ai meno esperti di **leggere attentamente** tutto l'articolo, dopodichè potranno modificare i **parametri** nelle sole righe che noi indicheremo.
Come abbiamo accennato, il programma **CLOCK.HEX** ci dà la possibilità di **eccitare** o **diseccitare** uno o più **relè** anche contemporaneamente, su orari che noi stessi potremo stabilire, purchè non si superino più di **8 cicli** o **periodi** nell'arco delle **24 ore**.
Questo programma potrà servire per **accendere** o **spegnere** una o più caldaie, delle insegne luminose ad orari prestabiliti, ecc.
Per mettere a **punto** le **ore** dell'orologio si utilizzerà il pulsante **P2** e per mettere a punto i **minuti** il pulsante **P1**.

Facciamo presente che è possibile soltanto far **avanzare** i **numeri** e non **indietreggiare**.
Appena accenderete l'orologio **tutti** i **4 relè** o **triac** partiranno **eccitati**.
Se volete che all'accensione dell'orologio tutti i relè risultino **diseccitati**, dovrete andare alla riga **N.59** dove troverete questa istruzione:

```
      ldi   port_b,11110011b
```

e modificarla inserendo in sostituzione degli **1** degli **0** come qui sotto riportato:

```
      ldi   port_b,00000011b
```

Nota = Anche se questa riga è composta da **8 numeri**, dovrete modificare solo i primi **4** di sinistra.

Se volete far **eccitare** all'accensione il solo relè **RL4**, dovrete mettere un **1** in corrispondenza della prima cifra di sinistra come qui sotto riportato:

```
      ldi   port_b,10000011
```

A questo punto vi spieghiamo che cosa s'intende per **8 cicli** o **periodi** da utilizzare nell'arco delle **24 ore** che troverete riportati in queste righe:

1° periodo = righe 309 - 310 - 311
2° periodo = righe 316 - 317 - 318
3° periodo = righe 323 - 324 - 325
4° periodo = righe 330 - 331 - 332
5° periodo = righe 337 - 338 - 339
6° periodo = righe 344 - 345 - 346
7° periodo = righe 351 - 352 - 353
8° periodo = righe 358 - 359 - 360

Ogni ciclo è composto da **3 righe** d'istruzioni, quindi nel **1° ciclo** o **periodo** del nostro programma troverete:

```
      .byte   02         ;309 riga delle ore
      .byte   30         ;310 riga dei minuti
      .byte   11100000b  ;311 riga per comando relè
```

Attualmente il **1° ciclo** inizia alle ore **2,30 di notte**.
Per modificare l'orario basterà mettere nella **prima** riga l'**ora** che vi interessa, ad esempio **05-06-10**, e nella **seconda** riga i relativi **minuti**, ad esempio **00 - 10 - 30 - 50**.
Nella **terza** riga sono riportati i relè che desiderate **eccitare** e quelli che **non** desiderate eccitare all'orario da noi prestabilito.
Mettendo un **1** il relè si **ecciterà**, mettendo uno **0** si **diseccciterà**.
Ciò che dovrete modificare in questa terza riga so-

no **solo** i primi **quattro numeri** di sinistra posti dopo la parola **byte.**
Tenete presente che il **primo** numero di sinistra **piloterà** il relè **RL4** e l'**ultimo** numero di destra il relè **RL1**, quindi avrete in ordine:

RL4-RL3-RL2-RL1

Per farvi capire come modificare tutti questi numeri vi faremo un semplice esempio.
Ammesso che alle **06,10** desideriate **eccitare** i relè **RL2-RL1**, scriverete nelle righe **309 - 310 - 311** (1° ciclo) questi numeri:

```
.byte   06         ;309 riga delle ore
.byte   10         ;310 riga dei minuti
.byte   00110000b  ;311 riga per comando relè
```

Se alle **09,30** vorrete **diseccitare** ed **eccitare** il solo relè **RL4**, dovrete scrivere nelle righe **316 - 317 - 318** (2° ciclo) questi numeri:

```
.byte   09         ;316 riga delle ore
.byte   30         ;317 riga dei minuti
.byte   10000000b  ;318 riga per comando relè
```

Se alle **12,00** vorrete **diseccitare** anche il relè **RL4**, dovrete scrivere nelle righe **323 -324 -325** (3° ciclo) questi numeri:

```
.byte   12         ;323 riga delle ore
.byte   00         ;324 riga dei minuti
.byte   00000000b  ;325 riga per comando relè
```

Se alle **18,45** vorrete **eccitare** tutti i relè, dovrete scrivere nelle righe **330 - 331 - 332** (4° ciclo) questi numeri:

```
.byte   18         ;330 riga delle ore
.byte   45         ;331 riga dei minuti
.byte   11110000b  ;332 riga per comando relè
```

Se alle **22,30** vorrete **diseccitare** i relè **RL4-RL3**, dovrete scrivere nelle righe **337 - 338- 339** (5° ciclo) questi numeri:

```
.byte   22         ;337 riga delle ore
.byte   30         ;338 riga dei minuti
.byte   00110000b  ;339 riga per comando relè
```

Se alle **23,40** vorrete lasciare **eccitato** il solo relè **RL1**, dovrete scrivere nelle righe **344 - 345 - 346** (6° ciclo) questi numeri:

```
.byte   23         ;344 riga delle ore
.byte   40         ;345 riga dei minuti
.byte   00010000b  ;346 riga per comando relè
```

Se alle **24,00** vorrete **diseccitare** anche questo relè, dovrete scrivere nelle righe **351 - 352 - 353** (7° ciclo) questi numeri:

```
.byte   00         ;351 riga delle ore
.byte   00         ;352 riga dei minuti
.byte   00000000b  ;353 riga per comando relè
```

Avendo utilizzato solo **7 cicli** degli **8** disponibili, se l'ultimo non vi interessa lo **potrete cancellare** oppure inibire, mettendo davanti alle righe **358 - 359 - 360** un **punto** e **virgola** o mettendo sulla terza riga **000000b.**
Potrete **aggiungere** altri cicli se **8** risultassero insufficienti.

Facciamo presente che questi **cicli** si **ripeteranno** automaticamente all'**infinito** agli stessi **orari** tutti i giorni.

TIME90.HEX

Questo programma è un **timer** che, contando in **avanti**, ecciterà un relè o un triac quando raggiungerà i **minuti** e i **secondi** da noi prefissati.
Per farlo funzionare occorre inserire nel **bus LX.1202** la scheda dei display siglata **LX.1204** e quella dei relè siglata **LX.1205**, oppure quella dei triac siglata **LX.1206**.
Non appena alimenterete il circuito, il conteggio partirà da **00:00** e inizierà a contare in **avanti**; a questo punto potrete utilizzare i pulsanti **P1** e **P2** presenti sulla scheda display **LX.1204**.

Premendo **P1** il conteggio si **ferma**.
Premendo nuovamente **P1** il conteggio riparte dal **numero** sul quale si era fermato.
Premendo **P2** il contatore si **resetta**.
Premendo **P1** il contatore riparte da **00:00**.

Nota = Il pulsante **P2** di **reset** sarà attivo solamente se avrete **fermato** il conteggio con **P1.** Se premerete **P2** mentre è **attivo** il conteggio, questo non si azzererà.
I pulsanti presenti sulle schede Triac e Relè non risultano **attivati**.
Il conteggio del display arriva ad un massimo di **89 minuti** e **59 secondi**.

Il programma **TIME90.ASM**, come potrete notare, dispone di **4 cicli** perchè **quattro** sono i **relè** e i **triac** presenti sulle schede sperimentali.

1° ciclo = Dopo **20 secondi** dall'accensione si ecciterà il solo relè **RL1**.

Ovviamente sui display vedrete apparire **19** e, quando questo numero raggiungerà **00:00**, il relè si **ecciterà**.

2° ciclo = Passando al **secondo ciclo**, questo relè rimarrà **eccitato** per un tempo da noi prefissato in **1 minuto** e **30 secondi** e raggiunto questo **tempo** il relè **RL1** si **disecciterà** e automaticamente si **ecciterà** il relè **RL2**.
Il relè **RL2** si ecciterà un secondo dopo che sui display sarà apparso il numero **01:29** che cambierà in **00:00**.

3° ciclo = Dopo **47 secondi**, cioè quando sul display il numero **46** passerà sullo **00**, il relè **RL2** si **disecciterà** e si **ecciterà** il terzo relè **RL3**.

4° ciclo = Il conteggio continuerà ed allo scoccare dei **3 minuti** e **00 secondi** (tempo da noi prefissato) si **disecciterà** il relè **RL3** e si **ecciterà** il relè **RL4**, cioè si ritornerà al **1° ciclo** per ripetere all'infinito i **quattro cicli**.

Per **variare** i **tempi** che noi abbiamo prefissato dovrete variare queste righe:

1° ciclo = righe 289 - 290
2° ciclo = righe 295 - 296
3° ciclo = righe 301 - 302
4° ciclo = righe 307 - 308

Se volete che il **1° ciclo** abbia una durata di **1 minuto** e **30 secondi**, dovrete inserire nelle sue righe questi numeri:

```
ldi   stsex,30    ;289 secondi per RL1
ldi   stmix,1     ;290 minuti per RL1
```

Se volete che il **2° ciclo** abbia una durata di **50 secondi**, dovrete inserire nelle sue righe questi numeri:

```
ldi   stsex,50    ;295 secondi per RL1
ldi   stmix,00    ;296 minuti per RL1
```

Se volete che il **3° ciclo** abbia una durata di **15 minuti** e **20 secondi**, dovrete inserire nelle sue righe questi numeri:

```
ldi   stsex,20    ;301 secondi per RL1
ldi   stmix,15    ;302 minuti per RL1
```

Se volete che il **4° ciclo** abbia una durata di **2 minuti** e **10 secondi**, dovrete inserire nelle sue righe questi numeri:

```
ldi   stsex,10    ;307 secondi per RL1
ldi   stmix,2     ;308 minuti per RL1
```

Nelle righe **292/293 - 298/299 - 304/305 - 310/311** sono riportate le sigle dei **relè** che volete **eccitare** e di quelli che volete rimangano **diseccitati**.
Guardando l'esempio riportato nel programma **CLOCK.ASM** saprete già che scrivendo questa istruzione:

```
ldi   port_b,11110011b
```

potrete eccitare ad ogni **ciclo** anche **più relè** a vostra scelta.
Nei primi **quattro** numeri di sinistra (vedi **1111**) dovrete mettere un **1** sul relè che volete far **eccitare** ed uno **0** se **non** lo volete eccitare.

TEMP90.HEX

Questo programma è un **timer** che fa esattamente l'**inverso** del programma **TIME90**, cioè **conta all'indietro** e quando raggiunge lo **00:00** eccita i relè.
I relè, come per il programma precedente, li ecciterete in **4 cicli** e come tempo **massimo** di partenza potrete impostare **90 minuti** e **00 secondi**.
Non appena alimenterete il circuito, il conteggio partirà da **00:20** (questo tempo lo abbiamo prescelto noi, ma poi vi spiegheremo come modificarlo) e procederà all'**indietro**.
Dopo che avrà avuto inizio il conteggio, potrete utilizzare i pulsanti **P1** e **P2** presenti sulla scheda display **LX.1204**.

Premendo **P1** il conteggio si **ferma**.
Premendo nuovamente **P1** il conteggio riparte dal **numero** sul quale si era fermato.
Premendo **P2** il contatore si **resetta**.
Premendo **P1** il contatore riparte dal tempo che avete impostato come **partenza** per il conteggio all'**indietro**.

Nota = Il pulsante **P2** di **reset** sarà attivo solamente se avrete **fermato** il conteggio con **P1**. Se premerete **P2** mentre è **attivo** il conteggio, questo non si azzererà. Premendo **P2** per **resettarlo**, è intuitivo che contando all'**indietro** sul display ritorni il tempo di **partenza**, cioè **00:20**.

Nei **4 cicli** impostati otterrete queste condizioni:

1° ciclo = All'accensione si ecciterà il solo relè **RL1** e sui display apparirà **00:20** e a questo punto avrà inizio il conteggio alla **rovescia** che si fermerà sullo **00:00**.

2° ciclo = Dopo un secondo si ecciterà il relè **RL2** e a questo punto inizierà il **secondo ciclo**, che farà apparire sui display **01:30** (tempo **1 minuto** e **30 secondi**) che, secondo per secondo, decrementerà fino ad arrivare a **00:00**.

3° ciclo = A questo punto si **ecciterà** il relè **RL3** e sui display apparirà **00:47** che decrementerà fino ad arrivare allo **00:00**.

4° ciclo = L'ultimo ciclo farà eccitare il relè **RL4** e farà apparire sui display il numero **03:00** (**3 minuti**). Quando con il conteggio alla **rovescia** si arriverà al numero **00:00**, questo relè si **disecciterà** e contemporaneamente si **disecciterà** il relè **RL1**, cioè si ritornerà al **1° ciclo** per ripetere all'infinito i **quattro cicli**.

Per **variare** i **tempi** prefissati dovrete variare queste righe:

1° ciclo = righe 290 - 291
2° ciclo = righe 296 - 297
3° ciclo = righe 302 - 303
4° ciclo = righe 308 - 309

Se volete che il **1° ciclo** abbia una durata di **1 minuto** e **30 secondi**, dovrete inserire nelle sue righe questi numeri:

```
ldi   stsex,30   ;290 secondi per RL1
ldi   stmix,1    ;291 minuti per RL1
```

Se volete che il **2° ciclo** abbia una durata di **50 secondi**, dovrete inserire nelle sue righe questi numeri:

```
ldi   stsex,50   ;296 secondi per RL1
ldi   stmix,00   ;297 minuti per RL1
```

Se volete che il **3° ciclo** abbia una durata di **15 minuti** e **20 secondi**, dovrete inserire nelle sue righe questi numeri:

```
ldi   stsex,20   ;302 secondi per RL1
ldi   stmix,15   ;303 minuti per RL1
```

Se volete che il **4° ciclo** abbia una durata di **2 minuti** e **10 secondi**, dovrete inserire nelle sue righe questi numeri:

```
ldi   stsex,10   ;308 secondi per RL1
ldi   stmix,2    ;309 minuti per RL1
```

Nelle righe **292/293 - 298/299 - 304/305 - 310/311** sono riportate le sigle dei **relè** che si **ecciteranno** e di quelli che si **disecciteranno**.
Anche in questo programma possiamo sostituire le righe sopra menzionate con questa sola riga d'istruzione:

```
ldi   port_b,11110011b
```

Nei primi **quattro** numeri di sinistra (vedi **1111**) dove metterete **1** il relè si **ecciterà**, dove metterete **0** si **disecciterà**.

NOTA

I programmi che vi abbiamo fornito e che in seguito vi forniremo sono a sfondo **didattico** e servono per far capire ai principianti come si debba scrivere un'istruzione per ottenere una specifica funzione e per questo abbiamo aggiunto, di fianco ad ogni riga di programma, un **commento**.
Blocchi di un programma si possono prelevare e trasferire su un altro programma, cosa che potrete fare quando avrete già acquisito una certa esperienza.

COSTO DI REALIZZAZIONE

Tutti i componenti necessari per la realizzazione di questa scheda, compresi circuito stampato, connettore, triac da 500 Volt 5 Amper, fotoaccopiatori, zoccoli, pulsanti, morsettiere, condensatori e resistenze (vedi fig.8) .....€ 18,60

Il solo circuito stampato **LX.1206** .....€ 4,91

Nota = Nel kit **non è incluso** il dischetto dei programmi **DF1202.3** perchè già fornito agli acquirenti del kit **LX.1202**.
A chi non avesse acquistato questo kit e volesse il solo dischetto dei programmi, possiamo inviarlo a € 6,20 più le spese postali.

Ai prezzi riportati, già comprensivi di IVA, andranno aggiunte le sole spese di spedizione a domicilio.

SCHEDA con DISPLAY

Dopo avervi presentato una scheda per eccitare dei relè, una per pilotare dei triac e una terza per pilotare dei normali display a 7 segmenti, oggi vogliamo proporvi una scheda per display numerici LCD dopodichè passeremo ai display alfanumerici.

Chissà quanti di voi essendo in possesso di un display **LCD** avranno tentato di far apparire un numero senza riuscirci, pur avendolo alimentato correttamente, collegato i vari **segmenti** e controllato più volte il montaggio per verificare di non aver commesso involontariamente qualche errore.
Il motivo per il quale non siete riusciti ad accenderlo ve lo spiegheremo in questo articolo, quindi ammesso che non vi interessi sapere come si programmi un microprocessore **ST6** per pilotare un display **LCD**, vi consigliamo ugualmente di leggerlo.

GESTIRE un DISPLAY a 7 SEGMENTI

Se volessimo pilotare **4 display** a **7 segmenti** con un integrato **driver**, questo dovrebbe avere come minimo **32 piedini** d'uscita, infatti oltre ai **7 segmenti** presenti su ogni display per un totale di **4 x 7 = 28** terminali, essendo presente su ognuno di questi il **punto decimale**, dovremo aggiungere altri **4** terminali e, in tal modo, otterremo un totale di **32 piedini**.
Come già saprete, per accendere uno o più **segmenti** basterà alimentarli con una tensione che potremo prelevare da una normale pila.
Per pilotare **4 display** esistono degli integrati provvisti di **34** piedini d'**uscita** necessari per accendere tutti i vari segmenti.
Per l'**ingresso** troviamo invece solo **3 piedini** perchè questi lavorano in modo **seriale**.
Questi **3** piedini d'**ingresso** sono chiamati:

Enable
Data
Clock

Nell'integrato **driver** tipo **M.5450** in grado di pilotare **4 display** a **7 segmenti**:

- il piedino **23** dell'**Enable** lo dovremo collegare a **massa**.
- il piedino **22** del **Data** serve per ricevere i **dati seriali** necessari per far accendere i vari segmenti.
- il piedino **21** del **Clock** serve per ricevere un treno di onde quadre per **sincronizzare** i segnali.
I due piedini supplementari d'uscita corrispondenti ai bit **33-34** potrebbero risultare utili per far lampeggiare ad una cadenza di **1 secondo** due diodi led interposti tra i due numeri delle **ore** e dei **minuti**, se realizzeremo un orologio.
I piedini d'uscita di questo integrato, come visibile in fig.1, vengono indicati **BIT1 - BIT2 - BIT3**, ecc.
Quando tutti i segmenti dei display sono **spenti**, nel piedino del **clock** entrerà un treno di onde quadre e nel piedino **data** il solo impulso di **Start** (bit **0**) e due impulsi, uno di **Load** (bit **35**) ed uno di **Reset** (bit **36**)(vedi fig.2).
A questo punto potremo assegnare ciascuno di questi **34 bit** ad un singolo **segmento** presente in ogni display come visibile in fig.2 e come riportato nella Tabella N.1).

Se in questi **4 display** volessimo accendere il numero **1 0 3 2** tramite il **programma software**, dovremmo portare a **livello logico 1** tutti i bit corrispondenti ai segmenti che desideriamo accendere.

Sul **display N.1**, dovendo accendere i segmenti **B-C**, dovremo portare a **livello logico 1** i **bit 2-3**.
Sul **display N.2**, dovendo accendere i segmenti **A-B-C-D-E-F**, dovremo portare a **livello logico 1** i **bit 9-10-11-12-13-14**.
Sul **display N.3**, dovendo accendere i segmenti **A-B-C-D-G**, dovremo portare a **livello logico 1** i **bit 17-18-19-20-23**.
Sul **display N.4**, dovendo accendere i segmenti **A-B-G-E-D**, dovremo portare a **livello logico 1** i **bit 25-26-31-29-28** (vedi fig.3).

LCD pilotata con un ST6

TABELLA N.1

BIT comando	segmento DISPLAY 1	piedino INTEGRATO
1	A	18
2	B	17
3	C	16
4	D	15
5	E	14
6	F	13
7	G	12
8	punto	11

BIT comando	segmento DISPLAY 2	piedino INTEGRATO
9	A	10
10	B	9
11	C	8
12	D	7
13	E	6
14	F	5
15	G	4
16	punto	3

BIT comando	segmento DISPLAY 3	piedino INTEGRATO
17	A	2
18	B	40
19	C	39
20	D	38
21	E	37
22	F	36
23	G	35
24	punto	34

BIT comando	segmento DISPLAY 4	piedino INTEGRATO
25	A	33
26	B	32
27	C	31
28	D	30
29	E	29
30	F	28
31	G	27
32	punto	26

BIT comando	diodi LED	piedino INTEGRATO
33	led 1	24
34	led 2	25

Per accendere un numero su un normale display a 7 segmenti è sufficiente portare a "livello logico 1" i bit che pilotano i segmenti interessati (vedi figg.2-3).

Come avrete intuito, basta portare a **livello logico 1** uno di questi **34 bit** per far apparire un **numero** oppure anche una **lettera**, ad esempio una **L**, una **C** o una **H**.
Tramite software, il **display 1** potremo farlo diventare il **4°** oppure il **3°** in modo da adattarlo al disegno del circuito stampato.
Un altro particolare **molto importante** da sottolineare è quello di non confondere il **livello logico** del **bit** con il livello logico che in pratica ci ritroveremo sul **piedino d'uscita** dell'integrato che, come potrete constatare, risulta **invertito**, quindi portando ad esempio i **bit 25-26** a **livello logico 1**, sui corrispondenti piedini **33-32** dell'integrato **M.5450** (vedi Tabella N.1) ci ritroveremo un **livello logico 0**.
Quindi se qualcuno andasse a controllare con un tester la tensione presente sui piedini d'uscita che **accendono** un segmento, troverebbe **0 volt** e non una tensione **positiva**.

GESTIRE un DISPLAY LCD

Come già spiegato, se volessimo pilotare un **display a led** con **4 cifre** con un integrato **driver**, questo dovrebbe avere **34 piedini** d'uscita.
Per quanto riguarda i display **LCD** dobbiamo far presente che il **punto decimale** che dovrebbe risultare presente sul **4° display** spesso viene sostituito da **due punti** posti tra le prime due e le ultime due cifre.

Questi **due punti** vengono spesso utilizzati negli orologi per dividere le **ore** dai **minuti**.
A differenza del **drive M.5450**, utilizzato per pilotare i display a **7 segmenti**, che dispone di tre **ingressi** chiamati **Enable-Data-Clock**, il **drive M.8438/AB6**, utilizzato per i display **LCD**, dispone di tre **ingressi** chiamati:

Strobe
Data
Clock

- Il piedino **2** dello **Strobe** si porta a **livello logico 1** ogni volta che deve inviare al display il treno dei **dati**.
- Il piedino **34** del **Data** serve per ricevere i **dati seriali** necessari per far accendere i vari segmenti.
- Il piedino **40** del **Clock** invia un treno di onde quadre per **sincronizzare** i segnali (vedi fig.5).

Come potrete notare, il **drive** per un display **LCD** ha l'ingresso **Strobe** che mancava nel **drive** per display a **7 segmenti a led** ed ha in più un piedino chiamato **Back/Plane** (piedino **30**) che viene utilizzato per alimentare il display con un'onda quadra di circa **80 Hertz**.
Quindi i quattro display **LCD** non vengono alimentati da una **tensione continua**, ma da una tensione che cambia in continuità il suo **livello logico** da **1** a **0** e questo, come vedremo, è indispensabile per poter **accendere** i vari segmenti.

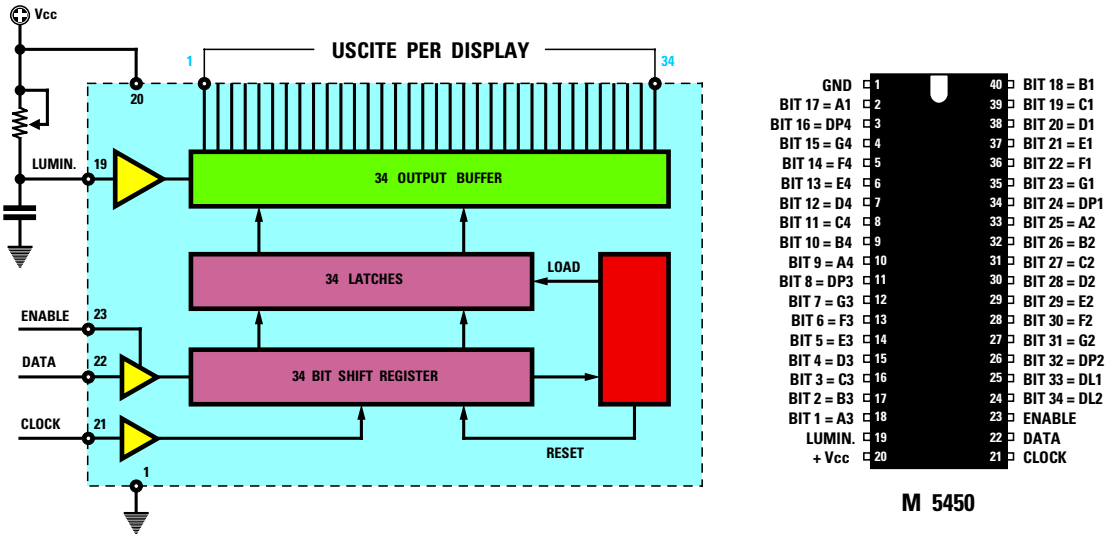


Fig.1 Per pilotare 4 display a 7 segmenti occorre un integrato drive tipo M.5450 o altri equivalenti. Dei tre piedini d'ingresso, quello indicato Enable viene collegato a massa, e quelli indicati Data e Clock vengono utilizzati per ricevere i dati seriali e i segnali di sincronismo.

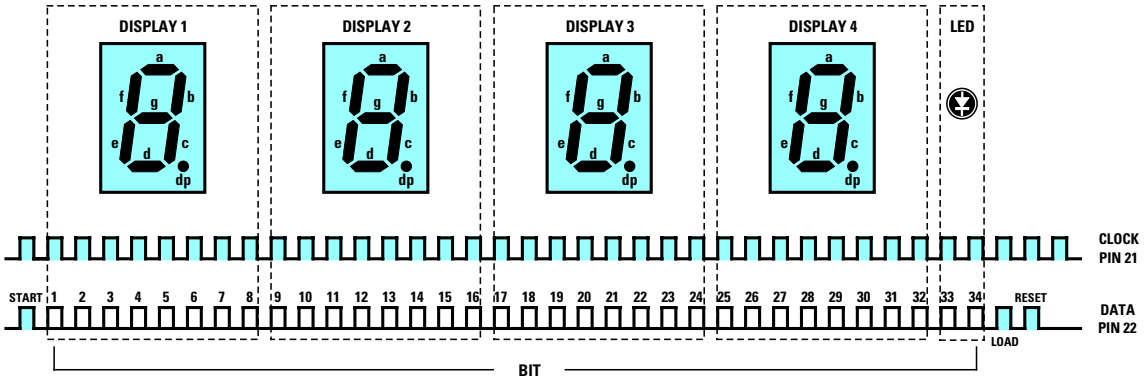


Fig.2 Poiché per ogni display sono necessari 8 bit (uno serve per il punto decimale - vedi Tabella N.1), l'integrato drive M.5450 convertirà i dati seriali applicati sugli ingressi in dati paralleli a 34 bit. Oltre a questi 34 bit l'integrato invia in uscita un bit di Start, uno di Load (carica dati) ed uno di Reset.

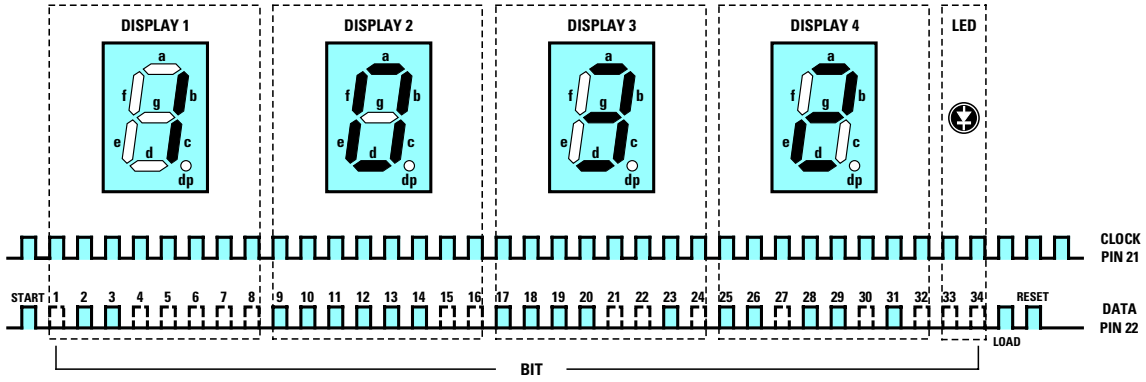


Fig.3 Per accendere il numero "1032", sul primo display dovremo portare a livello logico 1 i bit 2-3, sul secondo display i bit 9-10-11-12-13-14, sul terzo display i bit 17-18-19-20-23 e sull'ultimo display i bit 25-26-28-29-31. Non confondete il livello logico del "bit" con quello che apparirà sul piedino d'uscita che risulta invertito.

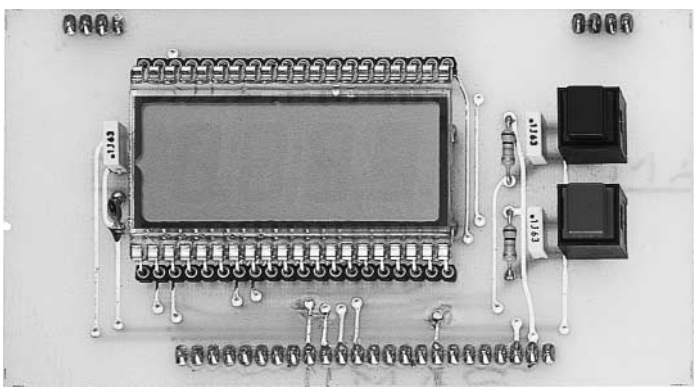


Fig.4 Per pilotare un display LCD dovremo sempre portare a "livello logico 1" i bit dei segmenti che vogliamo accendere, ma come potete vedere in fig.7 i segmenti che rimangono spenti non vengono portati a "livello logico 0".

I piedini d'uscita di questo integrato, come visibile in fig.5, indicati **A1 - B1 - C1**, ecc., andranno collegati ad ogni singolo **segmento** presente in ciascun display (vedi fig.6 e Tabella N.2).

TABELLA N.2

BIT comando	segmento DISPLAY 1	piedino INTEGRATO
1	A	10
2	B	9
3	C	8
4	D	7
5	E	6
6	F	5
7	G	4
8	punto	11

BIT comando	segmento DISPLAY 2	piedino INTEGRATO
9	A	18
10	B	17
11	C	16
12	D	15
13	E	14
14	F	13
15	G	12
16	punto	19

BIT comando	segmento DISPLAY 3	piedino INTEGRATO
17	A	26
18	B	25
19	C	24
20	D	23
21	E	22
22	F	21
23	G	20
24	punto	27

BIT comando	segmento DISPLAY 4	piedino INTEGRATO
25	A	39
26	B	38
27	C	37
28	D	33
29	E	32
30	F	29
31	G	28
32	punto	3

Anche in un **display LCD** se volessimo accendere i numeri **1 0 3 2** tramite il **programma software**, dovremmo portare a **livello logico 1** tutti i segmenti che desideriamo accendere.

Sul **display N.1**, dovendo accendere i segmenti **B-C**, dovremo portare a **livello logico 1** i **bit 2-3**.
Sul **display N.2**, dovendo accendere i segmenti **A-B-C-D-E-F**, dovremo portare a **livello logico 1** i **bit 9-10-11-12-13-14**.
Sul **display N.3**, dovendo accendere i segmenti **A-B-C-D-G**, dovremo portare a **livello logico 1** i **bit 17-18-19-20-23**.
Sul **display N.4**, dovendo accendere i segmenti **A-B-G-E-D**, dovremo portare a **livello logico 1** i **bit 25-26-31-29-28**.

A questo punto si potrebbe pensare che un **drive** per pilotare un display a **7 segmenti** possa servire anche per pilotare un display **LCD**, ma purtroppo questo non è possibile perchè i suoi **segmenti** non vengono pilotati da una **tensione continua**, ma da un treno di **onde quadre** che lo stesso integrato **M.8438/AB6** invia al segmento da **accendere**, sfasandolo di **180°** rispetto alle onde quadre che giungono sui segmenti che debbono rimanere **spenti**.

Quindi se controllassimo con un oscilloscopio tutti i **32 piedini** d'uscita di questo integrato, troveremmo in **tutti** delle onde quadre, perciò potrebbe risultare alquanto complesso capire come si riescano ad accendere questi **segmenti**.

Per spiegarvelo in modo che possiate comprenderlo perfettamente vi proponiamo questo semplice esempio.
Ammesso di voler accendere in un display i segmenti **B-C** in modo da far apparire il numero **1**, in questi due **soli** segmenti giungeranno delle onde quadre **invertite** rispetto alle onde quadre che giungono invece sul **Back/Plane** (vedi fig.7).
Di questa inversione del segnale ad onda quadra non dovremo preoccuparci, perchè sarà l'integrato **M.8438/AB6** che provvederà ad **invertirlo** quando, tramite software, gli diremo di accendere i segmenti **B-C**.

Il software per i display **LCD** in pratica è molto similare a quello per i display a **7 segmenti**, sempre che si rispettino le connessioni d'uscita dell'integrato **M.8438/AB6** con i piedini che fanno capo ai segmenti dei quattro **display** (vedi Tabelle **N.1** e **N.2**).

Nelle connessioni del display tipo **S.5126** oppure **LC.513040** riportate in fig.8, abbiamo indicato con **A1-B1-C1**, ecc., tutti i segmenti del display **N.1**, con **A2-B2-C2**, ecc., tutti i segmenti del display **N.2** e così dicasi per il display **N.3** e **N.4**.
Se abbiamo un display con una sigla diversa, do-

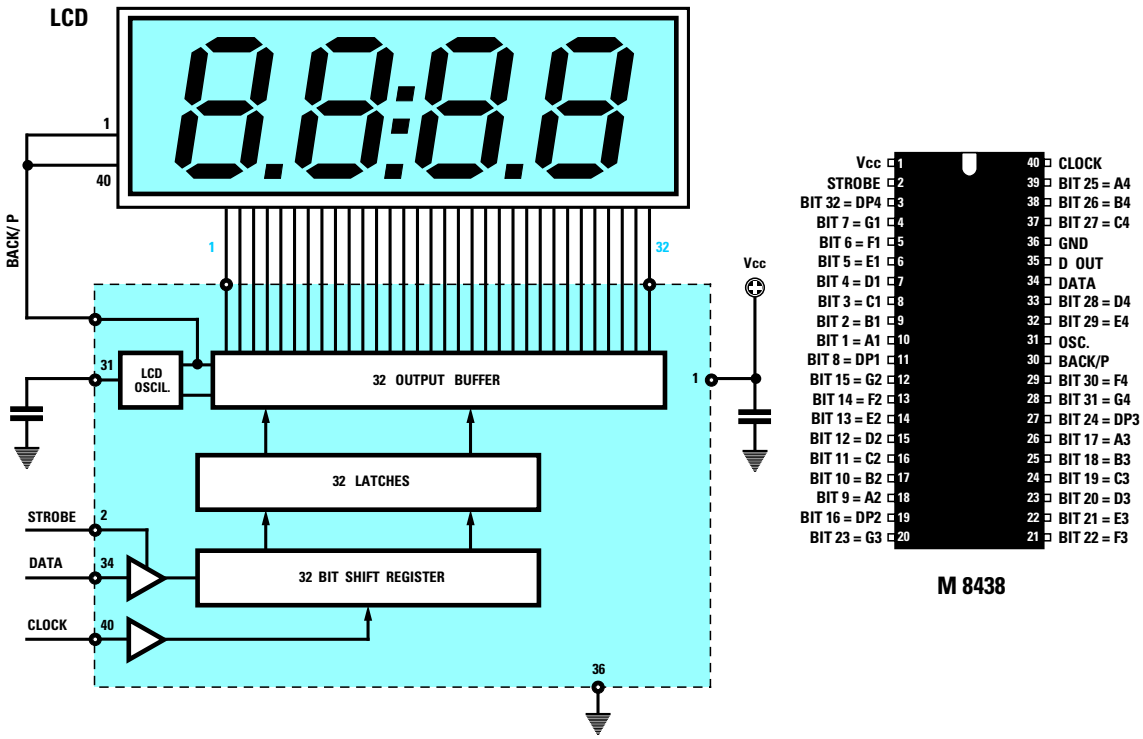


Fig.5 Per pilotare 4 display LCD occorre un integrato drive tipo M.8438 o altri equivalenti. Dei tre piedini d'ingresso, quello indicato Strobe si porta a livello logico 1 quando deve caricare i dati, quelli indicati Data e Clock vengono utilizzati per ricevere i dati seriali e i segnali di sincronismo. Il display LCD viene alimentato direttamente dall'integrato M.8438 tramite il piedino Back/Plane.

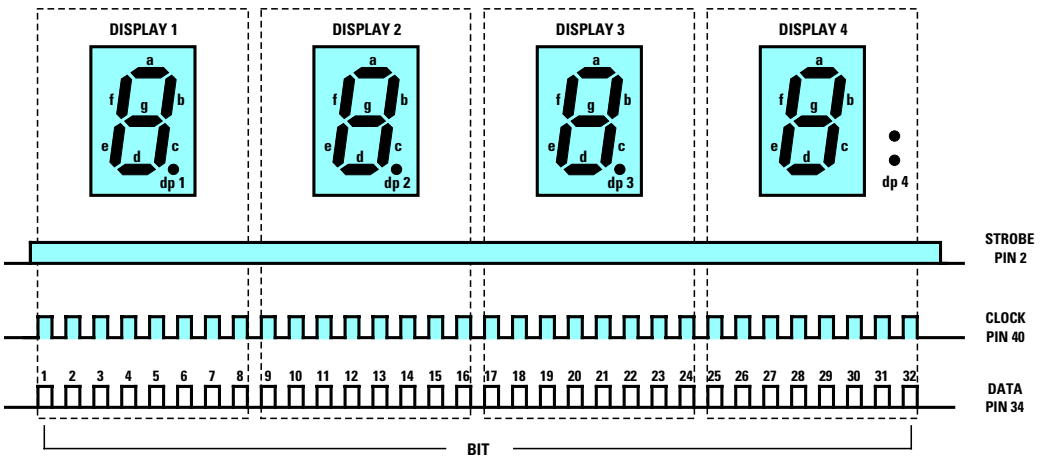


Fig.6 Ogni bit corrisponde ad un singolo segmento di ogni display (vedi Tabella N.2) e per agevolarvi abbiamo riportato su ogni piedino dell'integrato M.8438 (vedi fig.5) sia il numero del bit sia quale segmento si accenderà sui display 1-2-3-4.

vremo controllare a quali piedini fanno capo i segmenti **A-B-C-D-E-F-G** del primo, del secondo, del terzo e del quarto display, per collegarli in ordine ai piedini d'uscita dell'integrato **drive**.

SCHEMA ELETTRICO

Dopo avervi spiegato la differenza che esiste tra un display a **7 segmenti a led** ed uno a **LCD** possiamo passare a presentarvi lo schema elettrico.
Guardando la fig.9 è possibile notare che dal connettore siglato **CONN. 1/2** preleviamo:

- dal piedino **B0** i **dati** da inviare al piedino **34** dell'integrato **M.8438/AB6.**
- dal piedino **B1** il segnale del **Clock** da inviare al piedino **40** di **IC1**.
- dal piedino **C4** il segnale di **Strobe** da inviare al piedino **2** di **IC1**.
- dal piedino **+5V** la tensione di alimentazione per l'integrato.

Gli altri due piedini **B2-B3** sono collegati ai pulsanti **P1-P2** che ci servono, realizzando un orologio, per mettere a punto le ore e i minuti.
Poichè l'integrato **M.8438/AB6** ha bisogno del segnale di **Strobe**, dovremo necessariamente utilizzare dei microprocessori **ST6** con **28** piedini, cioè:

ST62E25 (da 4K **cancellabile**)
ST62T25 (da 4K **non cancellabile**)
ST62T15 (da 2K **non cancellabile**)

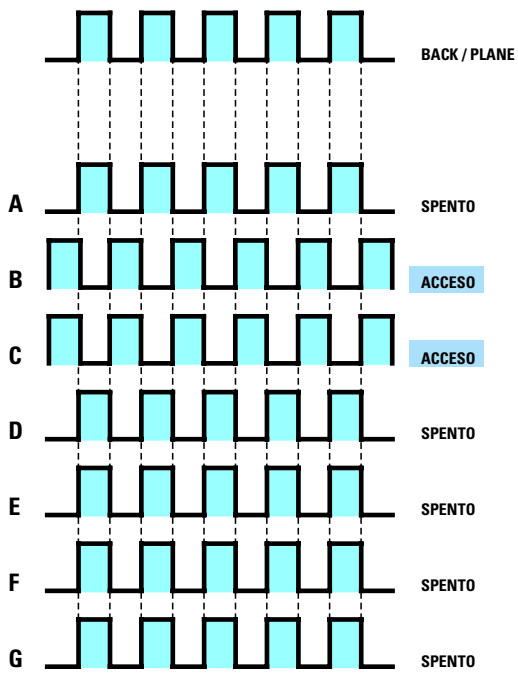


Fig.7 Per accendere un segmento in un display LCD l'integrato IC1 invierà sul piedino corrispondente un'onda quadra sfasata di 180° rispetto a quella del Back/Plane. In questo esempio risultano accesi i soli segmenti B-C.

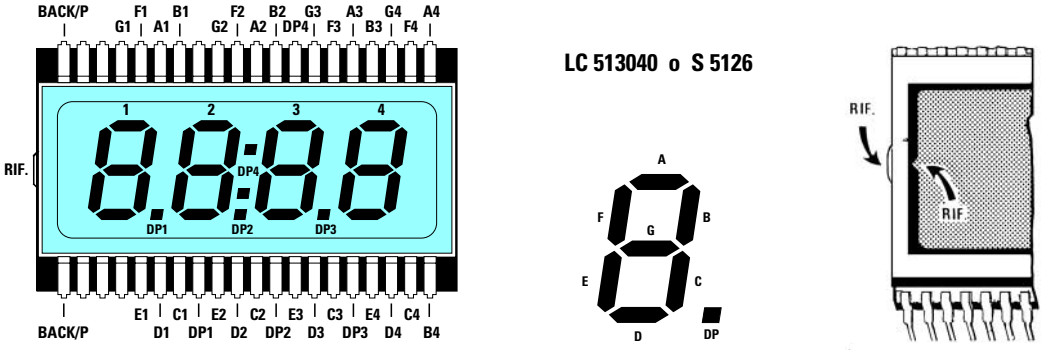


Fig.8 Connessioni del display LCD. Su ogni terminale abbiamo riportato la lettera dei sette segmenti A-B-C ecc. seguita dai numeri del display cioè 1-2-3-4. La tacca di riferimento è costituita da una "goccia" di vetro o dal segno ">" posto da un lato del corpo.

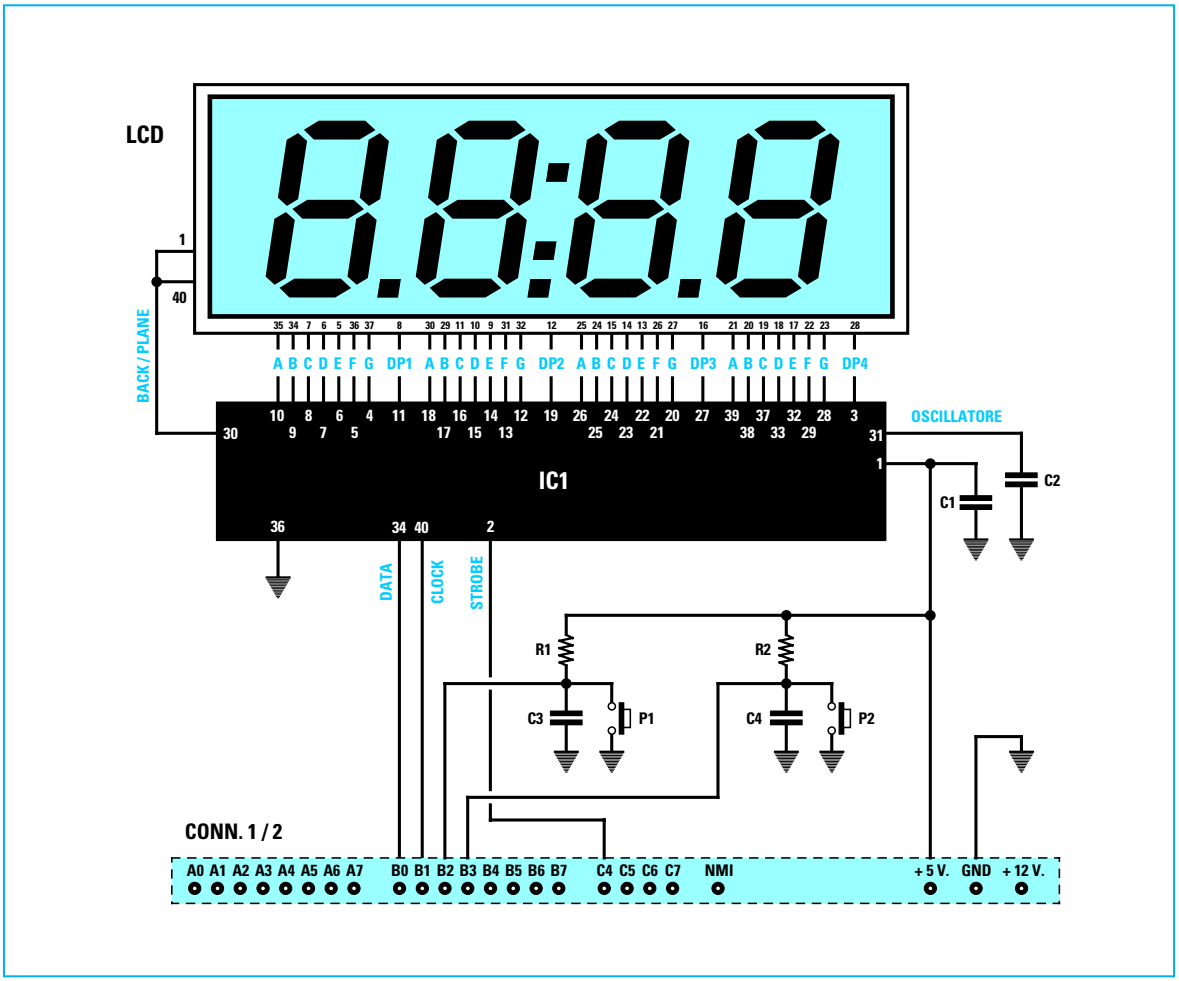


Fig.9 Schema elettrico da noi utilizzato per pilotare un display LCD.

ELENCO COMPONENTI LX.1207

R1 = 10.000 ohm 1/4 watt
R2 = 10.000 ohm 1/4 watt
C1 = 100.000 pF poliestere
C2 = 22 pF ceramico
C3 = 100.000 pF poliestere
C4 = 100.000 pF poliestere
IC1 = M.8438-AB6 o M.8438-B6
LCD = display LCD tipo S.5126
CONN.1/2 = connettore 24 poli
CONN. = 2 connettori 4 poli
P1 = pulsante
P2 = pulsante

Il motivo per il quale si è obbligati ad utilizzare dei micro **ST6** a **28 piedini** è dovuto al fatto che ci serve la porta **C4** per il segnale di **Strobe**.

A questo punto qualcuno ci potrà far osservare che il segnale di **Strobe** potevamo benissimo prelevarlo da una bit libero della porta **A** oppure **B** e, in tal modo, potevamo ancora utilizzare un micro **ST6** con soli **20** piedini.

Purtroppo nel **bus** siglato **LX.1202** (vedi rivista **N.179**) se oltre alla scheda di questo display volessimo inserire anche la scheda **LX.1205** che pilota dei **relè**, oppure la scheda **LX.1206** che pilota dei **triac**, constateremmo che tutti i **bit** delle porte **A-B** risultano occupati.

Poichè in questa scheda utilizziamo anche la **porta C**, il programma che vi forniremo per questo **display LCD** sarà completamente diverso rispetto al software che vi avevamo già fornito per la scheda display **LX.1204** e questo potrà risultarvi molto utile perchè, confrontando i due programmi, potrete notare le differenze.

Comunque questo programma per **display LCD** svolgerà le stesse funzioni che svolge tutt'ora il programma per il display a **7 segmenti**.

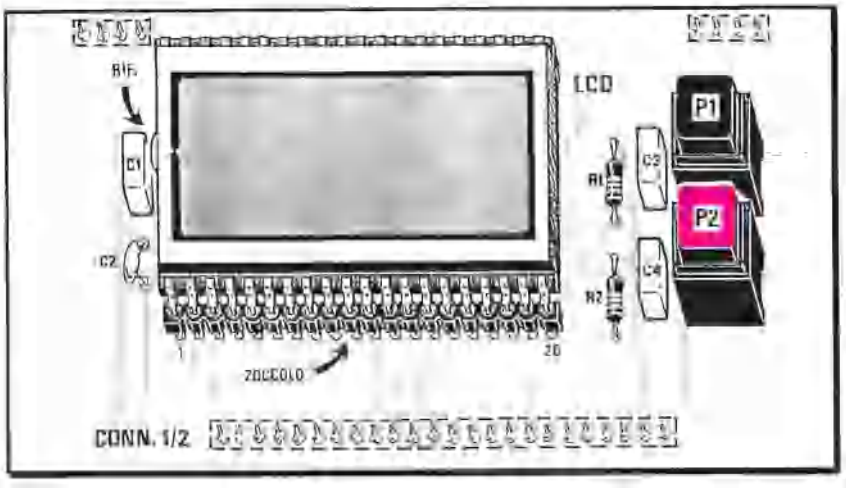


Fig.10 Scheda vista dal lato del display.

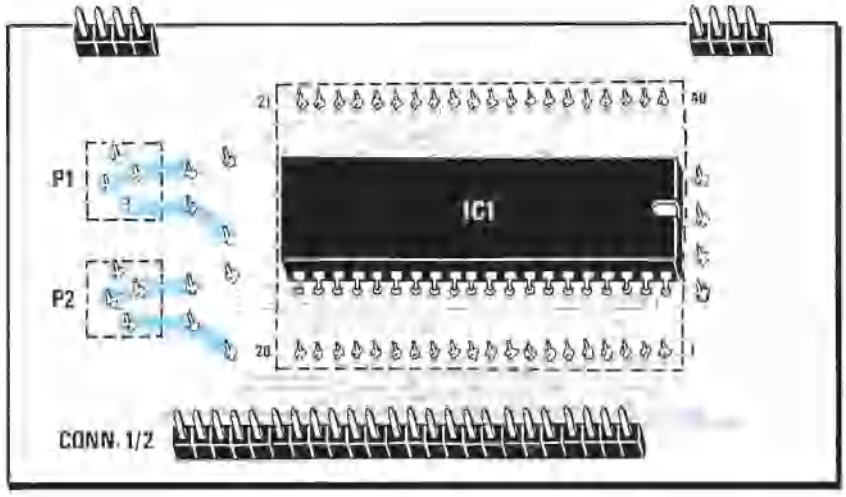


Fig.11 Circuito stampato visto dal lato dell'integrato.

Ritornando al nostro schema elettrico, non dovrete preoccuparvi delle connessioni dell'integrato **IC1** con il **display LCD**, perchè queste vengono automaticamente effettuate tramite le **piste** in rame presenti sul circuito stampato.
In questo schema di critico c'è il solo condensatore **C2** collegato tra il piedino **31** e la **massa**.
Questo condensatore, come potrete leggere nell'elenco componenti, deve risultare da **22 picoFarad** e tale valore serve per generare, tramite uno stadio oscillatore interno, una frequenza di circa **20.500 Hz** che, divisa internamente per **256**, farà uscire dal piedino **30** di **IC1** un'onda quadra di circa **80 Hz**.
Questa frequenza chiamata **Back/Plane**, è quella che ci servirà per alimentare i piedini **1-40** del display **LCD**.

REALIZZAZIONE PRATICA

Realizzare questa scheda per display **LCD** che abbiamo siglato **LX.1207** è molto semplice.
Nel circuito stampato a doppia faccia con fori metallizzati dovrete inserire tutti i componenti richiesti e per iniziare vi consigliamo di saldare lo zoccolo per l'integrato **IC1**.
Dopo aver saldato tutti i piedini sulle piste dello stampato, potrete inserire i due connettori maschi da **4 piedini** e da **24 piedini**, che vi serviranno per innestare questa scheda nel **bus** siglato **LX.1202** (vedi rivista **N.179**).
Terminata questa operazione, capovolgete lo stampato e da questo lato inserite i due connettori femmina da **20** piedini che vi serviranno come zoccolo per il display **LCD**.

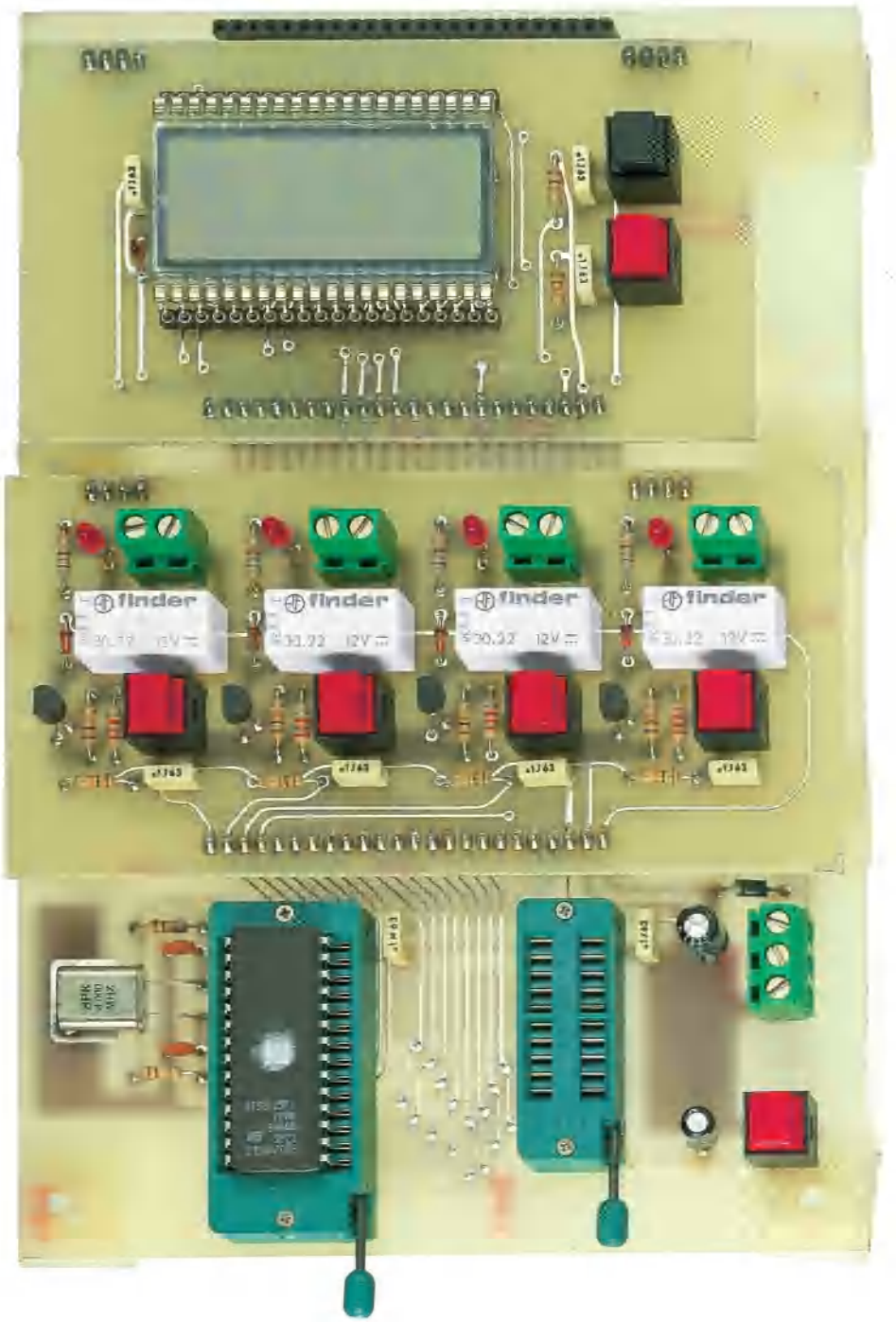


Fig.12 La scheda del display LCD siglata LX.1207 andrà inserita nel Bus LX.1202 congiunta anche alla scheda dei Relè o dei Triac. Come microprocessore ST6 dovrete necessariamente utilizzare quelli a 28 piedini, cioè l'ST62.E25 che è riprogrammabile o i tipi ST62.T15 e ST62.T25 che non sono riprogrammabili.

Per completare il montaggio dovrete inserire le resistenze **R1-R2**, i due pulsanti **P1-P2**, i tre condensatori poliestere ed il ceramico da **22 picoFarad** ed innestare nello zoccolo l'integrato **M.8438/AB6**, orientando la tacca a forma di **U** impressa sul suo corpo come visibile nella fig.10.
Dal lato opposto dovrete inserire nei due connettori femmina tutti i piedini presenti nel display **LCD** e qui forse incontrerete qualche difficoltà, perchè spesso i piedini del display risultano troppo divaricati.
Per poterli restringere in modo uniforme potrete premere sul piano di un tavolo tutti i terminali.
Prima di inserire il display nei connettori, dovrete ricercare sul suo corpo la **tacca di riferimento**, perchè se lo inserirete in senso inverso non potrà funzionare.
In questi display questa tacca di riferimento non è molto visibile, perchè quasi sempre è costituita da una piccola **goccia** di vetro posta su una sola estremità.
Da questo stesso lato troverete spesso sulla cornice nera che contorna l'interno del display il segno **>** (vedi fig.8).
Questa tacca di riferimento va sempre rivolta verso i condensatori **C1-C2**.
Spingete il display in corrispondenza dei lati in cui sono presenti i terminali e mai del centro perchè potrebbe spezzarsi.

PROGRAMMI

Per far funzionare questa scheda display **LCD** abbiamo preparato **5 programmi** che troverete inseriti in un dischetto floppy da **3** pollici siglato **DF1207.3**.
Una volta in possesso del dischetto, per caricarlo nell'Hard-Disk dovrete procedere come di seguito spiegato.

Uscite da qualsiasi programma che stavate utilizzando, **Windows - Pcshell - Norton** ecc., e quando sul monitor del computer appare **C:\>** inserite il dischetto nel drive **A**, quindi digitate:

C:\>A poi premete Enter

Quando appare **A:\>** scrivete:

A:\>installa e premete Enter

Il programma vi chiederà subito in quale **directory** volete installare il contenuto del disco.
Noi abbiamo già definito una **directory** che abbiamo chiamato **LX1207**, quindi se premete Enter il

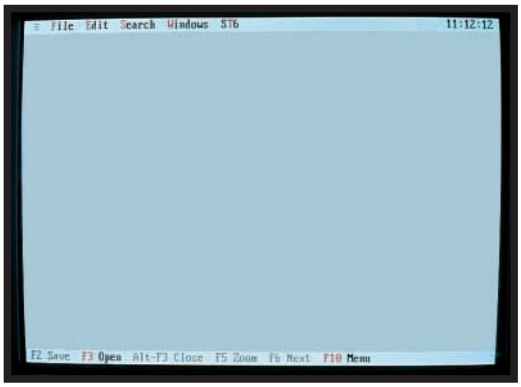


Fig.13 Quando sul monitor appare la finestra dell'Editor potete premere i tasti ALT+F poi F3 e vedrete apparire tutti i files con l'estensione .ASM.

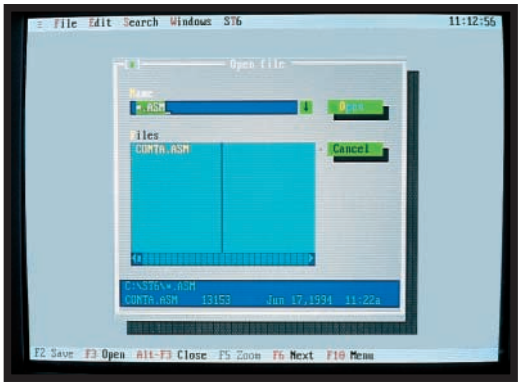


Fig.14 Nel nuovo dischetto LX1207 non troverete nessuno dei vecchi programmi (vedi foto), ma i soli programmi da utilizzare per questa scheda con display LCD.

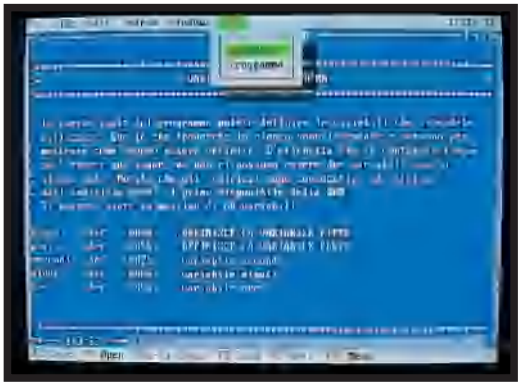


Fig.15 Potrete trasferire il programma scelto nella memoria di un ST6 solo dopo averlo assemblato. Per assemblarlo dovrete premere i tasti ALT+T poi Enter.

programma creerà automaticamente una directory con questo nome e, **scompattandolo**, copierà il contenuto del dischetto all'interno dell'Hard-Disk.
Se sul monitor dovesse apparire la scritta **error**, potrete ricaricare il dischetto con questo diverso sistema.

Quando appare **C:\>** dovrete creare la directory scrivendo:

C:\>MD LX1207 poi premete Enter

Quando riapparirà il prompt di **C:\>** inserite il dischetto floppy nel drive **A** poi scrivete:

C:\>COPY A:*.* C:\LX1207 poi premete Enter

Nota = Poiché nella digitazione è necessario rispettare la spaziatura, per agevolarvi abbiamo interposto una **barra** in colore che corrisponde allo **spazio** che occorre lasciare tra le lettere.

Quando il computer avrà terminato di copiare dal dischetto tutti i programmi, questi non saranno ancora stati **scompattati** quindi dovrete farlo voi scrivendo:

CD:\>CD LX1207 poi premete Enter

Quando appare **C:\LX1207>** digitate:

C:\LX1207>installa poi premete Enter

In questo modo vedrete via via comparire sul monitor i **nomi** dei files che si stanno **scompattando**.

CONVERTIRE i files .ASM in .HEX

Prima di trasferire un file nella memoria di un **ST6** occorre, come abbiamo precisato nei precedenti numeri, **assemblarlo** in modo da ottenere un secondo ed identico file, ma con l'estensione **.HEX**.
Se tenterete di trasferire un file **.ASM** nella memoria del micro vi verrà segnalato **error**.
Il micro da usare per i **display LCD** deve necessariamente avere **28 piedini**, quindi potrete adoperare un **ST62.E25** se volete riprogrammarlo e cancellarlo più volte, oppure un **ST62.T15** o un **ST62.T25** che come sapete **non sono** cancellabili.
Anche se nei precedenti articoli vi abbiamo spiegato come procedere per trasformare un file **.ASM** in un file **.HEX**, lo ripeteremo nuovamente.

Quando sul monitor appare **C:\LX1207>** dovete scrivere:

C:\LX1207>ST6 poi premere Enter

Con questo comando apparirà la finestra dell'**Editor** (vedi fig.13).
A questo punto dovrete premere prima i tasti **ALT+F** poi il tasto **F3** e sul monitor vedrete apparire tutti i files con estensione **.ASM** (vedi fig.14).

LCDCLOCK.ASM
LCDCRONO.ASM
LCDOROLO.ASM
LCDTEM90.ASM
LCDTIM90.ASM

Per posizionare il cursore su uno dei cinque files premete **ALT+F** poi premete i tasti freccia giù/su e quando sarete sul file che vi interessa premete **Enter**.
Entrerete così nell'editor del file selezionato e avrete in linea le istruzioni del programma.
Per **assemblare** il programma dovete premere i tasti **ALT+T** e poi Enter (vedi fig.15).
Ammesso che abbiate scelto il file **LCDCLOCK**, dopo pochi secondi apparirà sul monitor la scritta:

***** success *****

con il tempo di compilazione e di seguito la scritta:

C:\LX1207>

Premendo Enter rientrerete nel programma **LCDCLOCK**.
Per uscire dovrete premere **ALT+F3**.
Vedrete nuovamente la maschera dell'**Editor** e a questo punto premendo i tasti:

ALT+F poi **F3** poi Enter

apparirà nuovamente la maschera di tutti i files con estensione **.ASM**.
Nella riga in alto dovrete sostituire la scritta **.ASM** con la scritta **.HEX** poi premere Enter.
In questo modo apparirà l'elenco dei files convertiti in **.HEX**, e poiché è stato convertito il solo file **LCDCLOCK** comparirà:

LCDCLOCK.HEX

Vi ricordiamo che per modificare le righe di un programma dovrete sempre lavorare nell'estensione **.ASM**. Dopo aver fatto le modifiche le dovrete **salvare** premendo il tasto **F2**, poi dovrete **assemblare** il programma per convertirlo in un file **.HEX** come poc'anzi vi abbiamo spiegato.

I **5 programmi** che abbiamo inserito in questo dischetto hanno le stesse funzioni dei programmi che vi abbiamo presentato per i display a **7 segmenti**, ma sono stati riscritti e adattati per pilotare in **seriale** l'integrato **M.8438/B6**, quindi le righe che potrete modificare hanno un diverso numero.

LCDCRONO.HEX

Questo programma è un semplice **cronometro**, quindi per visualizzare i tempi occorre inserire nel **bus LX.1202** la **sola** scheda dei display siglata **LX.1207**.
Se nel bus inserirete le schede dei **relè** o dei **triac**, non potrete renderle attive perchè nel programma non è presente nessuna istruzione per gestirle.
Una volta caricato su un micro **ST62.E25 riprogrammabile** vergine il programma **LCDCRONO.HEX** ed inserito nello zoccolo presente sulla scheda bus **LX.1207**, appena alimenterete il circuito sui **4 display** apparirà il numero:

00:00

Premendo il pulsante **P1** il micro comincerà a contare in avanti ad intervalli di tempo di un **secondo**, quindi sui display vedrete apparire i numeri:

00:01 - 00:02 - 00:03 ecc.

Sui primi due display di sinistra vedrete i **minuti** e sui display di destra i **secondi**.
I **due punti** che separano i display dei minuti e dei secondi lampeggeranno con una cadenza di un secondo.
Come noterete, quando si è raggiunto un tempo di **00:59 secondi**, subito dopo si passerà al tempo successivo di **01:00**, cioè **1 minuto** e **00 secondi**.
Il massimo numero che potrete visualizzare sarà quindi di **99 minuti** e **59 secondi**, dopodichè apparirà **00:00**.
Se in fase di conteggio premerete **P1**, il conteggio si **bloccherà** sul tempo raggiunto e premendolo nuovamente ripartirà dal numero sul quale si era fermato.
Se invece premerete il pulsante **P2**, il conteggio ripartirà da **zero**, cioè il tempo visualizzato si **azzererà**.

LCDOROLO.HEX

Questo programma è un semplice **orologio.**
Per poter visualizzare le **ore** ed i **minuti** dovrete inserire nel **bus LX.1202** la **sola** scheda dei display siglata **LX.1207**.
Se nel bus inserirete le schede dei **relè** o dei **triac**, non potrete renderle attive, perchè nel programma non è presente nessuna istruzione per gestirle.
Una volta caricato su un micro **ST6** vergine il programma **LCDOROLO.HEX** ed inserito nello zoccolo presente nella scheda bus **LX.1202**, non appena alimenterete il circuito sui **4 display** apparirà il numero:

00:00

I primi due display di **sinistra** segneranno le **ore**, mentre i due di **destra** i **minuti**.
I due **punti** che separano i due display lampeggeranno con una cadenza di **1 secondo**.
Come noterete, raggiunte le **ore 23** ed i **59 minuti**, dopo **1 minuto** si passerà alle **24 ore** che verranno visualizzate con **00:00**.
Per mettere a **punto** le **ore** dell'orologio si utilizzerà il pulsante **P2** e, per mettere a punto i **minuti**, il pulsante **P1**.
Facciamo presente che potrete solo **far avanzare** i **numeri** e non **indietreggiare**.

LCDCLOCK.HEX

Questo programma è totalmente diverso dal precedente programma **LCDOROLO**, perchè oltre a visualizzare le **ore** e i **minuti** permette di **eccitare** un **relè** o un **triac** ad un'**ora** prestabilita e di **diseccitarlo** dopo un tempo che voi stessi potrete prefissare modificando alcune righe del programma.
Per farlo funzionare occorre inserire nel **bus LX.1202** la scheda dei display siglata **LX.1207** e quella dei relè siglata **LX.1205**, oppure quella dei triac siglata **LX.1206.**
Prima di spiegarvi quali righe dovrete modificare, consigliamo ai meno esperti di **leggere attentamente** tutto l'articolo, dopodichè potranno modificare i **parametri** nelle sole righe che noi indicheremo.
Come abbiamo accennato, il programma **LCDCLOCK.HEX** ci dà la possibilità di **eccitare** o **diseccitare** uno o più **relè** anche contemporaneamente, su orari che noi stessi potremo stabilire, purchè non si superino più di **8 cicli** o **periodi** nell'arco delle **24 ore**.
Questo programma potrà servire per **accendere** o **spegnere** una o più caldaie, delle insegne luminose ad orari prestabiliti, ecc.

Per mettere a **punto** le **ore** dell'orologio si utilizzerà il pulsante **P2** e per mettere a punto i **minuti** il pulsante **P1**.
Facciamo presente che è possibile soltanto far **avanzare** i **numeri** e non **indietreggiare**.
Appena accenderete l'orologio **tutti** i **4 relè** o **triac** partiranno **eccitati**.
Se volete che all'accensione dell'orologio tutti i relè risultino **diseccitati**, dovrete andare alla riga **N.57** dove troverete questa istruzione:

```
ldi   port_b,11110011b
```

e modificarla inserendo in sostituzione degli **1** degli **0** come qui sotto riportato:

```
ldi   port_b,00000011b
```

Nota = Anche se questa riga è composta da **8 numeri**, dovrete modificare solo i primi **4** di sinistra.

Se volete far **eccitare** all'accensione il solo relè **RL4**, dovrete mettere un **1** in corrispondenza della prima cifra di sinistra come qui sotto riportato:

```
ldi   port_b,10000011
```

A questo punto vi spieghiamo che cosa s'intende per **8 cicli** o **periodi** da utilizzare nell'arco delle **24 ore** che troverete riportati in queste righe:

1° periodo = righe 299 - 300 - 301
2° periodo = righe 306 - 307 - 308
3° periodo = righe 313 - 314 - 315
4° periodo = righe 320 - 321 - 322
5° periodo = righe 327 - 328 - 329
6° periodo = righe 334 - 335 - 336
7° periodo = righe 341 - 342 - 343
8° periodo = righe 348 - 349 - 350

Ogni ciclo è composto da **3 righe** d'istruzioni, quindi nel **1° ciclo** o **periodo** del nostro programma troverete:

```
.byte   02          ;299 riga delle ore
.byte   30          ;300 riga dei minuti
.byte   11100000b   ;301 riga per comando relè
```

Attualmente il **1° ciclo** inizia alle ore **2,30 di notte**.
Per modificare l'orario basterà mettere nella **prima** riga l'**ora** che vi interessa, ad esempio **05-06-10**, e nella **seconda** riga i relativi **minuti**, ad esempio **00 - 10 - 30 - 50**.
Nella **terza** riga sono riportati i relè che desiderate **eccitare** e quelli che **non** desiderate eccitare all'orario da noi prestabilito.

Fig.16 Dopo aver assemblato il programma prescelto (vedi fig.15) pigiate ALT+T poi la lettera P. Quando apparirà questa finestra dovrete premere un tasto qualsiasi.

Fig.17 Come vedrete, sullo schermo apparirà una finestra con tutti i tipi di ST62. Con i tasti freccia ricercate la sigla ST62E25 poi pigiate il tasto Enter.

Fig.18 Prima di programmare l'ST62E25 controllate attentamente che nel riquadro in basso appaia ST62E25. Se appare un'altra sigla dovrete ritornare alla fig.17.

Mettendo un **1** il relè si **ecciterà**, mettendo uno **0** si **diseccciterà**.
Ciò che dovrete modificare in questa terza riga sono **solo** i primi **quattro numeri** di sinistra posti dopo la parola **byte.**
Tenete presente che il **primo** numero di sinistra **piloterà** il relè **RL4** e l'**ultimo** numero di destra il relè **RL1**, quindi avrete in ordine:

RL4-RL3-RL2-RL1

Per farvi capire come modificare tutti questi numeri vi faremo un semplice esempio.
Ammesso che alle **06,10** desideriate **eccitare** i relè **RL2-RL1**, scriverete nelle righe **299 - 300 - 301** (1° ciclo) questi numeri:

```
.byte   06          ;299 riga delle ore
.byte   10          ;300 riga dei minuti
.byte   00110000b   ;301 riga per comando relè
```

Se alle **09,30** vorrete **eccitare** il solo relè **RL4**, dovrete scrivere nelle righe **306 - 307 - 308** (2° ciclo) questi numeri:

```
.byte   09          ;306 riga delle ore
.byte   30          ;307 riga dei minuti
.byte   10000000b   ;308 riga per comando relè
```

Se alle **12,00** vorrete **diseccitare** anche il relè **RL4**, dovrete scrivere nelle righe **313 -314 -315** (3° ciclo) questi numeri:

```
.byte   12          ;313 riga delle ore
.byte   00          ;314 riga dei minuti
.byte   00000000b   ;315 riga per comando relè
```

Se alle **18,45** vorrete **eccitare** tutti i relè, dovrete scrivere nelle righe **320 - 321 - 322** (4° ciclo) questi numeri:

```
.byte   18          ;320 riga delle ore
.byte   45          ;321 riga dei minuti
.byte   11110000b   ;322 riga per comando relè
```

Se alle **22,30** vorrete **diseccitare** i relè **RL4-RL3**, dovrete scrivere nelle righe **327 - 328- 329** (5° ciclo) questi numeri:

```
.byte   22          ;327 riga delle ore
.byte   30          ;328 riga dei minuti
.byte   00110000b   ;329 riga per comando relè
```

Se alle **23,40** vorrete lasciare **eccitato** il solo relè **RL1**, dovrete scrivere nelle righe **334 - 335 - 336** (6° ciclo) questi numeri:

```
.byte   23          ;334 riga delle ore
.byte   40          ;335 riga dei minuti
.byte   00010000b   ;336 riga per comando relè
```

Se alle **24,00** vorrete **diseccitare** anche questo relè, dovrete scrivere nelle righe **341 - 342 - 343** (7° ciclo) questi numeri:

```
.byte   00          ;341 riga delle ore
.byte   00          ;342 riga dei minuti
.byte   00000000b   ;343 riga per comando relè
```

Avendo utilizzato solo **7 cicli** degli **8** disponibili, se l'ultimo non vi interessa lo **potrete cancellare** oppure inibire, mettendo davanti alle righe **348 - 349 - 350** un **punto** e **virgola** o mettendo sulla terza riga **00000000b.**
Potrete **aggiungere** altri cicli se **8** risultassero insufficienti.

Facciamo presente che questi **cicli** si **ripeteranno** automaticamente all'**infinito** agli stessi **orari** tutti i giorni.

LCDTIM90.HEX

Questo programma è un **timer** che, contando in **avanti**, ecciterà un relè o un triac quando raggiungerà i **minuti** e i **secondi** da noi prefissati.
Per farlo funzionare occorre inserire nel **bus LX.1202** la scheda dei display LCD siglata **LX.1207** e quella dei relè siglata **LX.1205**, oppure quella dei triac siglata **LX.1206**.
Non appena alimenterete il circuito, il conteggio partirà da **00:00** e inizierà a contare in **avanti**; a questo punto potrete utilizzare i pulsanti **P1** e **P2** presenti sulla scheda display LCD **LX.1207**.

Premendo **P1** il conteggio si **ferma**.
Premendo nuovamente **P1** il conteggio riparte dal **numero** sul quale si era fermato.
Premendo **P2** il contatore si **resetta**.
Premendo **P1** il contatore riparte da **00:00**.

Nota = Il pulsante **P2** di **reset** sarà attivo solamente se avrete **fermato** il conteggio con **P1.** Se premerete **P2** mentre è **attivo** il conteggio, questo non si azzererà.
I pulsanti presenti sulle schede Triac e Relè non risultano **attivati**.
Il conteggio del display arriva ad un massimo di **89 minuti** e **59 secondi**.

Il programma **LCDTIM90.ASM**, come potrete no-

tare, dispone di **4 cicli** perchè **quattro** sono i **relè** e i **triac** presenti sulle schede sperimentali.

1° ciclo = Dopo **20 secondi** dall'accensione si ecciterà il solo relè **RL1**.
Ovviamente sui display vedrete apparire **19** e, quando questo numero raggiungerà **00:00**, il relè si **ecciterà**.

2° ciclo = Passando al **secondo ciclo**, questo relè rimarrà **eccitato** per un tempo da noi prefissato in **1 minuto** e **30 secondi** e raggiunto questo **tempo** il relè **RL1** si **diseccciterà** e automaticamente si **ecciterà** il relè **RL2**.
Il relè **RL2** si ecciterà un secondo dopo che sui display sarà apparso il numero **01:29** che cambierà in **00:00**.

3° ciclo = Dopo **47 secondi**, cioè quando sul display il numero **46** passerà sullo **00**, il relè **RL2** si **diseccciterà** e si **ecciterà** il terzo relè **RL3**.

4° ciclo = Il conteggio continuerà ed allo scoccare dei **3 minuti** e **00 secondi** (tempo da noi prefissato) si **diseccciterà** il relè **RL3** e si **ecciterà** il relè **RL4**, cioè si ritornerà al **1° ciclo** per ripetere all'infinito i **quattro cicli**.

Per **variare** i **tempi** che noi abbiamo prefissato dovrete variare queste righe:

1° ciclo = righe 292 - 293
2° ciclo = righe 298 - 299
3° ciclo = righe 304 - 305
4° ciclo = righe 310 - 311

Se volete che il **1° ciclo** abbia una durata di **1 minuto** e **30 secondi**, dovrete inserire nelle sue righe questi numeri:

```
ldi   stsex,30   ;292 secondi per RL1
ldi   stmix,1    ;293 minuti per RL1
```

Se volete che il **2° ciclo** abbia una durata di **50 secondi**, dovrete inserire nelle sue righe questi numeri:

```
ldi   stsex,50   ;298 secondi per RL1
ldi   stmix,00   ;299 minuti per RL1
```

Se volete che il **3° ciclo** abbia una durata di **15 minuti** e **20 secondi**, dovrete inserire nelle sue righe questi numeri:

```
ldi   stsex,20   ;304 secondi per RL1
ldi   stmix,15   ;305 minuti per RL1
```

Se volete che il **4° ciclo** abbia una durata di **2 minuti** e **10 secondi**, dovrete inserire nelle sue righe questi numeri:

```
ldi   stsex,10   ;310 secondi per RL1
ldi   stmix,2    ;311 minuti per RL1
```

Nelle righe **295/296 - 301/302 - 307/308 - 313/314** sono riportate le sigle dei **relè** che volete **eccitare** e di quelli che volete rimangano **diseccitati**.
Guardando l'esempio riportato nel programma **LCDCLOCK.ASM** saprete già che scrivendo questa istruzione:

```
ldi   port_b,11110011b
```

potrete eccitare ad ogni **ciclo** anche **più relè** a vostra scelta.
Nei primi **quattro** numeri di sinistra (vedi **1111**) dovrete mettere un **1** sul relè che volete far **eccitare** ed uno **0** se **non** lo volete eccitare.

LCDTEM90.HEX

Questo programma è un **timer** che fa esattamente l'**inverso** del programma **LCDTIM90**, cioè **conta all'indietro** e quando raggiunge lo **00:00** eccita i relè.
I relè, come per il programma precedente, li ecciterete in **4 cicli** e come tempo **massimo** di partenza potrete impostare **90 minuti** e **00 secondi**.
Non appena alimenterete il circuito, il conteggio partirà da **00:20** (questo tempo lo abbiamo prescelto noi, ma poi vi spiegheremo come modificarlo) e procederà all'**indietro**.
Dopo che avrà avuto inizio il conteggio, potrete utilizzare i pulsanti **P1** e **P2** presenti sulla scheda display **LCD LX.1207**.

Premendo **P1** il conteggio si **ferma**.
Premendo nuovamente **P1** il conteggio riparte dal **numero** sul quale si era fermato.
Premendo **P2** il contatore si **resetta**.
Premendo **P1** il contatore riparte dal tempo che avete impostato come **partenza** per il conteggio all'**indietro**.

Nota = Il pulsante **P2** di **reset** sarà attivo solamente se avrete **fermato** il conteggio con **P1**. Se premerete **P2** mentre è **attivo** il conteggio, questo non si azzererà. Premendo **P2** per **resettarlo**, è intuitivo che contando all'**indietro** sul display ritorni il tempo di **partenza**, cioè **00:20**.

Nei **4 cicli** impostati otterrete queste condizioni:

1° ciclo = All'accensione si ecciterà il solo relè **RL1** e sui display apparirà **00:20** e a questo punto avrà inizio il conteggio alla **rovescia** che si fermerà sullo **00:00**.

2° ciclo = Dopo un secondo si ecciterà il relè **RL2** e a questo punto inizierà il **secondo ciclo**, che farà apparire sui display **01:30** (tempo **1 minuto** e **30 secondi**) che, secondo per secondo, decrementerà fino ad arrivare a **00:00**.

3° ciclo = A questo punto si **ecciterà** il relè **RL3** e sui display apparirà **00:47** che decrementerà fino ad arrivare allo **00:00**.

4° ciclo = L'ultimo ciclo farà eccitare il relè **RL4** e farà apparire sui display il numero **03:00** (**3 minuti**). Quando con il conteggio alla **rovescia** si arriverà al numero **00:00**, questo relè si **disecciterà** e contemporaneamente si **disecciterà** il relè **RL1**, cioè si ritornerà al **1° ciclo** per ripetere all'infinito i **quattro cicli**.

Per **variare** i **tempi** prefissati dovrete modificare queste righe:

1° ciclo = righe 295 - 296
2° ciclo = righe 301 - 302
3° ciclo = righe 307 - 308
4° ciclo = righe 313 - 314

Se volete che il **1° ciclo** abbia una durata di **1 minuto** e **30 secondi**, dovrete inserire nelle sue righe questi numeri:

```
ldi   stsex,30   ;295 secondi per RL1
ldi   stmix,1    ;296 minuti per RL1
```

Se volete che il **2° ciclo** abbia una durata di **50 secondi**, dovrete inserire nelle sue righe questi numeri:

```
ldi   stsex,50   ;301 secondi per RL1
ldi   stmix,00   ;302 minuti per RL1
```

Se volete che il **3° ciclo** abbia una durata di **15 minuti** e **20 secondi**, dovrete inserire nelle sue righe questi numeri:

```
ldi   stsex,20   ;307 secondi per RL1
ldi   stmix,15   ;308 minuti per RL1
```

Se volete che il **4° ciclo** abbia una durata di **2 minuti** e **10 secondi**, dovrete inserire nelle sue righe questi numeri:

```
ldi   stsex,10   ;313 secondi per RL1
ldi   stmix,2    ;314 minuti per RL1
```

Nelle righe **297/298 - 303/304 - 309/310 - 315/316** sono riportate le sigle dei **relè** che si **ecciteranno** e di quelli che si **disecciteranno**.
Anche in questo programma possiamo sostituire le righe sopra menzionate con questa sola riga d'istruzione:

```
ldi   port_b,11110011b
```

Nei primi **quattro** numeri di sinistra (vedi **1111**) dove metterete **1** il relè si **ecciterà**, dove metterete **0** si **disecciterà**.

NOTA

Per imparare a programmare i microprocessori ST6 vi consigliamo di rileggere tutti i precedenti articoli riportati sulle riviste N.172/173 - 174 - 175/176 - 179 - 180, perché da oggi in avanti non ripeteremo più quello che vi abbiamo già spiegato.
Sul prossimo numero vi presenteremo un progetto completo dei relativi programmi per gestire un display LCD ALFANUMERICO a più righe, quindi proseguiremo spiegandovi come si dovrà procedere per ottenere dei programmi sempre più perfetti e funzionali.

KIT ESAURITO
perché l'integrato M.8438/AB6
è fuori produzione

COSTO DI REALIZZAZIONE

Tutti i componenti per realizzare questa scheda con display LCD, cioè circuito stampato, connettori maschi, pulsanti, integrato **M.8438/AB6**, display a cristalli liquidi tipo **S.5126** o **LC.513040** (escluso il solo software inserito nel dischetto **DF.1207/3**).....L.58.000

Costo del solo stampato **LX.1207**.....L.9.000

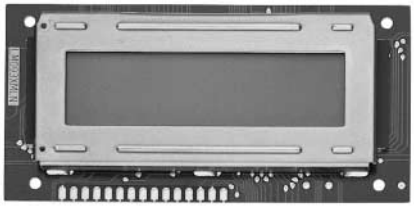
Nota = Per far funzionare questa scheda vi servono i **5 programmi** inseriti nel dischetto siglato **DF.1207/3** del costo di L.12.000

UNA SCHEDA per pilotare

Se guardiamo il lato posteriore di un **normale** LCD vedremo il **vetro** del suo supporto, se guardiamo quello di un LCD **alfanumerico** vedremo un circuito stampato con sopra fissati due integrati in **SMD** provvisti di **62-80** piedini (vedi fig.2).
Se potessimo osservare anteriormente l'interno di un **normale** LCD vedremmo **quattro** caselle con i soliti **7 segmenti** che, accendendosi, ci permettono di far apparire un numero qualsiasi da **0** a **9**.
Poiché questo Display può visualizzare solo dei **numeri** viene definito **numerico**.
Se si potesse osservare, sempre anteriormente, l'interno di un Display **alfanumerico,** si vedrebbero tante **caselle** rettangolari che, anziché essere composte da 7 segmenti, presentano ben **40 punti** distribuiti **8** in senso **verticale** e **5** in senso **orizzontale** (vedi fig.3).
Questi Display, conosciuti anche con il nome di **DMLCD** (vale a dire **D**ot **M**atrix **L**iquid **C**ristal **D**isplay che in italiano significa **D**isplay a **C**ristalli **L**iquidi con **M**atrice di **P**unti), sono chiamati **alfanumerici**.
Infatti, accendendo questi **40 punti** nelle varie combinazioni, potremo far apparire un qualsiasi **carattere alfabetico** maiuscolo o minuscolo, tutti i **numeri** da **0** a **9,** un qualsiasi **simbolo grafico,** come ad esempio **frecce,** $\sqrt{\ }$**,** Ω**,** $\prod$ e, volendo, anche caratteri **cinesi - arabi - greci - cirillici,** ecc.
Questi display **alfanumerici** li possiamo reperire con **1-2-3 righe,** ecc., anche se nel nostro caso ne abbiamo scelto uno con **16 caratteri 2 righe** per spiegarvi come si possa scrivere nella riga superiore ed in quella inferiore.
La definizione **16 caratteri** sta ad indicare che vi sono **16 caselle** per riga, quindi avendo scelto un Display con **2 righe** avremo un totale di **32 caselle** e poiché in ciascuna vi sono **40 punti** potremo accendere ben:

40 x 32 = 1.280 punti

Ammesso di voler visualizzare su un **normale** Display a **7 segmenti** il numero **3,** potremo risolvere il problema con estrema facilità alimentando i soli **segmenti a-b-g-c-d,** ma in un Display a **matrice** composto da **40 punti** le cose diverrebbero ben più complesse perché dovremmo alimentare, nella prima riga superiore i **5 punti** in orizzontale**,** nella seconda-terza-quarta-quinta riga **1 punto** nella posizione richiesta, nella sesta riga **2 punti,** uno all'inizio della riga ed uno alla fine e nella settima riga **3 punti** centrali.
Immaginatevi quindi quanto sarebbe complicato scrivere nelle **32 caselle** delle **frasi** o dei **numeri.**
In teoria lo si potrebbe fare con un microprocessore da **1.280 bit,** ma poiché non esiste, vi chiederete come si possano scrivere in tutte queste caselle **lettere - numeri - simboli**.



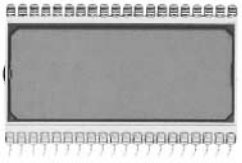


Fig.1 Le dimensioni di un display alfanumerico (vedi a sinistra) sono nettamente superiori a quelle del display numerico riprodotto a destra.

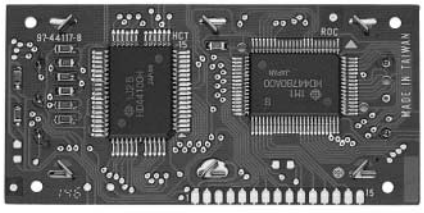
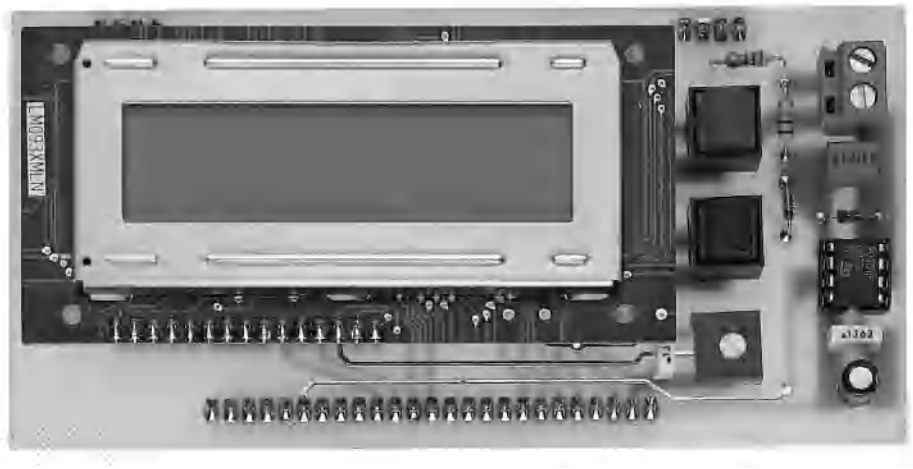


Fig.2 Dal lato opposto del solo display alfanumerico è presente un micro HD.44780 piu' un integrato siglato HD.44100.



un DISPLAY alfanumerico

Oltre ai normali display LCD a 7 segmenti presentati nella rivista N.181, in grado di visualizzare “4 numeri”, esistono anche dei display LCD “alfanumerici” in grado di riprodurre un qualsiasi carattere grafico. In questo articolo vi spiegheremo come dovrete pilotarli per poter far apparire parole - numeri - simboli.

Per questi Display **alfanumerici** si sfrutta la stessa tecnica utilizzata per far apparire sul **monitor** del vostro computer tutte le **lettere** e i **numeri** presenti sulla **tastiera**.
Quando sulla tastiera digitiamo la lettera **A** generiamo un **codice** che, entrando in un integrato **generatore** di **caratteri,** viene trasformato in un **codice ASCII** che provvede a far accendere sul **monitor** tutti i punti richiesti per creare il simbolo **A**.
Lo stesso avviene in questi Display, i quali vengono gestiti da un codice di **8 bit** che, applicato sui piedini **DB0 - DB1 - DB2 - DB3 - DB4 - DB5 - DB6 - DB7** (piedini dal numero **7** al numero **14),** entrerà negli ingressi del microprocessore siglato **HD.44780** (presente sul retro del display) al cui interno è presente una **CGROM**.
La parola **CGROM** significa **C**haracters **G**enerator **R**ead **O**nly **M**emory, cioè lista di caratteri già **memorizzati** al suo interno.
All’interno di questa **CGROM** sono memorizzate tutte le lettere e i simboli visibili nella **Tabella N . 1**, quindi, se sui suoi piedini d’ingresso faremo giungere un **codice** composto da livelli logici **0-1,** selezioneremo nella sua **memoria** la lettera o il simbolo abbinati a questo **codice;** per poter accendere tutti i **punti** necessari per far apparire sul Display la lettera o il simbolo da noi prescelti, il microprocessore **HD.44780** attenderà una conferma dal secondo integrato siglato **HD.44100.**
Detto questo, molti potrebbero pensare che sia sufficiente applicare sui piedini **DB0 - DB1 - DB2 - DB3 - DB4 - DB5 - DB6 - DB7** dei livelli logici **1-0** per far apparire una lettera o un numero.
Chi tentasse di farlo **non vedrebbe** accendersi **nessun punto,** perché i due integrati **HD.44780** e **HD.44100** devono essere gestiti con un complesso **set di istruzioni** che potremo ottenere solo utilizzando un **microprocessore esterno** appositamente programmato.

- Di questo set di istruzioni una parte viene utilizzata per **inizializzare** il microprocessore **esterno,** cioè l’**ST6**.
Le rimanenti istruzioni sono necessarie al Display per prepararsi a ricevere tutti i nostri **dati**, cioè per **configurarsi** correttamente per ricevere i dati in **8 bit** oppure in **4+4 bit.**

Se non utilizzeremo questo **set** di **istruzioni** non riusciremo mai a visualizzare sul Display alcun **carattere**.

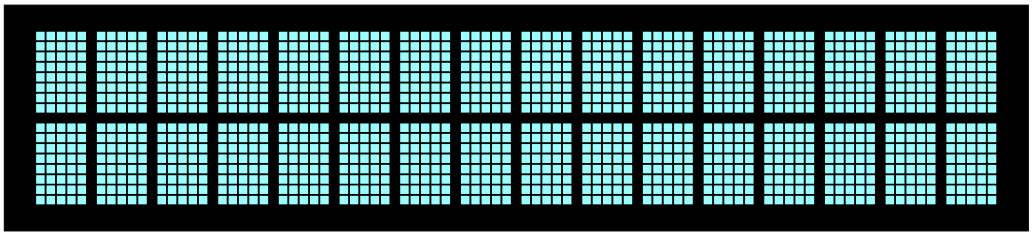


Fig.3 In un display 16 x 2 sono presenti 2 colonne di 16 caselle. In ogni casella vi sono 40 "punti" per accendere i quali occorrerebbe un microprocessore da 1.280 bit.

Per comunicare con il Display con **8 bit** si utilizzano tutti i piedini siglati da **DB0** a **DB7,** mentre per comunicare con **4+4 bit** si utilizzano i soli piedini siglati da **DB4** a **DB7** (gli altri piedini da **DB0** a **DB3** non vengono utilizzati).
Usando **4+4 bit,** verranno inviati al Display i **primi 4 bit,** poi i **successivi 4 bit**.
Nota = Nei nostri programmi abbiamo utilizzato il sistema dei **4+4 bit**.

- Come abbiamo detto**,** quando invieremo un **codice** all'**HD.44780** per far apparire un **carattere,** per poterlo visualizzare questo attenderà tutta una serie di **istruzioni**, ad esempio in quale delle **32 caselle** presenti nel Display vogliamo far apparire il segno **grafico**, se desideriamo utilizzare **entrambe** le **righe** del Display oppure **1 sola,** ecc.
Queste istruzioni verranno accettate solo quando sul **piedino 4** del **display,** denominato **R/S,** sarà presente un **livello logico 0**.

- Dopo aver inserito tutte le **istruzioni** richieste, dovremo mettere a **livello logico 1** il piedino **4** del **display** e solo a questo punto potremo inviare i **dati,** cioè la **lettera - numero - simbolo** che desideriamo far apparire.

- Ai due integrati **HD.44780 - HD.44100** occorre un certo **tempo** per eseguire tutte queste operazioni e questo **tempo** di **lavoro** lo dovremo considerare e **rispettare** anche se si tratta di pochi **millisecondi**, diversamente nella casella interessata potrebbero apparire dei **caratteri** strani e **non significativi.**

Nei programmi dei vari **esempi** che troverete nel dischetto **DF1208** troverete tutte queste istruzioni di **ritardo**, che dovrete necessariamente rispettare quando vi accingerete a scrivere dei vostri personali programmi.
Se non le rispetterete, non riuscirete mai a far funzionare un qualsiasi Display **alfanumerico**.

Questi Display vengono chiamati **intelligenti**, solo perché dispongono di una **memoria** con un **archivio** di caratteri, ma per poter funzionare necessitano sempre di un **microprocessore esterno** (nel nostro caso un **ST62/E25** con **28 piedini**) che indichi loro quali caratteri desideriamo far apparire nelle **32 caselle**.

TABELLA dei CARATTERI PREDEFINITI

Nella **Tabella N.1** abbiamo riprodotto tutti i **caratteri** presenti all'interno della **CGROM**.
Come potrete notare, sul lato **destro** sono presenti **4 bit** indicati con **x x x x** seguiti da altri **4 bit** predefiniti con **0** e **1**, ad esempio:

x x x x 0 0 0 1

In alto sono riportati altri **4 bit** predefiniti con **0** e **1,** ad esempio:

0 0 1 1

Questa Tabella si usa come una Tavola Pitagorica, quindi se volessimo far apparire sul display la lettera **A,** dovremmo sostituire le **x** presenti sul lato **destro** con i bit riportati nella casella **in alto**.
In questo esempio dovremo scrivere:

0 1 0 0 - 0 0 0 1

Nota = Abbiamo messo un segno **-** tra i **primi** quattro bit e i secondi quattro, solo per rendere l'esempio più chiaro, ma questo segno **non dovrete** mai inserirlo.

Se volessimo far apparire una **a** (minuscola) dovremmo scrivere:

0 1 1 0 - 0 0 0 1

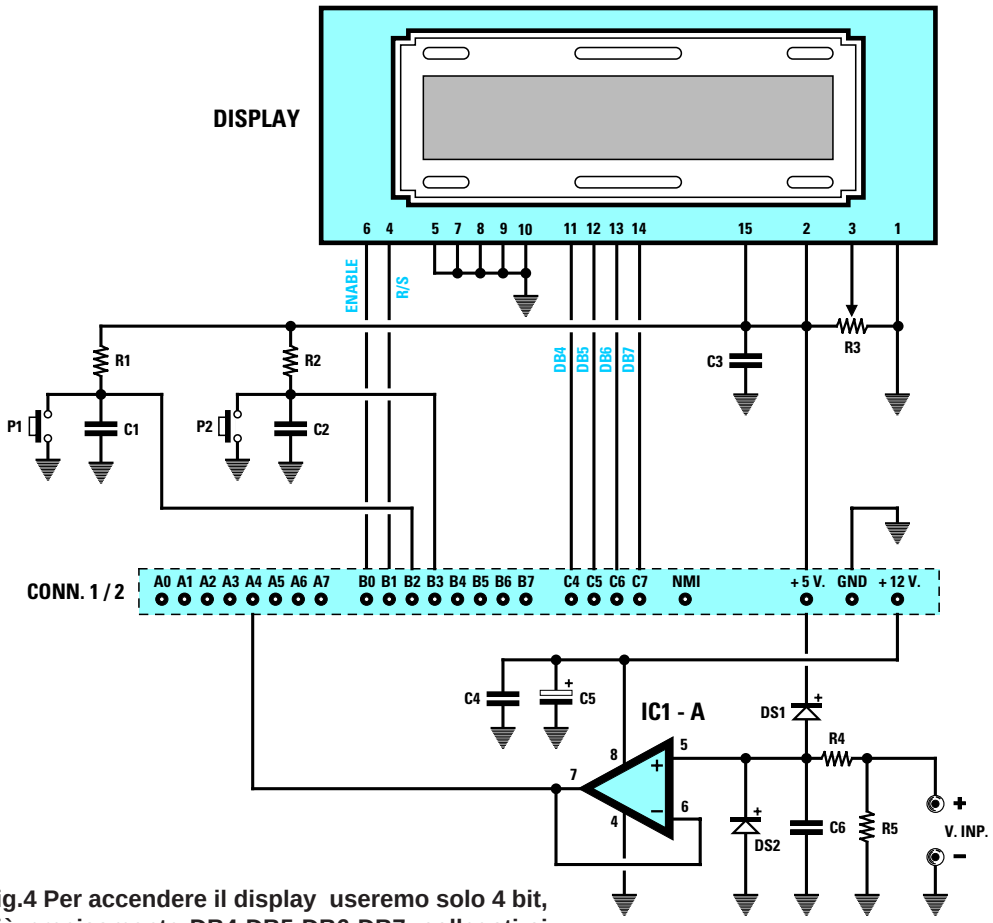


Fig.4 Per accendere il display useremo solo 4 bit, più precisamente DB4-DB5-DB6-DB7, collegati ai piedini C4-C5-C6-C7 del Connettore d'ingresso.

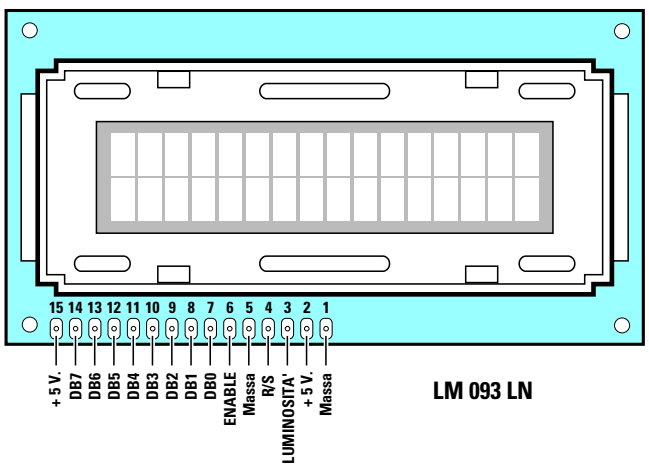


Fig.5 Gli altri bit che non vengono utilizzati, cioè DB0-DB1-DB2-DB3 che fanno capo ai piedini 7-8-9-10, andranno collegati a massa.

ELENCO COMPONENTI LX.1208

R1 = 10.000 ohm 1/4 watt
R2 = 10.000 ohm 1/4 watt
R3 = 10.000 ohm trimmer
R4 = 10.000 ohm 1/4 watt
R5 = 1 megaohm 1/4 watt
C1 = 100.000 pF poliestere
C2 = 100.000 pF poliestere
C3 = 100.000 pF poliestere
C4 = 100.000 pF poliestere
C5 = 10 mF elettr. 63 volt
C6 = 1 mF poliestere
DS1 = diodo tipo 1N.4150
DS2 = diodo tipo 1N.4150
IC1 = LM.358
DISPLAY = LCD tipo LM.093X
CONN.1/2 = connettore 24 poli
P1 = pulsante
P2 = pulsante

Se volessimo far apparire il segno grafico **>** dovremmo scrivere:

0 0 1 1 - 1 1 1 0

Anziché utilizzare questo **codice binario** per scrivere una **lettera** o un **carattere,** potremo usare anche un **codice decimale** e per questo motivo abbiamo inserito sotto ad ogni casella il rispettivo numero **decimale**.
Quindi scrivendo **65** sul Display apparirà la lettera **A maiuscola** e scrivendo **97** apparirà la lettera **a minuscola**.

Esempio in codice Binario

Per scrivere la lettera **A** in codice **binario** dovremo scrivere questa istruzione:

ldi car,01000001b

Esempio in codice Decimale

Per scrivere la lettera **A** in codice **decimale** dovremo scrivere questa istruzione:

ldi car,65

Esempio in codice ASCII

Anziché utilizzare un codice **binario** o **decimale,** potremo scrivere direttamente in **ASCII** ed in questo caso l'istruzione sarà la seguente:

car .ascii "A"

Tra i programmi dimostrativi riportati nel dischetto **DF1208** ne abbiamo inseriti diversi utilizzando questi tre diversi **codici**, quindi leggeteli attentamente perché con le istruzioni riportate comprenderete con estrema facilità quello che risulterebbe assai più complesso spiegare a parole.

OPERAZIONI MATEMATICHE

Molti si trovano in difficoltà con le operazioni **matematiche,** perché non pensano che il numero che desiderano far apparire è un **carattere grafico** che verrà prelevato all'interno della **CGROM**.
Nel caso della somma **3+2** che ci dà come risultato **5,** consultando la **Tabella N. 1** vedremo che sotto al numero **5** è indicato il numero **53**.

Per far apparire sul display il segno grafico **"5",** a questo numero dovremo sommare la **costante 48** e così facendo otterremo **5+48 = 53** e se andiamo a vedere nella **Tabella N. 1** noteremo che il numero **decimale 53** corrisponde effettivamente al carattere grafico **"5"**.
Pertanto, l'**istruzione** che dovremo scrivere per svolgere questa operazione sarà:

```
ldi a,3
addi a,2
addi a,48
ld ddata,a
call dsend
```

Se svolgiamo questa seconda operazione **9+7 = 16** otterremo un risultato di **due** cifre, quindi per far apparire questi due segni grafici dovremo sommare a **1** la costante **48** e, in tal modo, otterremo **49**; consultando la **Tabella N. 1** noteremo che il numero **49** corrisponde al segno grafico **"1"**.
Sommando al numero **6** la costante **48** otterremo **6 + 48 = 54** e sempre guardando la **Tabella N. 1** scopriremo che **54** corrisponde al segno **grafico "6**".
Pertanto l'**istruzione** che dovremo scrivere per questa operazione sarà:

```
ldi a,1
addi a,48
ld ddata,a
call dsend

ldi a,6
addi a,48
ld ddata,a
call dsend
```

ISTRUZIONI di INIZIALIZZAZIONE

Come abbiamo già detto, quando scriverete dei nuovi programmi dovrete sempre **iniziare** con tutta una serie di istruzioni di inizializzazione.
Nei due programmi che troverete nel dischetto **DF1208** questo **set** di **istruzioni** sono riportate:

nel programma **DISP093** nelle **righe 77-109**
nel programma **TESTER** nelle **righe 81-113**

In questi due programmi questo **set** di **istruzioni** è posizionato nelle righe **77-109** e nelle righe **81-113** solo perché prima di queste abbiamo dovuto riportare due diverse serie di **variabili** necessarie per far funzionare i due programmi.

Come noterete questi due **set** di **istruzioni**, anche se posti in righe diverse, sono perfettamente identici.
Dovrete **sempre** riportare nei vostri programmi personalizzati tutte queste righe senza apportare alcuna modifica, dopo tutte le vostre **variabili**.

TABELLA n. 1

0010	0011	0100	0101	0110	0111	1010	1011	1100	1101	1110	1111	→
32	 48	 64	 80	 96	 112	160	176	 192	 208	 224	 240	xxxx0000
33	 49	 65	 81	 97	 113	 161	 177	 193	 209	 225	 241	xxxx0001
 34	 50	 66	 82	 98	 114	 162	 178	 194	 210	 226	 242	xxxx0010
 35	 51	 67	 83	99	115	163	179	195	211	227	243	xxxx0011
36	52	68	84	100	116	164	180	196	212	228	244	xxxx0100
37	53	69	85	101	117	165	181	197	213	229	245	xxxx0101
38	54	70	86	102	118	166	182	198	214	230	246	xxxx0110
39	55	71	87	103	119	167	183	199	215	231	247	xxxx0111
40	56	72	88	104	120	168	184	200	216	232	248	xxxx1000
41	57	73	89	105	121	169	185	201	217	233	249	xxxx1001
42	58	74	90	106	122	170	186	202	218	234	250	xxxx1010
43	59	75	91	107	123	171	187	203	219	235	251	xxxx1011
44	60	76	92	108	124	172	188	204	220	236	252	xxxx1100
45	61	77	93	109	125	173	189	205	221	237	253	xxxx1101
46	62	78	94	110	126	174	190	206	222	238	254	xxxx1110
47	63	79	95	111	127	175	191	207	223	239	255	xxxx1111

PER PASSARE dalla 1° alla 2° RIGA

Poiché normalmente si scrive partendo dalla **1° riga** per poi passare alla **2° riga,** le prime istruzioni che dovrete scrivere saranno:

res 1,port_b "prepararsi per l'istruzione"
ldi ddata,00000010b "istruzione per la **1° riga**"
call dsend "subroutine per invio dati"

Continuerete quindi con le istruzioni che servono ad **incrementare** di una casella, cioè a far sì che la prima lettera o numero che vorrete far apparire si posizioni automaticamente nella **prima** casella, la seconda lettera nella **seconda** casella, ecc.
Ammesso di voler scrivere **A**, dovrete scrivere il suo numero **decimale**, quindi:

ldi ddata,00000110b "incrementa di una casella"
call dsend "subroutine per invio dati"
set 1,port_b "fine set istruzioni"
ldi ddata,65 "trasferisci A in ddata"
call dsend "subroutine per invio dati"

Per scrivere 16 caratteri in ogni **riga** si potrebbe scrivere 16 volte questa istruzione, mettendo nella riga **ldi ddata** il numero **decimale** che si desidera far apparire, ma poiché questa soluzione risulta **poco pratica,** vi consigliamo di andare a vedere nel programma **DISP093** come abbiamo risolto in modo più elegante il problema per far apparire sul Display la parola **N.ELETTRONICA**.

Per scrivere nella **2° riga** posta sotto la **1°,** dovrete scrivere queste istruzioni:

res 1,port_b "prepararsi per l'istruzione"
ldi ddata,11000000b "posizionamento in **2° riga**"
call dsend "subroutine per invio dati"
set 1,port_b "fine set istruzioni"

Ammesso di voler far apparire nella **seconda casella** la lettera **B,** dovrete scrivere:

ldi ddata,66 "trasferisci **B** in ddata"
call dsend "subroutine per invio dati"

Vorremmo aggiungere che anche se sul display sono **visibili** solo **16 caselle** per **riga,** in pratica ve ne sono per ogni riga altre **24 nascoste** e queste righe **nascoste** possono servire nel caso si desiderino far scorrere sul display delle **scritture** da destra verso sinistra o viceversa.
Nel programma **DISP093** che troverete nel dischetto **DF1208**, oltre a tutte le **sorgenti** abbiamo riportato anche degli esempi per ottenere questa funzione.

NOTA per l'EDIT dell'ST6

Dobbiamo precisare che l'**EDIT,** che vi avevamo fornito nei precedenti dischetti **LX.1207** con l'intento di semplificare tutte le operazioni, risulta **insufficiente** per programmi molto **lunghi** come quelli utilizzati per questo Display **alfanumerico** siglato **LX.1208.**
Infatti questo **EDIT** accetta solo programmi che **non superino** i **30 Kilobyte** quindi, quando li andrete a **salvare**, tutto quello che **eccede** i **30 K** verrà **inesorabilmente cancellato**.
Se perciò vorrete **modificare** e trasferire nella memoria dell'**ST6** un programma per questo Display **alfanumerico** o altri che superino i **30 K,** dovrete **necessariamente** utilizzare l'**Editor** del **DOS** presente nel vostro computer.

Per caricare i due programmi presenti nel dischetto **DF1208** dovrete procedere come segue:

Quando sul monitor appare **C:\>** dovrete inserire il dischetto nel drive **A** e scrivere:

C:\>A: poi premete Enter
A:\> installa poi premete Enter

Il programma vi chiederà su quale **directory** volete installare il contenuto del dischetto e, poiché noi l'abbiamo già definita **LX1208,** dovrete solo premere il tasto Enter.

Si creerà così automaticamente la directory **LX1208** e mentre verranno trasferiti nell'Hard-Disk tutti i programmi presenti nel dischetto floppy verranno anche **scompattati**.

Se usando questo metodo vi apparirà la scritta **error,** vi consigliamo di ricaricare il dischetto nell'Hard-Disk utilizzando questo secondo metodo:

C:\>MD LX1208 poi premete Enter
C:\>COPY A:*.* C:\LX1208 poi premete Enter
C:\>CD LX1208 poi premete Enter
C:\LX1208>installa poi premete Enter

Nota = Per agevolarvi a rispettare le spaziature, abbiamo utilizzato una barra in **colore** che corrisponde ad uno **spazio**.

Ammesso che desideriate modificare il programma **TESTER** presente nel disco **DF.1208,** quando sul monitor appare **C:\>** scrivete:

C:\>CD LX1208 (chiama la directory)
C:\LX1028>Edit TESTER.ASM (chiama Editor del Dos)

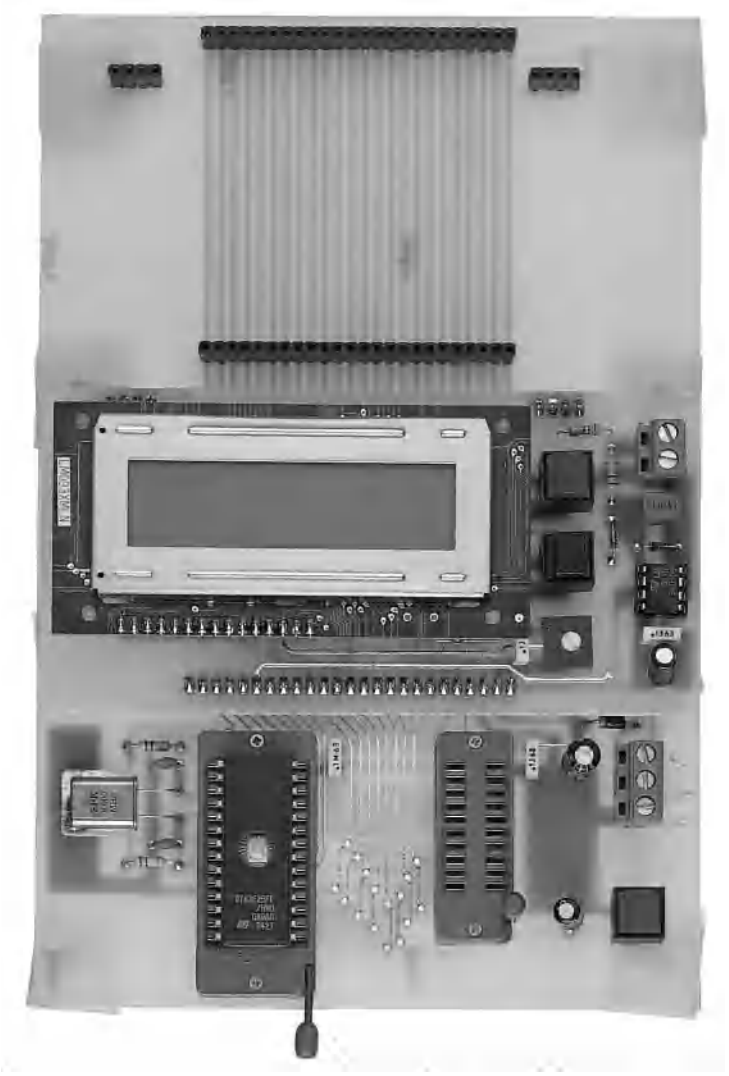


Fig.6 La scheda di questo display andrà inserita nel Bus siglato LX.1202, non dimenticando di innestare nello zoccolo textool un micro ST6 tipo ST6/E25 per trasferire il programma DISP093.HEX oppure il TESTER.HEX.

IMPORTANTE

Con questo programma **LX.1208** non viene più utilizzato l'**Editor** dell'**ST6** ma l'**Editor** del **DOS,** quindi per trasferire i programmi dall'Hard-Disk alla **memoria** dell'**ST62/E25** non dovrete più convertire i programmi da **.ASM** in **.HEX** come vi avevamo insegnato in precedenza per tutti i programmi presenti nel dischetto **LX.1207,** ma dovrete procedere in modo completamente diverso.

Dopo aver eseguito tutte le modifiche sui programmi dovrete premere i tasti **ALT F,** poi portare il cursore sulla riga **SALVA** e premere Enter ed infine sulla riga **ESCI** e premere Enter.

Ammesso di voler compilare il programma **DISP093,** quando sul monitor apparirà **C:\LX1208>** dovrete scrivere**:**

C:\LX1208>A DISP093.ASM

Nota = Dopo la lettera **A** non mettete "**:**" perché questa **A** è un programma **Batch.**

Con questa istruzione convertirete **automaticamente** il programma da **.ASM** a **.HEX**.

Per trasferire i programmi già compilati in **.HEX** nel microprocessore posto sull'interfaccia **LX.1202,** dovrete richiamare la **directory LX1208** e poi scrivere semplicemente:

C:\LX1208>ST6PGM poi premere Enter

A questo punto sul monitor apparirà una maschera che vi chiederà quale programma intendete trasferire e quale **micro** avete scelto e, una volta che avrete risposto a queste domande, potrete memorizzare il vostro micro **ST62/E25**.
Tutte le istruzioni relative a come trasferire un programma dall'Hard-Disk ad un micro **ST6** le abbiamo riportate negli articoli pubblicati sulle riviste **N.172/173-174-175/176-179-180-181,** quindi a chi fosse interessato a questo argomento suggeriamo di procurarsi tali numeri al più presto prima che vengano esauriti.

SCHEMA ELETTRICO

Lo schema elettrico di questo progetto, come potete vedere in fig.4, è quanto di più semplice si possa immaginare.
Abbiamo contrassegnato **otto** dei **15 piedini** del display con le sigle da **DB0** a **DB7** per non confonderli con i segnali da **B0** a **B7** presenti nel connettore che va inserito nella scheda bus **LX.1202.**
Nell'articolo abbiamo spiegato che per gestire questo display bisogna usare un codice di **8 bit,** che andrà applicato sui piedini **DB0 - DB1 - DB2 - DB3 - DB4 - DB5 - DB6 - DB7** (piedini dal numero **7** al numero **14**), mentre osservando lo schema elettrico riportato in fig.4 si potrà notare che i piedini da **DB0** a **DB3** sono collegati a **massa** e che il segnale entra nei soli piedini da **DB4** a **DB7**.
Questa configurazione è stata adottata perché per gestire questo display abbiamo usato un codice **4+4 bit**.
Oltre al display, nello schema elettrico è presente anche un amplificatore operazionale siglato **IC1/A;** a questo proposito vi chiederete se tale amplificatore sia indispensabile per far funzionare questo display e noi vi rispondiamo che **non serve.**
Infatti l'abbiamo **inserito** soltanto per potervi dimostrare come sia possibile, con il **programma Tester**, trasformare questo display in un **voltmetro.**
Dobbiamo subito precisare che nell'ingresso di questo **operazionale** non è possibile inserire delle tensioni superiori ai **5 volt;** per poterlo fare sarà necessario applicare sull'ingresso dei partitori **resistivi** da **1/10 - 1/100**.
Oltre a questo particolare, dobbiamo anche ricordarvi di rispettare la **polarità** della tensione sull'ingresso, perché se **invertirete** il positivo con il negativo sul display appariranno **0 volt**.
Il trimmer **R3** collegato al piedino **3** serve per variare la **luminosità** delle lettere o dei numeri che appariranno nelle diverse caselle.
Tutti i piedini del **display** e quello d'uscita dell'**operazionale** vengono collegati al **Connettore 1/2** che andrà innestato nella scheda bus **LX.1202**.

REALIZZAZIONE PRATICA

Sul circuito stampato siglato **LX.1208** dovrete montare tutti i componenti visibili in fig.7.
Vi consigliamo di iniziare dal **connettore maschio** a **24 terminali** e di procedere inserendo gli altri due **connettori maschi** a **4 terminali** (nello schema pratico si vede solo quello di destra) ed il **connettore femmina** a **15 terminali** che userete come zoccolo per il display.
Completata questa operazione, potrete inserire lo zoccolo per l'integrato **IC1** e tutti gli altri componenti richiesti, cioè pulsanti, trimmer, condensatori e resistenze.
Nel montaggio dovrete solo rispettare la polarità dei due diodi al silicio **DS1-DS2,** posizionando il lato del loro corpo contornato da una **fascia nera** come appare ben visibile nello schema pratico di fig.7.
Completato il montaggio, dovrete inserire nello zoccolo l'integrato **IC1,** rivolgendo la tacca di riferimento a forma di **U** presente sul suo corpo verso il condensatore **C4**.
Per fissare il **display** dovrete inserire nei quattro fori presenti sullo stampato i distanziatori plastici inclusi nel kit, dopodichè potrete innestare i terminali del display nello zoccolo femmina, facendo entrare i perni dei distanziatori nei fori presenti sullo stampato del display.

PROGRAMMI PER LM093

Nel dischetto **DF1208** sono riportati due programmi per utilizzare il display **alfanumerico** in tutti i modi possibili.

Questi due programmi sono denominati:

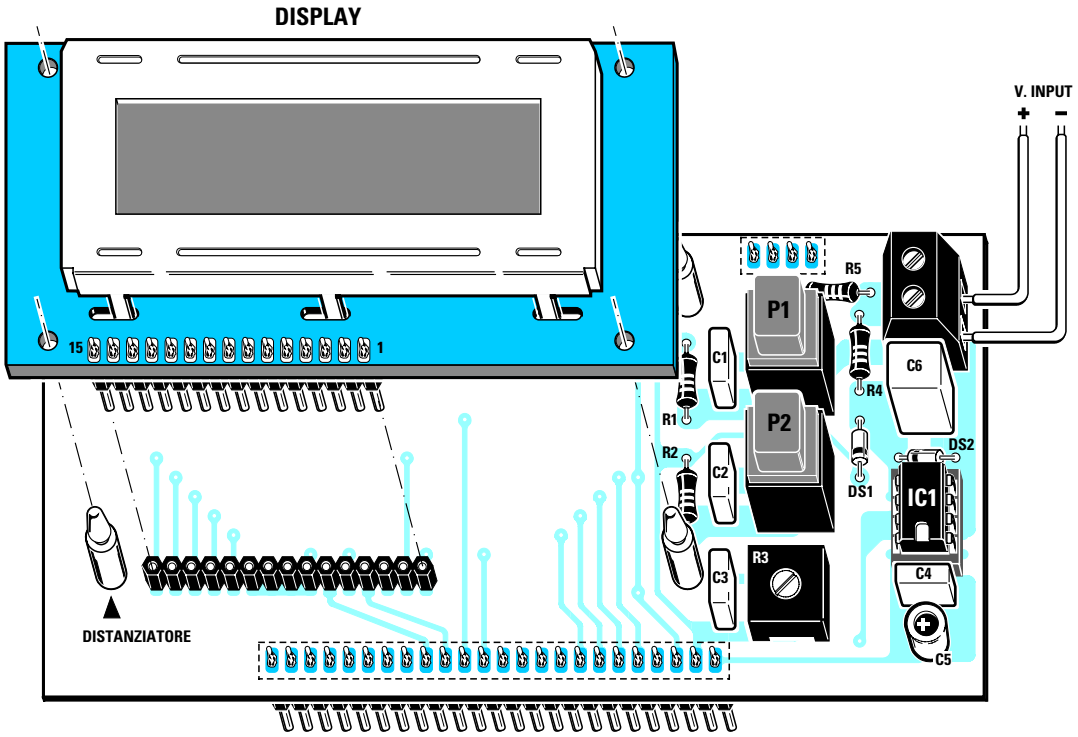
DISP093.ASM
TESTER.ASM

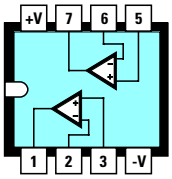
A questi due programmi occorrono altri due **files** chiamati:

TB_CGR01.ASM
TB_CGR02.ASM

Questi due ultimi files **TB_CGR** sono in pratica delle **tabelle** che vi serviranno per utilizzare una **direttiva** denominata **.input**.
Questa direttiva altro non è che una **istruzione** inserita nel file sorgente, che in fase di compilazione "richiama" una serie di **dati** contenuti in un file **diverso** dal sorgente.

Quando **assemblerete** il programma **DISP093.A-**





LM 358

Fig.7 Schema pratico di montaggio della scheda LX.1208. Sul circuito stampato dei display dovrete saldare il connettore maschio a 15 terminali, che dovrete poi innestare nel circuito stampato LX.1208.

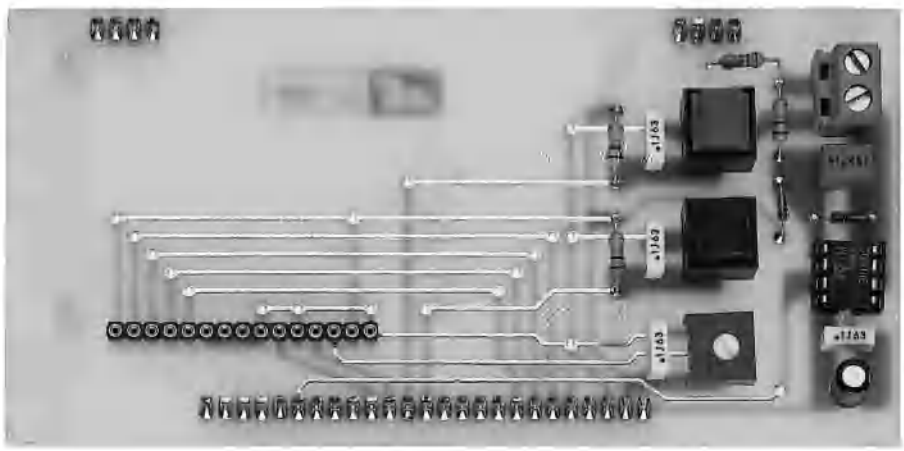


Fig.8 Foto dello stampato LX.1208 con sopra già montati tutti i componenti richiesti. Nota = Nel kit, anzichè trovare un connettore maschio da 24 poli e due da 4 poli potreste trovarne uno solo da 32 terminali, che dovrete tagliare per ottenere i tre pezzi richiesti.

SM, il compilatore andrà a ricercare il file:

TB_CGR01.ASM

e lo ingloberà al suo interno, formando così un **unico file** che si chiamerà: **DISP093.HEX**.

Quando **assemblerete** il programma **TESTER.ASM,** il compilatore andrà a ricercare il file **TB_CGR02.ASM** e lo ingloberà al suo interno, formando così un **unico file** che si chiamerà: **TESTER.HEX**.

Quindi quando **assemblerete** questi due programmi non dovrete assolutamente preoccuparvi di questi files **TB_CGR,** perché le operazioni di ricerca e di inserimento vengono eseguite automaticamente.
La direttiva chiamata **.input** è simile ad una subroutine, con la sola differenza che viene eseguita in fase di compilazione e non durante l'esecuzione del programma stesso.

Per terminare aggiungiamo che una volta inserita la **LX.1208** di questo display **alfanumerico** nel bus **LX.1202,** in quest'ultimo bus non potrete inserire altre schede, tipo **TRIAC** o **RELE**'.

Programma DISP093.HEX

Questo programma, che utilizza i soli pulsanti **P1 - P2** presenti nella scheda **LX.1208,** vi permetterà di visualizzare il vostro **nome** e **cognome** o qualsiasi altra **scritta** sul display, a condizione di utilizzare un **massimo** di **16 caratteri** per riga.
Facciamo presente che è necessario considerare gli **spazi** come **caratteri**.

Il vostro nome e cognome o una qualsiasi altra scritta, dovrà essere scritto all'**interno** del programma **.ASM** nelle **righe** che vi indicheremo.
Dopo aver scritto le parole che dovranno apparire sul display, dovrete **riassemblare** il programma ed infine caricarlo nel micro **ST62/E25,** che andrà inserito nello zoccolo **textool** presente nella scheda bus **LX.1202**.
Se caricherete nel micro **ST62/E25** il programma **DISP093.ASM,** ovviamente dopo averlo assemblato in **DISP093.HEX**, sul display vedrete apparire in ordine alcune scritte:

N.ELETTRONICA
**** DISP093 ****

Queste scritte rimarranno visualizzate per circa **3 secondi**, dopodichè apparirà:

[P1] >
PER PROCEDERE

Se non premerete il pulsante **P1** vedrete alternarsi le due scritte sopra riportate con una cadenza di circa **1 secondo**.

Se invece premerete il pulsante **P1** per più di **3 secondi** circa, apparirà questa nuova scritta:

-NOME-NOME-NOME-
-COGNOME-COGNOME

Come noterete, ogni riga occupa un totale di **16 caratteri**.

Questa scritta rimarrà visualizzata sul display per circa **6 secondi,** dopodichè apparirà la scritta:

LE FUNZIONI
PREVISTE SONO:

Anche questa scritta rimarrà visualizzata per circa **6 secondi**, dopodichè apparirà questa scritta:

1-MAIUSC.> minus
2-ROTAZIONE

Questa rimarrà visualizzata per circa **6 secondi,** dopodichè apparirà:

3-SCOMPOSIZIONE
4-VISUAL.CG RAM

e nuovamente vedrete apparire:

[P1] >
PER PROCEDERE

Come noterete, sulla base delle scritte apparse, potrete ottenere **4 diverse** funzioni che sono numerate **1-2-3-4**.
Solo dopo che saranno apparse **tutte** le scritte che vi abbiamo sopra indicato, potrete utilizzare il **tasto P1** per selezionare **una** della **4 funzioni**.
Se **non premerete** il pulsante **P1** vedrete nuovamente ripetersi all'infinito le stesse scritte.
Quando apparirà **P1 > PER PROCEDERE** dovrete tenere premuto questo tasto per almeno **3** secondi e apparirà la scritta:

SCELTA FUNZIONE
.....[?] [P2] >

A questo punto, utilizzando il tasto **P2** potrete scegliere una della quattro funzioni numerate **1-2-3-4**.

Questa scritta rimarrà sui display fino a quando non premerete il pulsante **P2**.
Se terrete premuto per almeno **3 secondi** il pulsante **P2,** apparirà questa scritta:

SCELTA FUNZIONE
.....[1] [P2] >

Come potete vedere, nelle due parentesi quadre è sparito il **?** ed è apparso il numero **1.**
Premendo per una **seconda** volta **P2** apparirà il numero **2**, premendo una **terza** volta **P2** apparirà il numero **3** e premendolo per la **quarta** volta apparirà il numero **4.**
Fate attenzione a premerlo per la **quinta** volta, perché se sul display apparirà il numero **5** uscirete dal **menu.**

Ammesso di voler visualizzare la funzione **1-MAIUSC.> minus** quando appare:

SCELTA FUNZIONE
.....[1] [P2] >

dovrete premere per circa **3 secondi** il pulsante **P1** e comparirà la scritta:

-NOME-NOME-NOME-
-COGNOME-COGNOME

dopo circa **5 secondi** vi apparirà la stessa scritta ma in **minuscolo**, ovvero:

-nome-nome-nome-
-cognome-cognome

Tale scritta in **minuscolo** resterà visualizzata per circa **5** secondi, dopodichè vi riapparirà nuovamente la scritta:

SCELTA FUNZIONE
.....[?] [P2] >

A questo punto, se premerete il tasto **P2** per **due** volte consecutive, sceglierete la funzione **2-ROTAZIONE,** quindi quando apparirà:

SCELTA FUNZIONE
.....[2] [P2] >

tenendo premuto per **3 secondi** il tasto **P1,** vedrete apparire un divertente effetto perché il vostro nome partirà dalla seconda riga, scorrendo da sinistra verso destra, per poi riportarsi sulla prima riga scorrendo da destra verso sinistra per **4 volte** consecutive, dopodichè riapparirà la scritta:

SCELTA FUNZIONE
.....[?] [P2] >

Se ora premerete il tasto **P2** per **tre** volte consecutive, sceglierete la funzione **3-SCOMPOSIZIONE,** quindi quando apparirà:

SCELTA FUNZIONE
.....[3] [P2] >

dovrete tenere premuto il pulsante **P1** per circa **3 secondi** e rilasciandolo vedrete che i caratteri riportati nella sola prima riga cominceranno a **scomporsi**, cioè vedrete i caratteri della **prima riga** allontanarsi **ad uno ad uno** scorrendo verso destra, fino a scomparire totalmente, poi li vedrete ritornare da destra verso sinistra fino a **ricostruire** l'intera parola.
Una volta ricostruiti i **16** caratteri sulla prima riga, nuovamente vedrete apparire la scritta:

SCELTA FUNZIONE
.....[?] [P2] >

Se ora premerete il tasto **P2** per **quattro** volte consecutive, sceglierete la funzione **4-VISUAL.CG RAM** quindi quando apparirà:

SCELTA FUNZIONE
.....[4] [P2] >

dovrete sempre tenere premuto il tasto **P1** per almeno **3 secondi** e sulla prima riga del display vedrete apparire **8 simboli grafici** generati appositamente a scopo didattico, più un cursore non lampeggiante.

I programmi inseriti nel dischetto **DF.1208** servono principalmente per farvi vedere come si debbano scrivere le varie istruzioni per far funzionare questo display **alfanumerico**.
Solo dopo che avrete preso una certa confidenza con questi programmi, potrete modificarli, o prelevare direttamente dalle nostre **sorgenti** tutte le righe che potrebbero interessarvi.
Una modifica **molto semplice** che potrete apportare è quella di far apparire sui display il vostro **nome** e **cognome** o qualsiasi altra scritta.

Se sulla **prima riga** volete far apparire il vostro **nome** che potrebbe essere **ALESSANDRO - MARCO - VINCENZO,** ecc. dovrete andare alla **riga**

N.684 posta all'interno del programma sorgente **DISP093.ASM** e sostituire la scritta da noi inserita con il vostro nome.

Se in corrispondenza della **seconda riga** volete far apparire il vostro **cognome** che potrebbe essere **BIANCHI - ALBERTAZZI - FANTOZZI,** ecc., dovrete andare alla **riga N.685** e sostituire la scritta da noi inserita con il vostro cognome.

Nota importante = Qualsiasi cosa scriverete nelle righe **684** e **685,** dovrete sempre farlo in caratteri **maiuscoli** e non superare mai i **16 caratteri** per riga compresi gli **spazi**.
Dopo aver eseguito queste modifiche dovrete premere i tasti **ALT F,** poi portare il cursore sulla riga **SALVA** e premere Enter ed infine sulla riga **ESCI** e premere Enter.
Dopodichè dovrete richiamare il programma **LX1208** scrivendo:

C:\>CD LX1208 poi premere Enter

e scrivere:

C:\LX1208>A DISP093.ASM

Nota = Dopo la lettera **A** non mettete **":",** perché questa **A** non è altri che un programma Batch che lancia il compilatore in assembler**.**

Con questa istruzione convertirete **automaticamente** il nostro programma da **.ASM** in **.HEX**.

Per trasferire questo programma già compilato in **.HEX** nel microprocessore posto sull'interfaccia **LX.1202,** dovrete richiamare la **directory LX1208** e poi scrivere semplicemente:

C:\LX1208>ST6PGM poi premere Enter

A questo punto sul monitor apparirà una maschera che vi chiederà quale programma desiderate trasferire e su quale **micro ST6;** fornite al computer le esatte risposte, il vostro programma modificato verrà memorizzato nel micro **ST62/E25**.

TESTER.HEX

Con questo programma dimostrativo desideriamo insegnarvi ad utilizzare l'**A/D** converter presente all'interno del microprocessore **ST6/E25** e per farlo abbiamo realizzato con questo display **alfanumerico** un semplice **voltmetro elettronico** utilizzando entrambe le righe presenti nel display.

Sulla **prima riga** faremo apparire il valore della tensione in **numero,** mentre sulla **seconda riga** faremo apparire una **barra** che si allungherà di **1 riga** ogni **0,1 volt** e di **1 quadretto** ogni **0,5 volt**.
Chi fosse interessato a comprendere come siamo riusciti ad ottenere queste due condizioni, dovrà leggere attentamente il programma **TESTER.ASM** e i **commenti** riportati su ogni riga.
Vogliamo subito far presente che il **massimo valore** di tensione che potremo leggere con questo voltmetro è di soli **5 volt,** quindi non applicate sull'ingresso dell'operazionale **IC1/A** tensioni maggiori.
Per leggere tensioni di **50 volt fondo scala,** dovrete necessariamente utilizzare dei partitori resistivi come illustrato nelle figg.10-11.

La tensione da misurare, applicata sull'ingresso dell'operazionale **IC1/A,** verrà prelevata dal suo piedino d'uscita **7** ed inviata all'**A/D converter** presente all'interno del microprocessore **ST6/E25,** che la convertirà in un numero **decimale** compreso tra **0** e **255**.

L'**A/D converter** per un valore di tensione di **5 volt** ci dà un numero **decimale** di **255;** se dividiamo **255** per **5** otteniamo **51,** quindi è intuitivo che per un valore di tensione di **1 volt** l'**A/D converter** ci darà un numero **decimale** di **51** e per un valore di **2 volt** ci darà un numero **decimale** di **102** e per **3 volt** un numero **decimale** di **153.**
Se misurassimo una tensione di **2,5 volt,** in teoria l'**A/D converter** dovrebbe darci il numero **51 x 2,5 = 127,5**, ma poiché in pratica non ci darà mai un numero con la **virgola,** sulla sua uscita otterremo dei numeri variabili molto prossimi a **127,** ad esempio **127-128-128-129-127-129,** perché l'**A/D converter** dell'ST6 non è molto stabile.

Sommando i **6** numeri del nostro esempio otterremo un totale di **768**, che diviso per **6** ci darà il valore medio:

768 : 6 = 128

Dividendo **128** per **51** otterremo:

128 : 51 = 2,50

Per ottenere una maggiore precisione nel nostro programma leggeremo i **numeri decimali** che l'**A/D converter** ci fornirà per ben **32 volte,** poi una volta **sommati** li divideremo per **32.**
Per far apparire il numero **2,5** metteremo il numero **2** in un **byte** e il numero **5** in un altro **byte.**

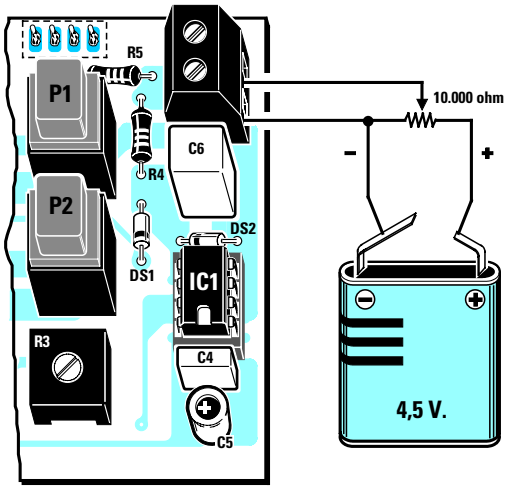


Fig.9 Il programma TESTER.HEX permette di utilizzare questo display alfanumerico in un Voltmetro in grado di misurare un massimo di 5 volt. Per provare questo Voltmetro potrete procurarvi un trimmer da 10.000 ohm ed una pila da 4,5 volt.
Ruotando il suo cursore vedrete apparire sui display il valore della tensione.

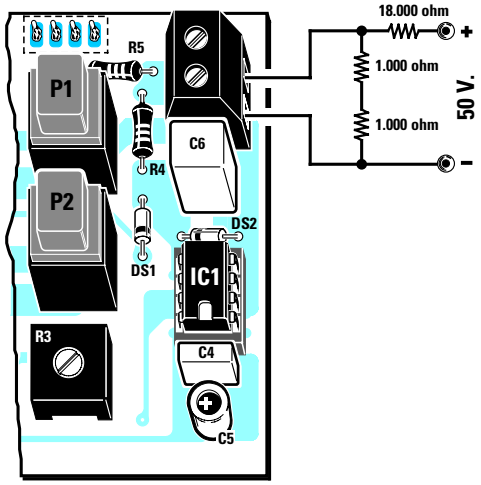


Fig.10 Volendo utilizzare la funzione Voltmetro per leggere tensioni maggiori, dovrete applicare sull'ingresso un partitore resistivo composto da tre sole resistenze.
Con questo partitore potrete leggere fino ad un valore massimo di 50 volt.

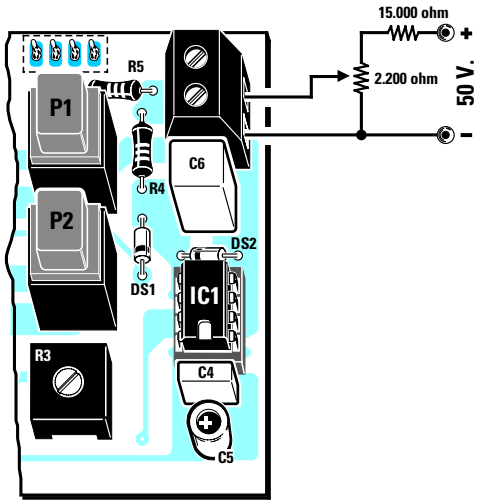


Fig.11 Utilizzando il partitore resistivo di fig.10 la lettura potrebbe non risultare precisa a causa delle "tolleranze" delle resistenze. Per risolvere questo problema, potrete utilizzare una sola resistenza ed un trimmer che andrà tarato in modo da leggere l'esatta tensione applicata sull'ingresso.

Come vi abbiamo già spiegato in precedenza, perché il display **intelligente** faccia apparire un qualsiasi **segno grafico** contenuto all'interno della sua **CGROM,** gli deve giungere un **numero** ben diverso dal **2** e dal **5** inseriti in questi due **byte,** per cui dovremo **sommare** questi due numeri alla **costante 48.**
Otterremo così:

2 + 48 = 50
5 + 48 = 53

Consultando la Tabella N.1 vedremo che il numero **50** corrisponde al **simbolo grafico 2** ed il numero **53** al **simbolo grafico 5**.

Per visualizzare la **barra** che appare sulla **seconda riga** utilizziamo i due numeri **2** e **5,** che abbiamo messo in precedenza nei due **byte,** e con questi due numeri andiamo nel file **TB_CGR02.ASM** per prelevare i **simboli grafici** che ci serviranno per accendere tutte le **caselle** interessate.
Poiché con **5 volt** si accendono **10 caselle** orizzontali, è ovvio che disponendo di una tensione di **2,5 volt** si accenderanno solo **5 caselle**.

Per provare questo voltmetro potrete procurarvi un trimmer da **10.000 ohm,** più una pila da **4,5 volt,** collegandoli come visibile in fig.9.
Ruotando il cursore del trimmer da un estremo all'altro, vedrete variare sulla **prima riga** del display il numero da **0 volt** fino ad un massimo di **4,5 volt** e sulla **seconda riga** la **barra** aumentare progressivamente fino a raggiungere un massimo di **9 quadretti**.

Volendo utilizzare questo tester per misurare tensioni superiori a **5 volt,** dovrete applicare sull'ingresso un partitore resistivo con i valori riportati in fig.10 e, in tal modo, otterrete un **fondo scala** di **50 volt**.
Non è possibile utilizzare dei partitori resistivi che diano dei valori di **fondo scala** di **10-100-200 volt,** perché il programma è impostato per leggere un **massimo** di **5 volt**.
Come noterete, tra le due cifre rimane sempre inserita la **virgola,** quindi se avete utilizzato il **partitore** di fig.10 e sull'ingresso inserite **18 volt,** sul display apparirà il numero **1,8 volt**.
Poiché le resistenze hanno una loro **tolleranza,** il **partitore** di fig.10 potrebbe non fornirvi l'esatto valore di tensione; per risolvere questo problema la soluzione migliore sarebbe quella di utilizzare lo schema riprodotto in fig.11.
Per tarare il **trimmer** potrete prendere una esatta tensione **continua** che non risulti maggiore di **50 volt** e partendo con il cursore tutto ruotato **verso massa,** lo ruoterete lentamente in senso inverso fino a leggere l'esatta tensione applicata sull'ingresso.
Ammesso di aver scelto una tensione di **28 volt,** dovrete ruotare questo trimmer fino a leggere sui display il numero **2,8 volt**.

NOTA IMPORTANTE

Non invertite la polarità della tensione sull'ingresso dell'operazionale **IC1/A,** perché sui display vedrete sempre apparire **0,0 volt**.
Non applicate sull'ingresso tensioni maggiori di **5 volt** (per pochi istanti l'integrato accetta anche **9 volt**), diversamente si potrebbe danneggiare il microprocessore **ST6/E25**.
Al programma **TESTER.ASM, che funziona** solo come voltmetro per leggere una tensione di **5 volt massimi,** non è possibile apportare alcuna modifica.
Questo programma, come già vi abbiamo accennato, è un dimostrativo che vi permetterà di vedere tutte le varie soluzioni che abbiamo adottato per far apparire un numero proporzionale alla tensione e come si utilizzano le tabelle del **TB_CGR02.ASM** e l'**A/D converter**.
Come già saprete, per poter memorizzare il programma **TESTER.ASM** all'interno del micro **ST6,** lo dovrete prima convertire in **.HEX** e per farlo dovrete digitare:

C:\>CD LX1208 poi premete Enter
C:\LX1208>A TESTER.ASM poi premete Enter

Per poter trasferire il programma convertito in **.HEX** sul micro **ST6** dovrete semplicemente scrivere:

C:\LX1208>ST6PGM poi premere Enter

e rispondere, come già sapete, a tutte le domande che appariranno sul monitor del computer.

KIT ESAURITO
vedi LX.1208/N nelle pagine seguenti

COSTO DI REALIZZAZIONE

Tutti i componenti necessari per la realizzazione di questo progetto per Display alfanumerico, che potete vedere riprodotti in fig.7 (Escluso il disco DF.1208) L.78.500

Il programma DF.1208.....L.12.000

Costo dello stampato LX.1208 .....L.10.000

Ai prezzi riportati, già comprensivi di IVA, andranno aggiunte le sole spese di spedizione a domicilio.

IL KIT LX.1208/N con il nuovo DISPLAY WH.1602A

Poiché il display alfanumerico **LM.093LN** non viene più fabbricato, ci siamo dati da fare per cercare un sostituto che lo rimpiazzasse nel kit **LX.1208**.

Dopo un'accurata ricerca, abbiamo scelto il display **WH.1602A** della **Hitachi**, che è equivalente al display **LM.093N**, eccetto che nella disposizione di alcuni piedini e nella definizione di alcuni caratteri alfanumerici (vedi tabella nelle pagine seguenti).

Proprio perché la piedinatura del display **WH.1602A** non collima perfettamente con quella del display **LM.093N** (se confrontate la fig.5 con la fig.12 vi accorgerete subito che il display della Hitachi ha un piedino in più), abbiamo pensato noi a disegnare e a fare incidere un nuovo circuito stampato al quale abbiamo dato la sigla **LX.1208/N**.

In questo modo non incontrerete alcuna difficoltà nel realizzare la scheda e soprattutto nel montare il nuovo display alfanumerico.
Inoltre, come siamo soliti fare per tutti i nostri kit, abbiamo già montato e collaudato questa scheda, e quindi possiamo assicurarvi che il circuito funziona esattamente come funzionava l'altro.

Vale la pena sottolineare che il display **WH.1602A** utilizzato, è un display **LCD alfanumerico** composto da **due righe** di **16 caratteri**.

Come vi abbiamo anticipato, rispetto al display **LM.093LN**, che aveva solo un piedino per regolare la luminosità, il display **WH.1602A** ha due controlli: con i collegamenti al positivo di alimentazione e a massa dei piedini **15-16** viene retro illuminato, mentre il trimmer **R4** collegato al piedino **3** consente di regolarne il contrasto.

Per la descrizione dello schema elettrico e per il montaggio dei componenti sul circuito stampato, rimandiamo a quanto già descritto nelle pagine precedenti, perché il funzionamento del circuito non è cambiato.
Per quanto riguarda l'**elenco componenti** e i **disegni** degli **schemi** elettrico e pratico, tenete invece presenti quelli riportati in queste pagine.

IL SET dei CARATTERI ALFANUMERICI

Nella pagina seguente abbiamo riportato anche la tabella relativa ai caratteri alfanumerici gestiti dal display **WH.1602A**, che, come vi dicevamo, non coincide perfettamente con il set di caratteri che veniva gestito dal vecchio display. Infatti se la confrontate con la **Tabella N.1** di questo articolo, vedrete che alcuni caratteri sono diversi.

In particolare, il display **WH.1602A** non ha tra i suoi caratteri le due frecce che nella **Tabella N.1** si trovano alle posizioni **126** e **127**.

Per questo motivo alcune delle istruzioni presenti nei programmi **DISP093.ASM** e **TESTER.ASM** vanno modificate, come ora vi spieghiamo.

Dopo la modifica, al posto delle frecce alle posizioni **126** e **127** della **Tabella N.1**, appariranno le frecce alle posizioni **62** e **60** della tabella del set di caratteri del display **WH.1602A**.

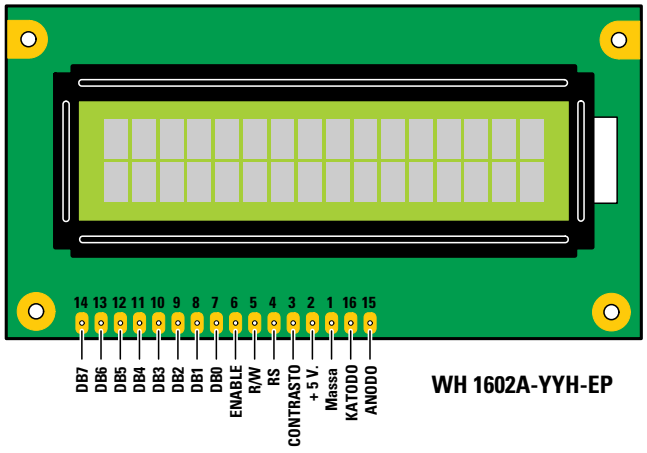


Fig.12 Connessioni del display alfanumerico WH.1602A. Il microprocessore ST6 esterno dialoga con questo display a 4+4 bit attraverso i piedini da DB4 a DB7, che fanno capo ai piedini 11-12-13-14 (vedi fig.13). I piedini da DB0 a DB3, che fanno capo ai piedini dal 7 al 10, vanno collegati a massa perché non vengono utilizzati.

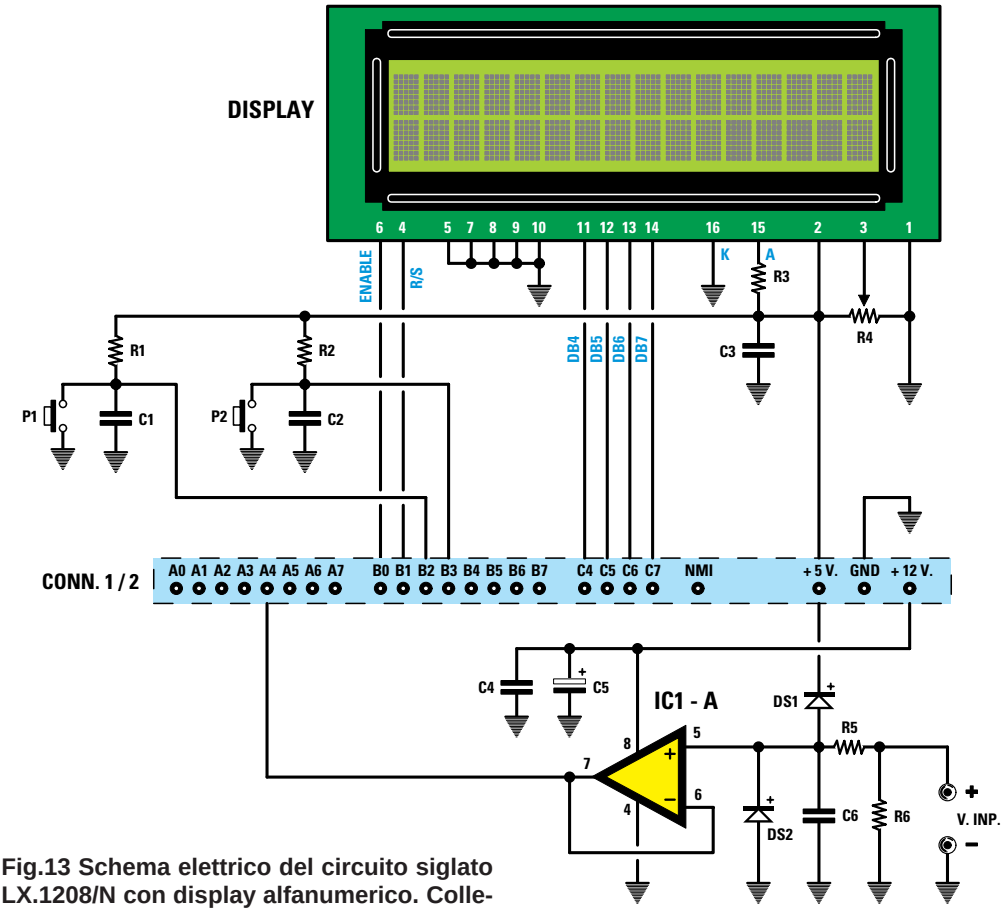
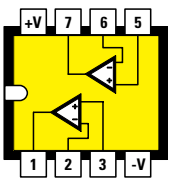


Fig.13 Schema elettrico del circuito siglato LX.1208/N con display alfanumerico. Collegando i piedini 16-15 rispettivamente a massa e al positivo di alimentazione, il display si retro illumina. Per regolare il contrasto, dovete agire sul trimmer R4.



LM 358

Fig.14 Connessioni viste da sopra dell'amplificatore operazionale LM.358, siglato IC1/A nello schema elettrico di fig.13. Come già spiegato nell'articolo, questo amplificatore non è indispensabile al funzionamento del display, ma è stato inserito per dimostrarvi come sia possibile, con opportune istruzioni di programma (vedi programma TESTER.ASM), trasformare il display in un voltmetro.

ELENCO COMPONENTI LX.1208/N

R1 = 10.000 ohm 1/4 watt
R2 = 10.000 ohm 1/4 watt
R3 = 4,7 ohm 1/2 watt
R4 = 10.000 ohm trimmer
R5 = 10.000 ohm 1/4 watt
R6 = 1 Megaohm 1/4 watt
C1 = 100.000 pF poliestere
C2 = 100.000 pF poliestere
C3 = 100.000 pF poliestere
C4 = 100.000 pF poliestere
C5 = 10 microF. elettrolitico
C6 = 1 microF. poliestere
DS1 = diodo tipo 1N.4150
DS2 = diodo tipo 1N.4150
IC1 = integrato tipo LM.358
DISPLAY = LCD tipo WH.1602A
CONN.1/2 = connettore 24 poli
P1 = pulsante
P2 = pulsante

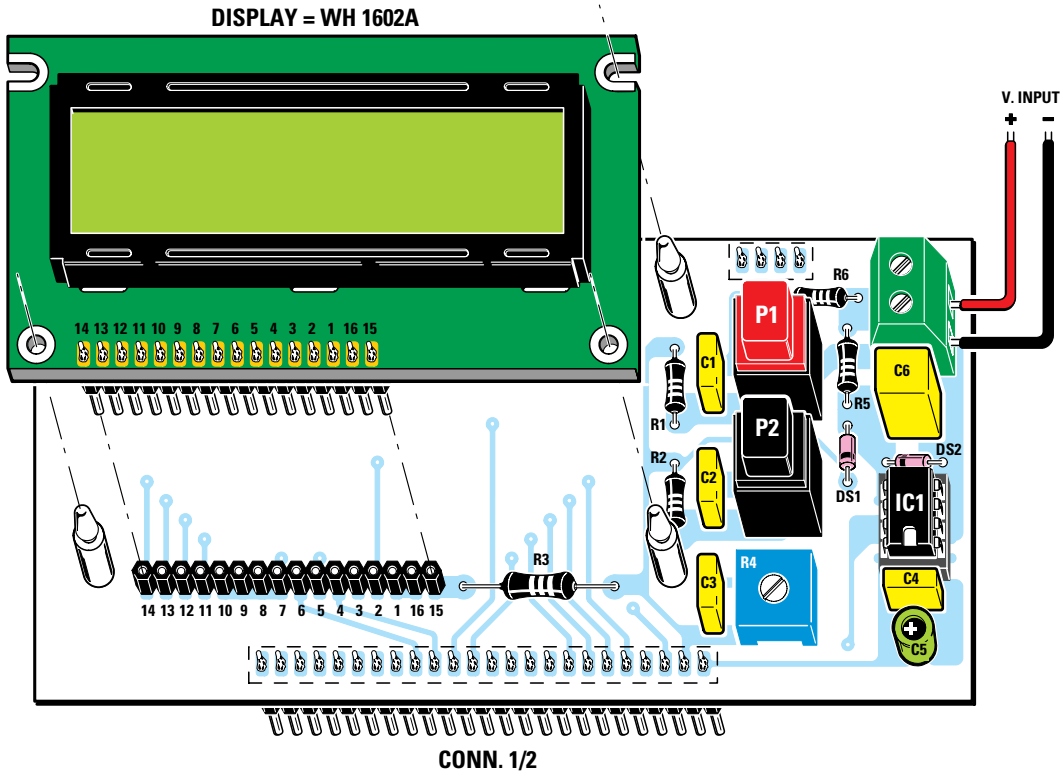


Fig.15 Schema pratico di montaggio della scheda LX.1208/N. Il montaggio di questo circuito non presenta particolari difficoltà e se rispetterete la polarità del condensatore elettrolitico C5 e dei due diodi al silicio DS1-DS2, il circuito funzionerà senza problemi. Sul lato opposto a quello visibile, vanno saldati i connettori a 4 terminali che, insieme al connettore CONN.1/2, vi serviranno per innestare la scheda nel Bus siglato LX.1202.

Nel programma **DISP093.ASM** alle righe **672 - 685 - 692**, dovete sostituire l'istruzione:

.byte 01111110b

con l'istruzione:

.byte 00111110b

Nel programma **TESTER.ASM** dovete invece andare **dopo** la riga **454** e nella tabella riportata, sostituire l'istruzione:

.byte 01111110b

con l'istruzione:

.byte 00111110b

Poi dovete lasciare invariata l'istruzione:

.byte 32,32,32

e infine sostituire l'istruzione:

.byte 01111111b

con l'istruzione:

.byte 00111100b

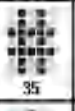
COSTO di REALIZZAZIONE

Costo dei componenti necessari per la realizzazione del kit con display alfanumerico **WH.1602A**, siglato **LX.1208/N**, visibile in fig.15, **escluso** solo il dischetto **DF.1208**
Euro 26,00

Costo del solo circuito stampato **LX.1208/N**
Euro 5,20

Costo del dischetto **DF.1208** con i programmi per display alfanumerico con micro ST6
Euro 6,20

TABELLA PER DISPLAY LCD tipo WH 1602A

0010	0011	0100	0101	0110	0111	1010	1011	1100	1101	1110	1111	→
 32	 48	 64	 80	 96	 112	 160	 176	 192	 208	 224	 240	xxxx0000
 33	 49	 65	 81	 97	 113	 161	 177	 193	 209	 225	 241	xxxx0001
 34	 50	 66	 82	 98	 114	 162	 178	 194	 210	 226	 242	xxxx0010
 35	 51	 67	 83	99	115	163	179	195	211	227	243	xxxx0011
36	52	68	84	100	116	164	180	196	212	228	244	xxxx0100
37	53	69	85	101	117	165	181	197	213	229	245	xxxx0101
38	54	70	86	102	118	166	182	198	214	230	246	xxxx0110
39	55	71	87	103	119	167	183	199	215	231	247	xxxx0111
40	56	72	88	104	120	168	184	200	216	232	248	xxxx1000
41	57	73	89	105	121	169	185	201	217	233	249	xxxx1001
42	58	74	90	106	122	170	186	202	218	234	250	xxxx1010
43	59	75	91	107	123	171	187	203	219	235	251	xxxx1011
44	60	76	92	108	124	172	188	204	220	236	252	xxxx1100
45	61	77	93	109	125	173	189	205	221	237	253	xxxx1101
46	62	78	94	110	126	174	190	206	222	238	254	xxxx1110
47	63	79	95	111	127	175	191	207	223	239	255	xxxx1111

Quando si scrivono programmi per qualsiasi microprocessore anche i più esperti possono commettere degli **errori** di **sintassi** oppure **logici**.
I primi, cioè quelli di **sintassi**, vengono già rilevati in fase di compilazione, perciò è abbastanza facile scoprirli e correggerli; i secondi, cioè quelli **logici**, possono essere scoperti solo se si dispone di un **emulatore real-time.**
Se non si possiede un **emulatore** il solo sistema per verificare che il programma risulti corretto è quello di trasferirlo in un micro **ST6 riprogrammabile**, cioè provvisto di una **finestra**.
Se, dopo averlo collocato nel circuito che dovrà gestire, si verifica che **non funziona**, bisogna ricontrollare il programma **istruzione** per **istruzione**, correggere gli errori commessi, sempre che si riesca a trovarli, ricompilare il programma con l'assembler, **cancellare** l'**ST6**, ed infine **riprogrammarlo** e "testarlo" nuovamente, perché non è detto che non vi siano altri errori che potrebbero essere sfuggiti ad un primo controllo.

Non è inoltre da escludere che nonostante la buona volontà non si riesca a capire in quale istruzione è presente l'**errore** e per scoprirlo ci potrebbe volere molto tempo e pazienza.

Con un **emulatore** risulta molto più facile ed anche meno costoso programmare qualsiasi **ST6**, perché si può controllare **passo** per **passo** ogni **istruzione** mentre viene eseguita. In questo modo è possibile capire dove e perché si è generato l'**errore**.

Ad esempio, potremmo aver scritto un programma che ad un **tempo** prefissato deve accendere una lampadina, e solo dopo aver programmato l'**ST6** ci accorgiamo che questo tempo risulta **dimezzato** o **raddoppiato** perché non abbiamo tenuto conto della frequenza del **quarzo** oppure abbiamo fatto una **somma** anziché una **moltiplicazione**.

Oppure potremmo aver scritto un programma per svolgere una semplice funzione, ad esempio:

- se suona il campanello
- vai ad aprire la porta

ma se non abbiamo tenuto presente che in casa ci sono diverse porte, quella d'ingresso, quella della cucina e quella della camera da letto e se non indichiamo nel programma quale porta **deve** essere **aperta**, si aprirà una porta qualsiasi e non quella d'ingresso come noi volevamo.

Un **emulatore** ci offre parecchi vantaggi.
Prima di tutto quello di non dover più acquistare un certo numero di **ST6 riprogrammabili** e, poiché il loro prezzo è salito alle stelle, si risparmierà una cifra considerevole.
Inoltre potendo controllare prima il programma, non si perderà del tempo per programmarli, cancellarli e riprogrammarli.

Infatti dopo aver eseguito un **test** completo sul vostro programma, se non rileverete delle anomalie potrete tranquillamente trasferirlo su un **ST6 non riprogrammabile** perché, avendolo già testato, avrete la matematica certezza che funzionerà.

SOFTWARE emulatore per

Per questi motivi i softwaristi e gli hobbisti sono alla ricerca di un **emulatore** corredato di **software** che risulti facile da usare, molto economico e che permetta un'emulazione completa ed in **tempo reale** di un micro **ST6**.

Per risolvere questo problema abbiamo acquistato tutti gli **emulatori** per **ST6** e relativo **software** che siamo riusciti a reperire sul mercato, poi ad uno ad uno li abbiamo **testati** inserendo apposta nei nostri programmi degli **errori** con diversi livelli di difficoltà per verificare con quale grado di facilità ci consentivano di individuarli.

Tra tutti quelli provati ne abbiamo trovato **uno** che, a nostro avviso, è molto **valido** ed **evoluto** sia come **hardware** sia come **software**.
Si tratta di quello della **SOFTEC** di **Azzano Decimo**, in provincia di Pordenone.
Il suo **software** è inoltre perfettamente compatibile con il sistema operativo **Windows 3.1** e precedenti ed anche con il più recente **Windows.95**, laddove molti altri software presentano invece dei problemi.
Nota: Il pacchetto **non funziona** sotto **DOS**.

Utilizzando il **software** è possibile risolvere l'**80%** dei problemi relativi alla programmazione.
Noi vi spiegheremo come deve essere usato per controllare **passo** per **passo** ogni istruzione e per scoprire tutti gli **errori logici** che potreste aver commesso nello scrivere un programma.

TESTARE i micro ST6

Con il software "emulatore" della SOFTEC riuscirete a programmare senza difficoltà tutti i micro della serie ST6210/15/20/25 perché se commetterete qualche "errore" potrete "rintracciarlo" e correggerlo. Questo software funziona sotto Windows 3.1 e sotto Windows 95.

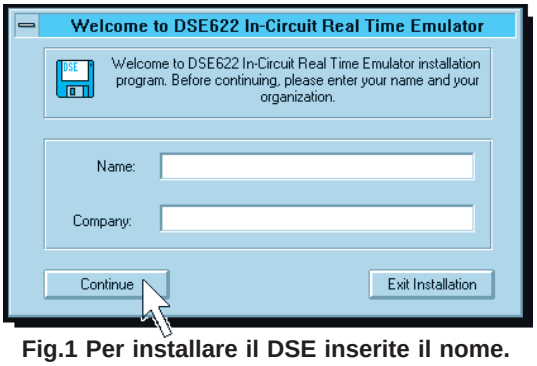


Fig.1 Per installare il DSE inserite il nome.

Fig.2 Messaggio di fine installazione.

Anche se supponiamo che tutti sappiano già come trasferire un programma da un floppy sull'Hard-Disk, riteniamo ugualmente utile ricordare queste poche istruzioni.

INSTALLARE il SOFTWARE sotto WINDOWS 3.1

Se nel vostro computer avete installato **Windows 3.1** o una versione precedente, dopo aver inserito il dischetto del **software DSE622** nel suo Drive entrate in **Program Manager**, poi portate il cursore in alto a sinistra sulla scritta **File** e cliccate, quindi andate sulla scritta **Esegui**, cliccate nuovamente e quando appare la finestra di dialogo digitate:

A:\setup poi cliccate su **OK**

In questo modo il **software DSE622** verrà trasferito dal floppy nell'Hard-Disk.

Quando appare la finestra **Name and Company** (vedi fig.1) inserite il vostro nome poi cliccate sulla scritta **Continue** per completare l'installazione.

Ad installazione avvenuta apparirà la finestra di fig.2: qui cliccate sulla scritta **OK** ed apparirà la finestra di fig.3.

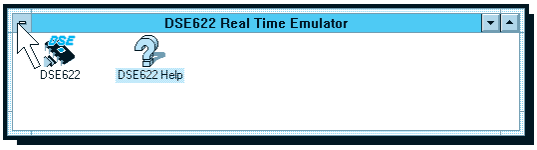


Fig.3 Finestra del Real Time Emulator.

Portate il cursore in alto a sinistra nel quadrettino con il segno **–** e cliccate per far apparire la finestra di fig.4.
Ora portate il cursore sulla scritta **Chiudi** e cliccate in modo che compaia la finestra del:

Program Manager di **Windows 3.1**

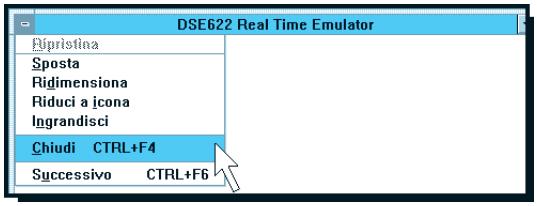


Fig.4 Per chiudere potete usare CTRL+F4.

A questo punto andate con il cursore sul simbolo di **File Manager** (vedi fig.5) e cliccate, quindi andate sul simbolo dell'**unità floppy disk A** (riportato in alto sulla sinistra) e cliccate nuovamente.
Appariranno così sulla destra del video queste tre scritte (vedi fig.6):

atest.asm
btest.asm
setup.exe

Ora dovete trasferire nell'**Hard-Disk** i due soli files **atest.asm** e **btest.asm**, perciò selezionate con il cursore la scritta **atest.asm**, andate sulla scritta **File** posta in alto sulla sinistra e cliccate in modo che appaia la finestra di fig.7.

Fig.5 Icona di File Manager in Windows.

Selezionate il comando **copia** e vedrete apparire la finestra di dialogo di fig.8, in cui dovete specificare dove volete copiare il file **atest.asm**.

Portate il cursore sulla finestra in basso e scrivete:

C:\ST6

poi andate sulla scritta **OK** e cliccate.
Tornerete così alla finestra di fig.6.

Nota: vi abbiamo fatto copiare nella **directory ST6** questo file, perché tutti i precedenti programmi inseriti nel dischetto **DF.1170**, contenenti il **software** di sviluppo dell'**ST6** della **SGS-Thomson**, prevedevano l'installazione in questa directory.

Ripetendo i passaggi appena descritti dovete ora copiare nella directory **ST6** anche il file **btest.asm**.

Al contrario **non dovete** assolutamente copiare il terzo file **setup.exe**, perché questo programma è già stato installato.

Per uscire da **File Manager** cliccate in alto a sinistra su **File** e selezionate la scritta **Esci**.

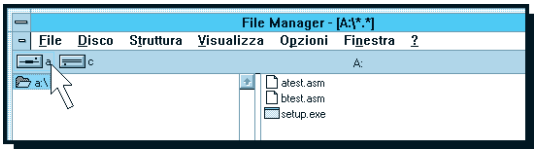


Fig.6 Contenuto del dischetto A.

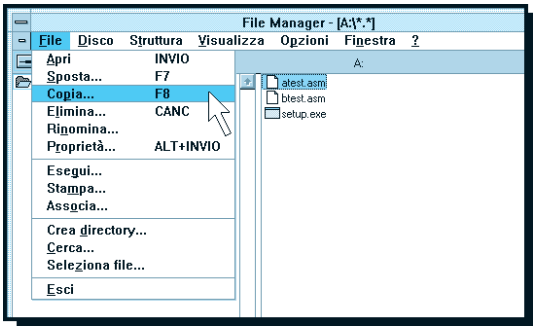


Fig.7 Cliccate sulla scritta COPIA.

Fig.8 Copiate il file ATEST.ASM in C:\ST6.

Installare il SOFTWARE sotto WINDOWS 95

Se nel vostro computer avete installato **Windows 95**, dopo aver inserito il dischetto con il **software DSE622** nel suo Drive, cliccate sulla scritta **Avvio** posta in basso, quindi andate sulla scritta **Esegui** e cliccate nuovamente.
Quando appare la finestra di dialogo dovete digitare:

A:\setup poi cliccate su **OK**

Il software **DSE622** verrà direttamente installato dal floppy nell'Hard-Disk.

Quando appare la finestra **Name and Company** (vedi fig.9) inserite il vostro nome, quindi andate sulla scritta **Continue** e cliccate.

Ad installazione avvenuta apparirà la finestra di fig.10: ora andate sulla scritta **OK** e cliccate.

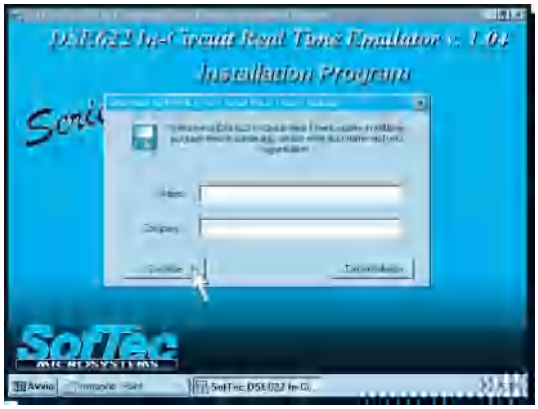


Fig.9 Quando appare la finestra dell'installazione del programma, inserite il vostro nome e cliccate su Continue.

Fig.10 Quando l'installazione sarà completata comparirà un messaggio. Per continuare cliccate su OK.

Quando appare la finestra visibile in fig.11, portate il cursore in alto a destra sull'icona con il disegno **X** e cliccate per tornare nella finestra **principale** di **Windows 95** (vedi fig.12).

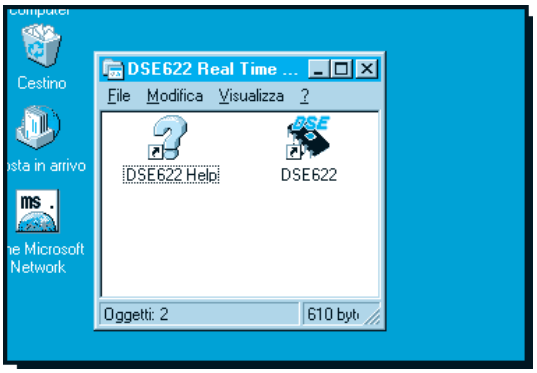


Fig.11 Per tornare alla finestra principale di Windows 95 cliccate sul simbolo X.

Fig.12 Dalla finestra principale di Windows 95 cliccate su Avvio.

Ora cliccando sulla scritta **Avvio** posta in basso attiverete un sottomenu nel quale dovrete selezionare la scritta **Programmi** e poi **Gestione risorse**. Dopo aver selezionato anche questa scritta apparirà la finestra di dialogo di fig.13.

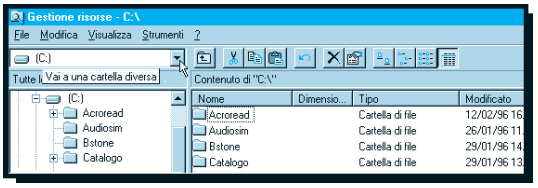


Fig.13 Cliccando sulla scritta Avvio, visibile in fig.12, e seguendo le istruzioni riportate nel testo entrate nella finestra Gestione Risorse di Windows 95.

Cliccate sulla **freccia** che appare nella piccola finestra in alto al cui interno è scritto **C:** per veder

apparire un'altra piccola finestra nella quale dovete selezionare la scritta:

Floppy da 3.5 pollici A:

Dopo aver cliccato (vedi fig.14), apparirà sulla destra il contenuto del dischetto floppy, cioè i tre files:

Atest.asm
Btest.asm
Setup.exe

A questo punto dovete trasferire nell'**Hard-Disk** solo i files **Atest.asm** e **Btest.asm**.
Per prima cosa selezionate il file **Atest.asm**, poi andate sull'icona **Copia** (vedi fig.15) e cliccate.

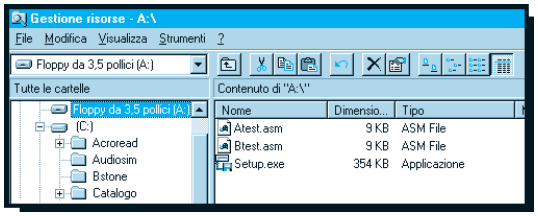


Fig.14 Contenuto del dischetto A.

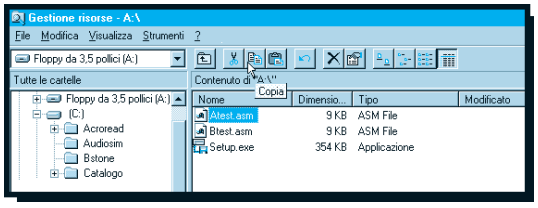


Fig.15 Andate su COPIA e cliccate.

Poiché questo file deve essere copiato nella **directory ST6**, nella finestra a sinistra (vedi fig.16) cercate con il mouse la scritta **ST6**, quindi fermate il cursore su questa riga e cliccate.

Per il momento avete **selezionato** la **directory**, ma il file non è ancora stato trasferito.

Per trasferirlo andate sull'icona **incolla** (vedi fig.17) e cliccate.

Ripetete la procedura visibile in fig.14 per copiare anche il secondo file **Btest.asm**.

Nota: non copiate il file **setup.exe** perché già installato.

Per uscire da questa finestra andate in alto a sinistra su **File**, poi portate il cursore su **Chiudi** oppure pigiate i tasti **ALT+F4**.

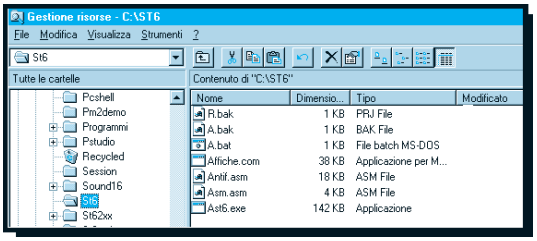


Fig.16 Selezionate la directory ST6.

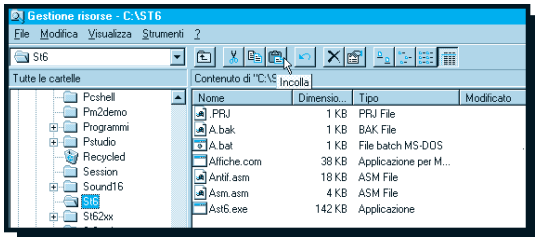


Fig.17 Copiate il file con l'icona INCOLLA.

I FILES ATEST e BTEST

I programmi che vi abbiamo fatto copiare nella directory **ST6** ci sono serviti per nostri **test** di **simulazione** e ve li proponiamo in modo che possiate imparare ad usare questo **software** di **simulazione**.
Il programma **ATEST** è stato concepito in modo da usare i quattro **piedini PA0 - PA1 - PA2 - PA3** di porta **A** come **ingressi** ed i **piedini PB0 - PB1 - PB2 - PB3** di porta **B** come **uscite**.

Applicando su uno di questi quattro **ingressi** un **livello logico 1**, tramite un interruttore o un microswitch ecc., vorremmo che apparisse sulla **corrispondente uscita** un **livello logico 1** da utilizzare per accendere un **diodo led** oppure per polarizzare la Base di un **transistor** o per eccitare un **relè** o una **sirena**.

Il programma **BTEST** differisce dal precedente solo perché vi abbiamo **inserito** alcuni **errori**, che ci permettono di mostrarvi come il **software** vi aiuti ad individuarli.

COME lavorare con il SOFTWARE DSE622

Come abbiamo già accennato, anche senza la **scheda emulatrice**, che la **SOFTEC** è in grado di fornire ad un prezzo molto competitivo, questo **software** permette di controllare in modo **trasparente** tutte le istruzioni di qualsiasi programma, aiutando così nel loro lavoro tutti i programmatori ed in particolar modo quelli che da poco hanno iniziato a programmare.
Quando si lancia il programma **DSE622**, il software testa se sull'uscita della porta **COM2** è collegata la **scheda emulatrice** della SOFTEC.

Ovviamente se non la trova segnala **"ERRORE"** (vedi fig.18), ma di questo **non dovete** assolutamente preoccuparvi.

Infatti dei tre tasti selezionabili in questa finestra, **Retry - Demo - Parameters**, basterà cliccare sul tasto **Demo** per iniziare a testare il programma.

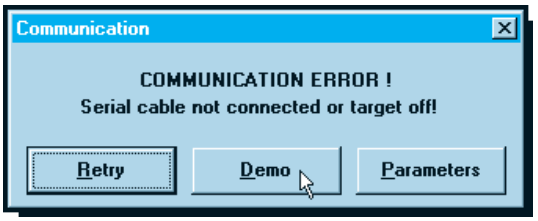


Fig.18 Se non avete la scheda dà errore.

Se cliccate sul tasto **Parameters** apparirà la finestra di fig.19, che, se un domani l'acquisterete, vi permette di indicare al computer su quale **porta seriale** avete collegato la **scheda emulatrice**.

Quando sarete in questa finestra vi consigliamo di cambiare la **velocità** di esecuzione (**Baud Rate**), e, potendo scegliere tra **9.600 - 19.200 - 28.800 - 57.600 - 115.200 Baud**, vi suggeriamo di scegliere la massima velocità, cioè **115.200 Baud**.
Digitate perciò questo numero poi cliccate sulla scritta **OK**.

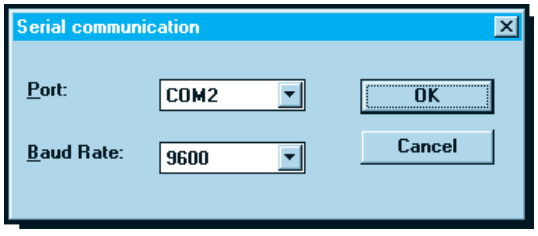


Fig.19 Scegliete la COM e i Baud Rate.

Quando appare la finestra principale del **software** di **simulazione** (vedi fig.20) si possono già iniziare a testare tutti i programmi.

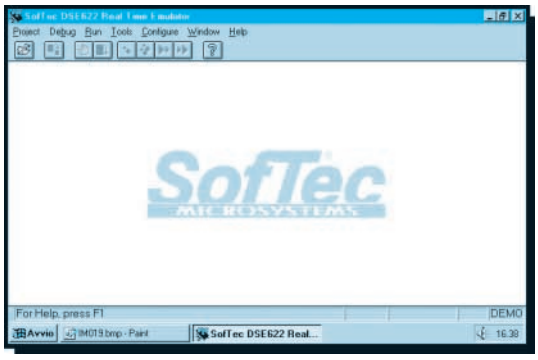


Fig.20 Finestra principale della SOFTEC.

Prima di proseguire riportiamo il significato di alcune parole che sono spesso richiamate nell'articolo.

cliccare - definiamo così l'azione che si effettua premendo il tasto del **Mouse** sulla scritta o icona indicata.

project - chiamiamo con questa parola tutti i files con estensione **.PRJ** che, oltre le caratteristiche del programma, contengono le specifiche definite con il **software simulatore** per testare il programma stesso.

source/file - chiamiamo con questa parola tutti i programmi già assemblati riconoscibili dall'estensione **.HEX**.

simulazione - chiamiamo con questa parola l'esecuzione dei **test dei programmi** con l'ausilio del **software DS622**, senza l'utilizzo dell'**Hardware** dell'**emulatore**.

variabili - chiamiamo con questa parola le definizioni degli indirizzi di memoria **Data Space**.

COME creare la LIBRERIA per l'ST6

Quando sul monitor appare la finestra con i **menu** (vedi fig.20), per creare la **libreria** cliccate sulla scritta **Configure** e vedrete apparire la piccola finestra visibile in fig.21.

- Cliccate sulla riga **Tools** per far apparire la finestra di dialogo visibile in fig.22.

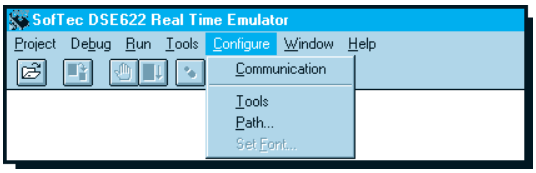


Fig.21 Finestra per creare la libreria.

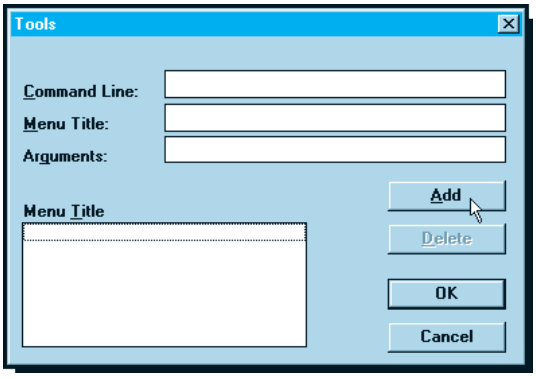


Fig.22 Finestra di dialogo TOOLS.

- Cliccate sulla scritta **Add** per far apparire la finestra di dialogo di fig.23.

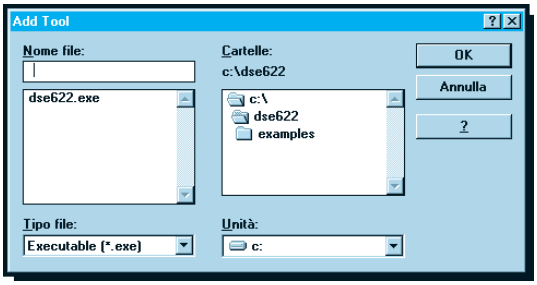


Fig.23 Finestra dialogo ADD TOOL.

- Nel riquadro a destra cliccate sulla riga **C:** e, sempre in questo riquadro, ricercate la **directory ST6**, quindi andate con il cursore su questa riga e cliccate.

- Nel riquadro a sinistra cercate il programma **ST6.EXE** (vedi fig.24) e selezionatelo, poi uscite cliccando su **OK**.
In questo modo nel riquadro **Command Line** (vedi fig.25) apparirà: **C:\ST6\ST6.EXE**

- Nella riga **Menu Title** dovete digitare: **ST6**

- Nella terza riga, **Arguments** (vedi fig.25), dovete digitare: **$2**

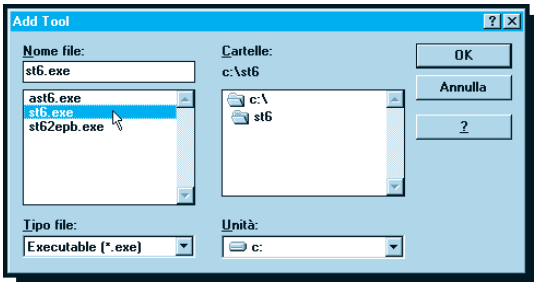


Fig.24 Selezionate il programma ST6.EXE.

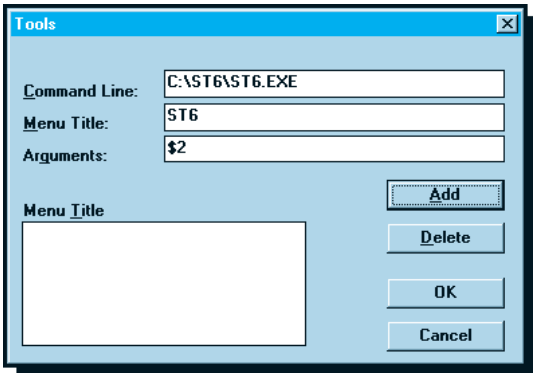


Fig.25 Digitate nelle righe quanto appare.

Definite tutte le specifiche richieste, tornate al menu principale (vedi fig.20) cliccando su **OK**.

In questo modo vi abbiamo fatto inserire nella libreria il programma **ST6\ST6.EXE**, che già da tempo vi abbiamo fornito. Con questo programma potrete assemblare, scrivere, correggere, ricercare un programma, come vi abbiamo spiegato nelle lezioni precedenti (vedi riviste N.172/173 - 174 - 175/176).

COMPILARE in ASSEMBLER il programma ATEST.ASM

Quando sul monitor appare la finestra con i **menu** (vedi fig.20) selezionate la riga **Tools**.
Nella piccola finestra che appare (vedi fig.26) cliccate sulla scritta **ST6** e così apparirà la finestra dei programmi di **sviluppo** dell'**ST6**, visibile in fig.27.

Per aprire la finestra dei files pigiate il tasto **F3** oppure andate sulla scritta **OPEN** e cliccate (vedi fig.28).
Ora cercate il programma:

atest.asm

e quando l'avete trovato cliccate sulla scritta, poi andate su **OPEN** e cliccate nuovamente.
Appariranno così sul monitor tutte le **istruzioni** di questo programma (vedi fig.29).

A questo punto portate il cursore sulla scritta **ST6** (prima riga in alto) e cliccate.
Sotto questa scritta si aprirà una finestra (vedi fig.30) con il cursore già posizionato sulla parola **Assembla** quindi cliccate.

Durante la **compilazione** in **assembler** verranno creati questi files:

ATEST.HEX
ATEST.SYM
ATEST.DSD

Per chi ancora non conoscesse il significato di queste **estensioni**, lo accenniamo qui brevemente:

.HEX - programma eseguibile in formato **Intel-Hex.**

.SYM - file contenente le definizioni delle **etichette** di **salto** ed il relativo indirizzo di memoria **Program Space**

.DSD - file contenente le definizioni delle **variabili**, le loro caratteristiche ed il relativo indirizzo di memoria **Data Space**.

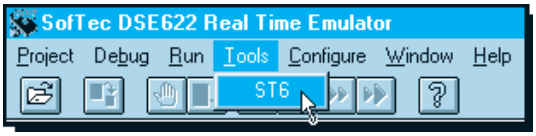


Fig.26 Cliccate sul sottomenu ST6 di Tools.

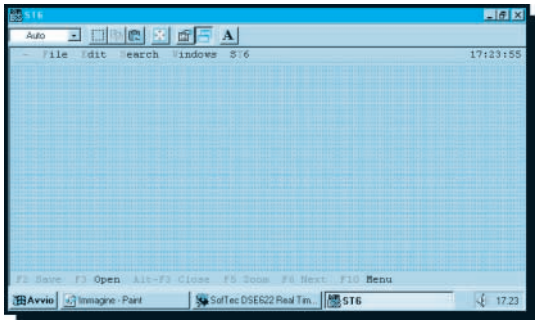


Fig.27 Finestra di sviluppo per micro ST6.

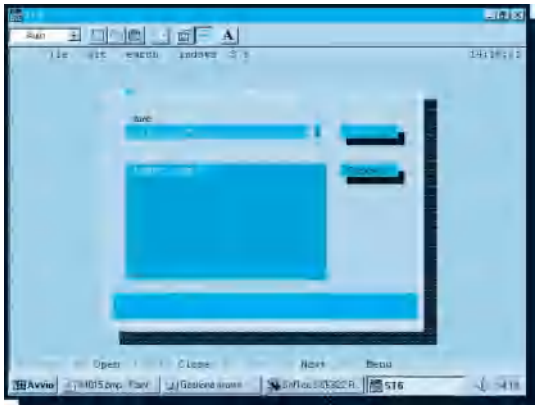


Fig.28 Aprite un programma pigiando F3.

Fig.29 Vedrete sul monitor le istruzioni.

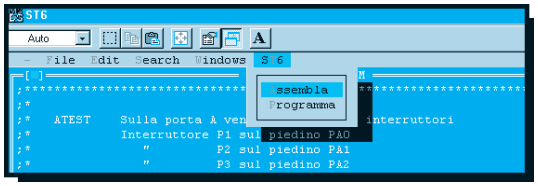


Fig.30 Cliccate sul sottomenu Assembla.

Completata la **compilazione** potete premere un qualsiasi tasto per rientrare nell'**Editor** dell'**ST6**, e quando apparirà la finestra di fig.29 cliccate sulla scritta **ALT-F3** oppure premete i tasti **ALT+F3**.
Apparirà un'altra finestra (vedi fig.27) in cui dovrete digitare **Alt X** per uscire dal programma **ST6** e rientrare automaticamente nel **software DS622**.

Quando appariranno i **menu** del **DS622** selezionate il menu **Project** e, nella finestra che appare visibile in fig.31, andate sulla scritta **New Project** e cliccate.

Questa operazione serve per creare il file con estensione **.PRJ**, che verrà utilizzato dal **simulatore** per testare il programma.

Quando appare la finestra di fig.32, digitate nella riga **Project name** il nome del file **ATEST** (non è necessario riportare dopo il nome l'estensione **.PRJ**).
Tenete presente che potete anche cambiare nome al file **project**, cioè dargli un nome differente dal **source file**.
In altre parole potrete ad esempio cambiare il nome **ATEST** in **BAUBAU** o **PLUTO**, ma se volete evitare che un domani non vi ricordiate più quale nome avevate scelto, vi consigliamo di mantenere lo stesso nome del programma assemblato, cioè nel nostro caso **ATEST**.
Dopo aver digitato il **nome** portate il cursore sulla parola **Create** (vedi fig.33) e cliccate.

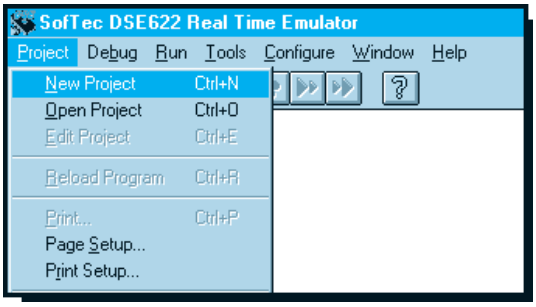


Fig.31 Comando per creare il file .PRJ.

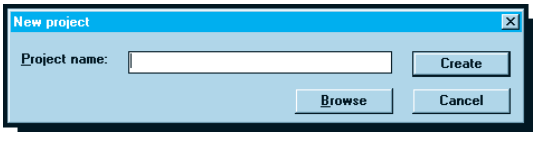


Fig.32 Digitate il nome del project.

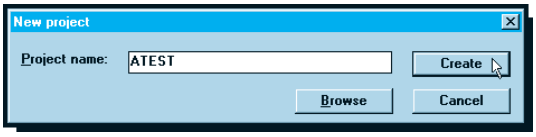


Fig.33 Scegliete Create e vedrete la fig.34.

Apparirà la finestra di dialogo **Edit Project** (vedi fig.34) in cui è molto importante inserire le specifiche richieste **senza** commettere **errori**.

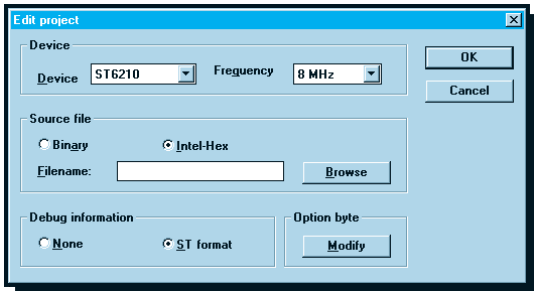


Fig.34 Digitate le specifiche richieste.

Portate il cursore nella finestra **Device** e cliccando la "freccia in giù" cercate la **sigla** del micro **ST6** che volete utilizzare. Ammesso che questo sia un **ST6210** andate sulla riga **ST6210** (vedi fig.35) e cliccate.

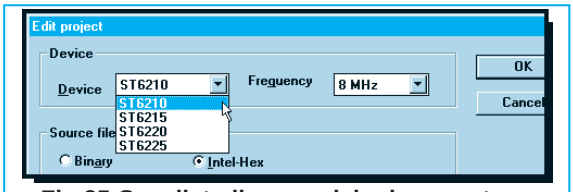


Fig.35 Scegliete il nome del micro usato.

Ora portate il cursore nella finestra **Frequency** (vedi fig.36) dove potete selezionare la **frequenza** del **quarzo** utilizzato per il **clock** del micro scegliendo tra **8 - 4 - 2 - 1 MHz**.
Nota: Se non possedete la **scheda emulatrice** della **Softec** scegliete a caso una di queste quattro frequenze.

Per completare i dati da inserire in questa finestra cliccate nel cerchietto accanto alla scritta **Intel-Hex** (vedi fig.37) cosicché apparirà un **punto**.

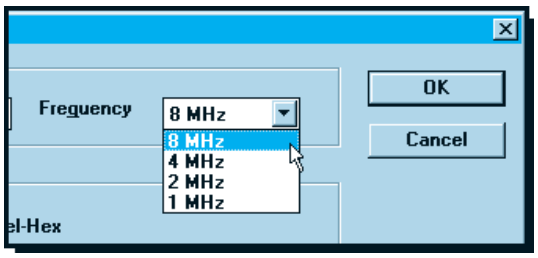


Fig.36 Scegliete la frequenza del quarzo.

Fig.37 Con Intel-Hex vedrete i file .EXE.

In questo modo indicate al programma di visualizzare i soli files con estensione **.HEX**.

A questo punto cliccate sulla scritta **Browse** in modo che appaia la finestra di dialogo di fig.38.

Nel riquadro posto a destra cliccate sulla riga **C:**, poi cercate la scritta **C:\ST6** e quando l'avete trovata selezionatela cliccando.
Nel riquadro a sinistra appariranno tutti i files con estensione **.HEX**.

Cercate tra questi la scritta **ATEST.HEX** (vedi fig.39), cliccate su questa riga, poi uscite cliccando su **OK**. In questo modo nel riquadro **Filename** (vedi fig.40) apparirà la scritta:

C:\ST6\ATEST. HEX

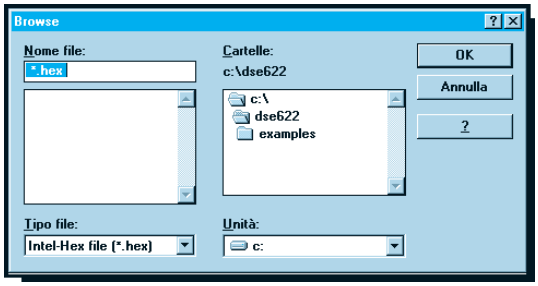


Fig.38 Finestra di dialogo Browse.

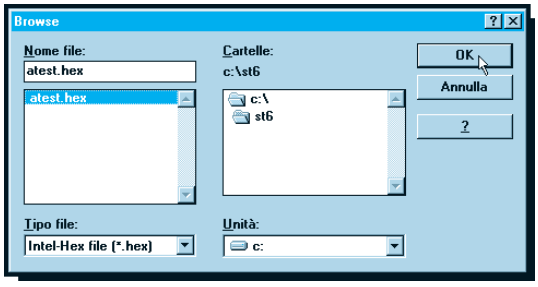


Fig.39 Quando siete nella finestra Browse, selezionate il file ATEST.EXE.

Fig.40 Nel riga Filename apparirà il nome del file selezionato nella finestra Browse.

Ora potete passare al **Debug Information** (vedi fig.34) e se all'interno del **cerchietto** posto a sinistra della scritta **ST format** trovate già un **punto** (vedi fig.42) non dovrete cliccare.

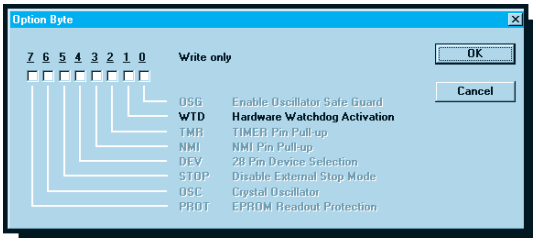


Fig.41 Se siete in simulazione, il WTD, Hardware Watchdog Activazion, non serve.

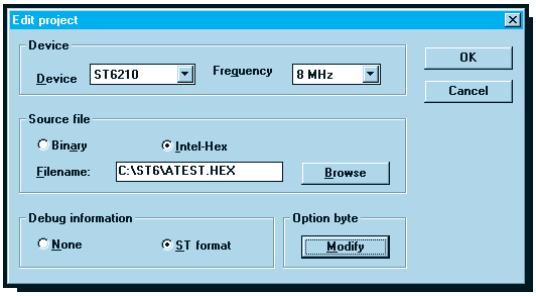


Fig.42 Per creare un file con estensione .PRJ, cliccate sulla scritta OK.

Fig.43 Nella videata principale del DSE622 vengono attivate tre finestre.

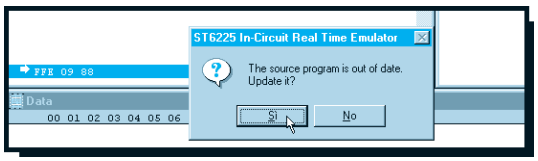


Fig.44 Dalla finestra di fig.27 si passa con ALT+X a questa finestra. Per andare nella finestra di fig.45 cliccate su SI.

Se per errore cliccate all'interno del **cerchietto** posto a sinistra della scritta **None** facendo apparire un **punto**, durante l'esecuzione del programma **non riuscirete** a vedere i **nomi delle variabili** o delle **etichette utilizzate**, quindi vi sarà molto più difficile controllare **passo** per **passo** il programma.

Ora cliccate sulla scritta **Modify** per far apparire la finestra di fig.41 che ha in evidenza questa scritta:

WTD = Hardware Watchdog Activazion

Siccome ci troviamo in **simulazione**, questa selezione non serve perciò portate il cursore su **OK** e cliccate per far apparire la finestra di fig.42.

A questo punto cliccando sulla scritta **OK** verrà creato il **Project ATEST.PRJ** nella directory **DES622** e sarete pronti per testare il programma (vedi fig.45).

Se sul video appare invece una di queste scritte:

source file not found
symbol table file not found
debugger file not found

potreste aver involontariamente **cancellato** dei files di compilazione. Per rigenerarli dovete cliccare su **OK** per far riapparire la finestra di fig.43.

Ora dovete cliccare nuovamente sulla scritta **Tools** e ripetere tutte le operazioni visualizzate dalla fig.20 alla fig.29 e vi ritroverete nella finestra visibile in fig.27.
Per uscire digitate **Alt+X** e sul video vedrete apparire la finestra di fig.44.
Cliccando su **SI** potrete vedere la finestra di fig.45 con tutti i dati e le istruzioni corrette.

Fig.45 La finestra per testare il programma appare sullo schermo così suddivisa.

Se invece appare la scritta:

C:\ST6\ATEST.HEX emulation buffer overflow

significa che avete scritto il programma per un tipo di micro **ST6** diverso da quello selezionato nella riga del **Device** (vedi fig.35).
Ad esempio potreste aver scritto un programma per il micro **ST62/10** ed aver inserito nel **Device** il micro **ST62/25** o viceversa.
In presenza di questo errore cliccate su **OK** e, quando appare la finestra di fig.43, selezionate il menu **Project** e cliccate su **Edit project** (vedi fig.46).
Vi ritroverete nella finestra di fig.35 dove potrete correggere la scritta corrispondente alla riga **Device** digitando la sigla corretta del micro utilizzato.
Per uscire cliccate su **OK**.

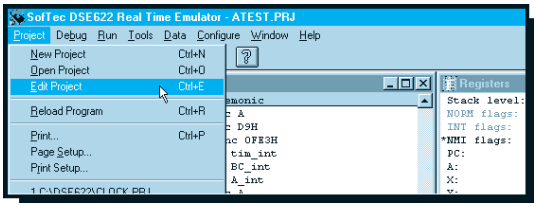


Fig.46 Finestra per correggere la fig.35.

Quando apparirà la finestra di fig.45 sarete pronti per effettuare la **simulazione** del programma **A-TEST.HEX** utilizzando il file **Projet ATEST.PRJ**.

Prima di proseguire è necessario che vi spieghiamo cosa contengono le **3 finestre** visibili in fig.45.

Nella finestra **Disassembler**, sotto le due colonne **Label** e **Mnemonic**, avete le istruzioni del programma da **testare** in formato leggibile.
Nelle colonne **Add** e **Opcode** appaiono le medesime istruzioni in formato **Intel.Hex**.

Nella finestra **Register** appaiono tutti i **registri**, lo **stack level**, gli stati dei tre **flags** ed il valore del **Program Counter**.

Nella finestra **Data** appare il contenuto del **Data Memory Space** del micro, cioè il contenuto delle **variabili**, dei **registri** e della **Data Rom Windows** definiti in questo programma.

Avendo sottocchio tutte queste finestre sarete in grado di controllare **passo x passo** in **simulazione** tutti i vostri programmi.
Prima di iniziare il controllo, è a nostro parere necessario configurare ulteriormente il **software DS622** così da vedere sul monitor in **tempo reale** anche altre funzioni che potrebbero risultarvi molto utili.

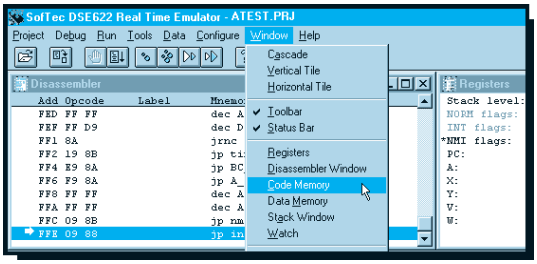


Fig.47 Aprite la finestra Code-Memory.

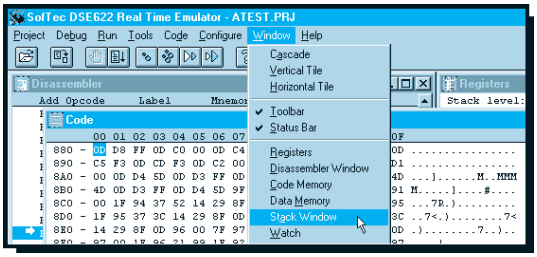


Fig.48 Aprite la finestra Stack-Window.

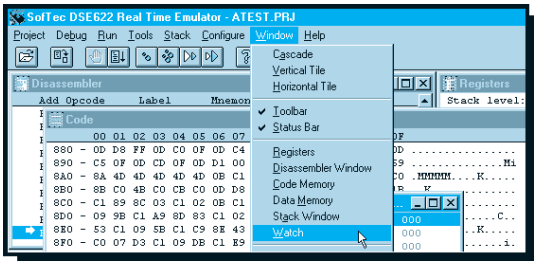


Fig.49 Aprite la finestra Watch.

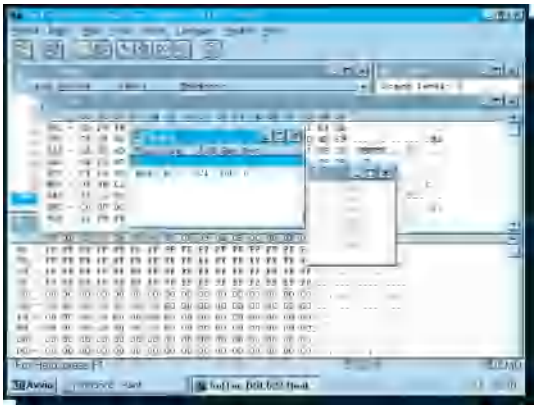


Fig.50 Sul video appaiono altre finestre.

Perciò dopo aver portato il cursore sulla scritta **Window** posta in alto (vedi fig.47) cliccate e nella finestra che appare selezionate la riga **Code Memory**.

Cliccate nuovamente sulla scritta **Window** e selezionate la scritta **Stack Windows** (vedi fig.48).

Ritornate nuovamente a cliccare sulla scritta **Windows** e selezionate la riga **Watch** (vedi fig.49).
Il sottomenu di Window sparirà.

In questo modo alle finestre che già apparivano nella fig.45 si aggiungono **3 supplementari finestre** operative (vedi fig.50), e cioè:

Stack
Code Memory
Watch

Vi consigliamo di rimpicciolire e spostare le finestre (vedi l'esempio riportato in fig.51) in modo da avere sempre sottocchio tutte le specifiche relative alla programmazione di qualsiasi micro **ST6**.

Fig.51 Vi suggeriamo di rimpicciolire e spostare tutte le finestre come visibile in questa figura.

Dopo aver posizionato le finestre nei punti che ritenete più opportuni, tutte le volte che richiamerete il file **ATEST.PRJ** o altri files con la stessa estensione, le finestre riappariranno dove le avevate posizionate.

La finestra **Stack** permette di visualizzare in **tempo reale** il valore dei suoi **6 registri**.

Se nel corso del programma viene eseguita l'istruzione **Call** (vedi rivista **N.174**) o viene attivato un **Interrupt** (vedi rivista **N.175/176**), il contenuto dei primi **5 registri** di **Stack** viene immediatamente **shiftato** di un livello **superiore** , vale a dire che il contenuto del 5° registro viene passato al 6°, il contenuto del 4° registro viene passato al 5° e così via.

A questo punto nel **1° registro** di **Stack** viene memorizzato il contenuto del **Program Counter**, cioè l'indirizzo di ritorno della subroutine richiamata nella **Call** (istruzione **Ret**) o nell'**Interrupt** (istruzione **Reti**), come appare visibile in fig.89.

In questo modo è possibile eseguire più **subroutine**, una all'interno dell'altra, fino ad un massimo di **6**.

La finestra **Code Memory** permette di visualizzare in **esadecimale** il contenuto della **Program Memory Space**, cioè la memoria del micro in cui sono contenute le istruzioni del programma sotto **test**.

La finestra **Watch**, che inizialmente è **vuota**, serve per inserire, come poi vi spiegheremo, le **variabili** delle quali ci interessa controllare il contenuto per vedere come questo si modifica durante l'esecuzione del programma.

Se di queste tre finestre volete che ne appaia **una sola** o **due**, dovete annullare una delle operazioni riportate nelle figg.47-48-49.

ESEMPI di EMULAZIONE

Le prime volte che userete il **software simulatore DSE622** vi consigliamo di **stampare** il listato del programma che volete **testare** (nel nostro esempio **ATEST.ASM**) per poter confrontare le istruzioni stampate con quelle che appariranno sul monitor.
Se avete spento e riacceso il computer, per richiamare il programma **ATEST** già creato come file **project**, che ora si chiamerà quindi **ATEST.PRJ**, dovrete cliccare sull'icona **apri file** (vedi fig.52).

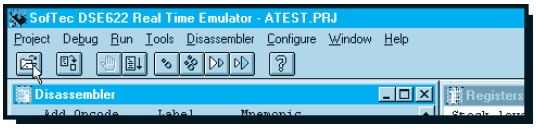


Fig.52 Cliccando sull'icona Apri File potrete aprire il file ATEST.PRJ.

Nella finestra di dialogo che appare, sotto il **Nome file** ci sarà la scritta ***.PRJ** e nella finestra sottostante il solo file **ATEST.PRJ**, perché per ora è stato creato un solo **project**.
Man mano che creerete files con estensione **.PRJ** troverete tutti i loro nomi in questa finestra.

Cliccate sul nome del file desiderato quindi uscite da questa finestra cliccando su **OK**. Apparirà la finestra visibile in fig. 51.

Sull'ultima riga visibile della finestra **Disassembler** (vedi fig.53) potete vedere una riga blu evidenziata da una **freccia rossa**.

Nota: la **freccia rossa** mette in evidenza la prima istruzione che il programma **eseguirà**.

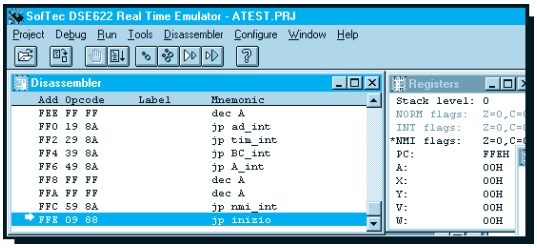


Fig.53 La freccia mostra da dove si parte.

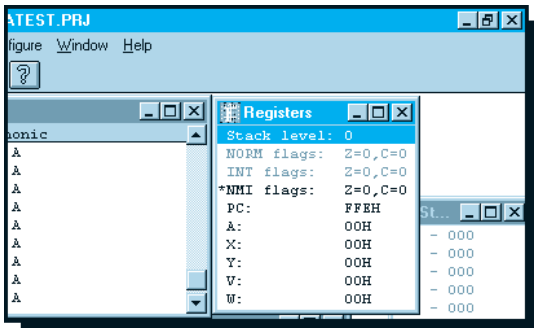


Fig.54 Finestra Registers.

Fig.55 Icona per avanzare passo/passo.

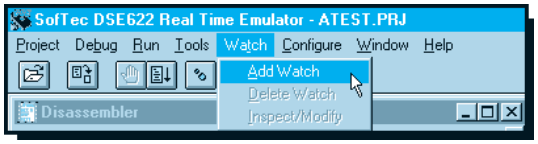


Fig.56 Aprite la fig.57 con Add Watch.

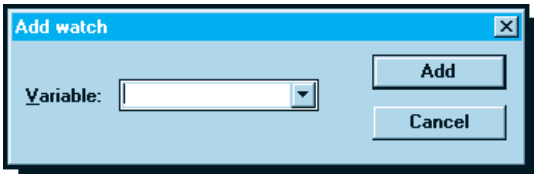


Fig.57 Per le variabili cliccate sulla freccia.

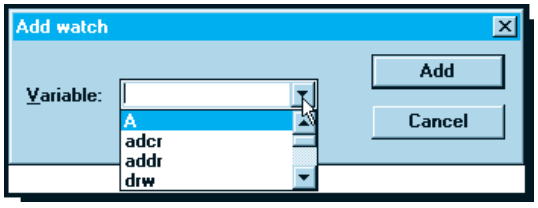


Fig.58 Selezionate la variabile che volete.

Le righe sopra quella evidenziata in blu sono quelle che abbiamo definito nel programma **ATEST** per i **vettori di interrupts**.

Nella finestra **Register** di fig.54 appare lo **Stack level**, i tre **Flags NORM - INT - NMI** tutti a **livello logico 0**, ed il **PC** (Program Counter) posizionato sul valore esadecimale **FFEH** (come già sapete la lettera **H** significa esadecimale), che corrisponde all'indirizzo della prima istruzione eseguibile (vedi la finestra **Disassembler** di fig.53).

Per far partire il programma in modo che ogni istruzione avanzi **passo x passo** dovete portare il cursore sull'icona di fig.55 e cliccare.

Ogni volta che cliccherete su questa icona il programma eseguirà una **sola istruzione** e questo vi consentirà di controllarlo riga per riga.

Come vi spiegheremo tra poco, è possibile eseguire anche più istruzioni per ogni **passo** oppure lanciare un'esecuzione in **automatico**.

La **prima** istruzione che il programma **ATEST** esegue è quella riportata all'indirizzo **FFEH**, come appare in fig.53.

INSERIRE una VARIABILE nella finestra WATCH

Poiché potrebbe risultare utile controllare lo **stato** dei piedini della **porta A** di entrata (input) e quello dei piedini della **porta B** di uscita (output), scrivendo nella finestra **Watch** le definizioni di queste **variabili** potrete averle sempre sottocchio.
In questo modo sarà più facile vedere come cambiano man mano che fate avanzare il programma.

Per visualizzare le due porte nella finestra **Watch** dovete eseguire queste semplici operazioni:

- Andate con il cursore nella finestra **Watch** e cliccate per **evidenziare** questa finestra.

- Come potete notare, nella barra dei menu posta in alto, tra le scritte **Tools** e **Configure**, appare la scritta **Watch** che vi permette di accedere ad un sottomenu dedicato a questa finestra.

Tra le scritte **Tools** e **Configure** appare infatti di volta in volta il menu relativo alla finestra posta in **primo piano**.

- Cliccate sulla scritta **Watch** e selezionate la scritta **Add Watch** (vedi fig.56) in modo da far apparire la finestra di dialogo visibile in fig.57.

- Andate con il cursore sulla **freccia** posta a destra della scritta **Variable** e vedrete apparire in **ordine alfabetico** l'elenco delle **variabili** presenti nel programma **ATEST** (vedi fig.58).

Cliccate su **port_a**, poi andate su **ADD** e qui cliccate. In questo modo nella finestra **Watch** apparirà la **variabile port_a** con l'indirizzo **ADD** di **Data Space**, il valore **esadecimale - Hex** ed il valore **decimale - Dec** (vedi fig.59).

Poiché ci interessa controllare anche i valori della **porta B**, dovrete ripetere tutte le operazione riportate nelle figg.56-57-58, quindi selezionare la variabile **port_b**.
Cliccando poi su **ADD** nella finestra **Watch** appariranno i dati della **port_b** (vedi fig.60).

COME INSERIRE un BREAKPOINT

Un'altra operazione che dovete imparare è come **attivare** un **breakpoint**.
Attivare un **breakpoint** significa mettere un **punto di blocco** ad un'istruzione del programma in modo che durante la simulazione si fermi su quella riga.

Ammesso che vogliate inserire un **breakpoint** sull'ultima riga **in basso** in cui è riportata l'istruzione (vedi fig.61):

898 0D D4 00 ldi tscr,00H

dovrete portare il cursore su questa riga e **cliccare due volte**. Vedrete così apparire sul monitor la finestra di comando visibile in fig.62.
Cliccate sulla scritta **Toggle Breakpoint** e a sinistra della **riga selezionata** comparirà un **punto esclamativo** che vi segnala che su quella riga è stato attivato il **breakpoint** (vedi fig.63).

COME usare il BREAKPOINT

Come già vi abbiamo accennato, il **breakpoint** serve per fermare il programma sulla riga di istruzione che avete **marcato** ogni volta che verrà lanciata una simulazione in modo **automatico** o a **passi multipli**.
Potete mettere quanti **breakpoint** volete, cioè **3 - 5 - 14 - 20 ecc**.
È sottinteso che quando il programma si sarà fermato su un **breakpoint** per proseguire dovrete nuovamente cliccare sull'icona **passo** per **passo** o a **passi multipli** oppure sull'esecuzione **automatica**.
Per togliere il **breakpoint** (ma ora **non toglietelo** perché ve lo faremo usare per fare un po' di pratica) dovrete andare sulla riga interessata e cliccare **due volte**, poi nella finestra che appare dovre-

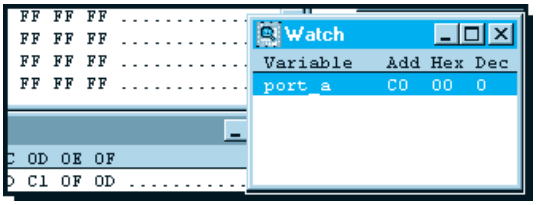


Fig.59 Esempio della Variabile port_a.

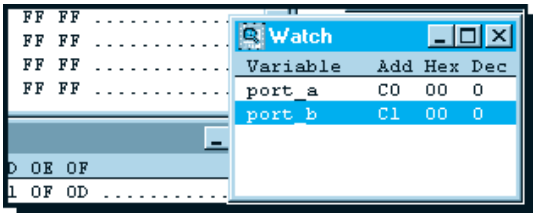


Fig.60 Esempio della Variabile port_b.

Fig.61 Finestre per testare il programma.

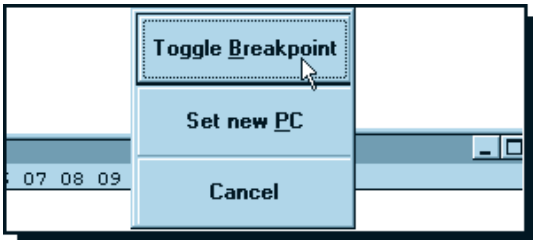


Fig.62 Per attivare un Breakpoint.

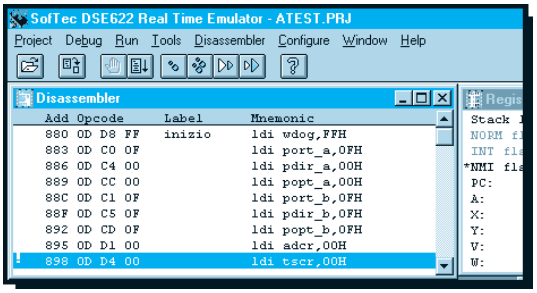


Fig.63 Dove c'è ! è attivato un Breakpoint.

te andare sulla scritta **Toggle Breakpoint** (vedi fig.62) e cliccare.
in questo modo il breakpoint verrà eliminato dalla riga in cui in precedenza era stato inserito.

Se per errore andate sulla scritta **Cancell** e cliccate **non toglierete** il breakpoint, ma farete solo sparire la finestra di fig.62.

ESECUZIONE in AUTOMATICO

Per far avanzare in modo **automatico** le istruzioni senza cliccare tutte le volte il tasto del **passo-passo** riportato in fig.55, dovete portare il cursore sull'icona visibile in fig.64, cioè sull'immagine di una pagina con una freccia rivolta verso il basso.

Fig.64 Icona per avanzare in automatico.

Cliccando su questa icona ottenete un'esecuzione sequenziale ed automatica di tutte le istruzioni, che si fermerà solo sull'istruzione in cui avete inserito il **Breakpoint**.

Quando viene eseguita l'istruzione:

883 0D C0 0F ldi port_a,0FH

se guardate nella finestra **Watch** al contenuto della **port_a** sotto la colonna **Dec.** trovate il numero **15**.

Quando viene eseguita l'istruzione:

88C 0D C1 0F ldi port_b,0FH

se guardate nella finestra **Watch** al contenuto della **port_b** sotto la colonna **Dec.** trovate il numero **15**.
Infatti sempre nella finestra **Watch** al numero esadecimale **0F** corrisponde il numero **decimale 15**.
Se non sapete ancora convertire un numero **decimale** in un numero **binario** vi consigliamo di andare a **pag.381** del nostro **Handbook** (se ne siete sprovvisti potete richiedercelo) dove troverete:

0 0 0 0 - 1 1 1 1

Per avere una riprova **visiva** di queste condizioni logiche dovete andare nella finestra **Data**.
Questa finestra riporta nella prima colonna gli indirizzi di memoria e nel righello in alto in grigio i valori **esadecimali**:

00-01-02-03-04 —— 09-0A-0B-0C-0D-0E-0F

Poiché l'indirizzo di **port_b** è **C1**, nella prima colonna dovrete cercare l'indirizzo di memoria **C0**, poi, prendendo come riferimento il righello in grigio, scendete dal valore esadecimale **01** fino ad incontrare la riga **C0**, come si farebbe con una **Tavola Pitagorica**, e così troverete il valore **0F**.

Portate il cursore su **0F** poi cliccate **2 volte** e vedrete apparire la finestra di dialogo **Edit data** riportata in fig.65.
Cliccando sulla scritta **Bits** apparirà sullo schermo la finestra di dialogo di **Port B Data Register** di fig.66.

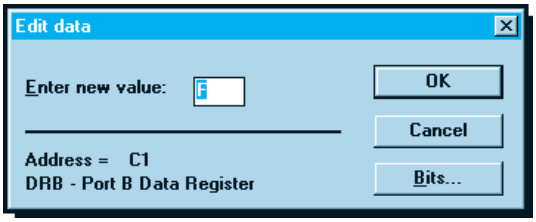


Fig.65 Finestra di dialogo Edit data.

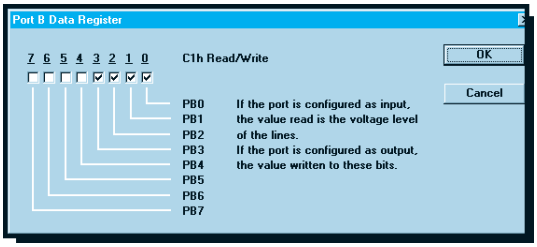


Fig.66 Finestra di Port B Data Register.

In **orizzontale** potete leggere i numeri **7 - 6 - 5 - 4 - 3 - 2 - 1 - 0**, che corrispondono ai piedini della porta B. Sotto questi numeri ci sono delle **caselle** che possono contenere una **V** oppure risultare **vuote**.
Nel nostro caso la **V** è presente solo sulle prime quattro caselle **3 - 2 - 1 - 0**, mentre nelle altre quattro caselle **7 - 6 - 5 - 4** non appare nulla.

Le caselle con la **V** sono quelle che si trovano a **livello logico 1**, cioè sui piedini corrispondenti risulta presente una tensione **positiva**, ed ovviamente quelle **senza** la V sono a **livello logico 0**.
Come già saprete il numero **decimale 15** corrisponde a **0 0 0 0 - 1 1 1 1** in **binario**.

Nella colonna **verticale** troverete gli **otto** piedini della porta **B** siglati **PB0 - PB1 - PB2 - PB3 - PB4 - PB5 - PB6 - PB7**.
Da questa finestra di dialogo potete dunque sapere quale **condizione logica** è presente sugli **otto piedini**.

Nel nostro esempio:

PB7 - PB6 - PB5 - PB4 = livello logico 0
PB3 - PB2 - PB1 - PB0 = livello logico 1

Dopo questa verifica potete cliccare su **OK**, poi cliccate ancora su **OK** nella successiva finestra e tornerete nella finestra **Data** di fig.67.

IMPORTANTE

Durante l'esecuzione del programma tutte le **quattro uscite** si portano a **livello logico 0** fino a quando uno dei **quattro ingressi** non viene collocato a **livello logico 1**.
Potendo vedere nella finestra di fig.66 le **condizioni logiche** presenti sulle **porte**, vi accorgerete subito se nel programma è stato commesso un **errore**.
Ammesso infatti che nel **piedino PB6** di porta B debba risultare presente un **livello logico 1** e non un **livello logico 0** e nel **piedino PB0** i porta B un **livello logico 0** e non un **livello logico 1**, potrete subito vedere la **situazione** sui piedini della porta.
La finestra di fig.66 non solo vi permette di vedere i **livelli logici** sui piedini di **port_b**, ma anche di correggerli. Infatti se, ad esempio, provate a cliccare nella **casella 6** comparirà una **V** che vi indica che questo piedino è passato a **livello logico 1**.

Poiché il programma **ATEST** non contiene **errori** non cambiate i **livelli logici** su **PB0 - PB1 - PB2 - PB4** e se lo fate, rimettete quelli che apparivano in precedenza, diversamente andrete a modificare il corretto proseguimento del **test.**

Allo stesso modo potrete controllare il **livello logico** del registro di controllo dell'**AD/Converter** (nel programma di **ATEST** l'AD/Converter non viene usato).
Andate nella finestra **Data** (vedi fig.67) e cercate negli indirizzi di memoria il valore esadecimale **D0**.

Poiché l'indirizzo di questo registro è **D1**, prendendo come riferimento il righello in grigio, poi scendete dal valore esadecimale **01** fino ad incontrare la riga **D0** e troverete il valore **00**.

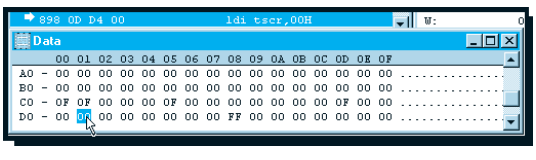


Fig.67 Per attivare la finestra di dialogo Edit Data dovete cliccare due volte su 00.

Ponendo il cursore su **00** e cliccando **due volte** vedrete apparire la finestra dell'**Edit data** di fig.68, ovviamente diversa da quella di fig.65.
Cliccando sulla scritta **Bits** apparirà la finestra di fig.69, leggermente diversa da quella di fig.66.

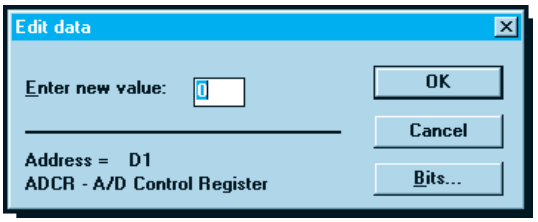


Fig.68 Finestra di dialogo Edit data.

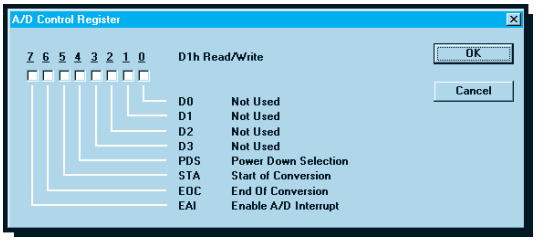


Fig.69 Dalla finestra di fig.68 cliccate sulla scritta Bits per aprire la finestra dell'A/D Control Register.

Dopo avervi spiegato come sia possibile controllare visivamente i **livelli logici** presenti sui **piedini** del micro, potrete divertirvi a vedere il contenuto delle **variabili** e dei **registri** che non vi abbiamo citato. In questo modo farete un po' di pratica che vi servirà in futuro per scrivere correttamente i vostri programmi.

ESECUZIONE a PASSI MULTIPLI

Per la funzione **passi multipli** sarebbe consigliabile togliere tutti i **breakpoints**, in ogni caso anche se li lascerete potrete ugualmente eseguire questa funzione.
L'esecuzione a **passi multipli** vi dà la possibilità di eseguire in modo **automatico** un numero di istruzioni che voi stessi potrete definire.

Ad esempio, potrebbe risultarvi comodo far eseguire al programma **5 - 8 - 10 istruzioni** di seguito prima i fermarvi per **controllare** i dati**.**
Stabilito il numero di istruzioni da eseguire di seguito, tutte le volte che cliccherete sull'icona di fig.70 verrà eseguito il numero di istruzioni che avete prefissato.

Attualmente il **software** è predefinito per fare **passi** di **2 sole istruzioni**, quindi ammesso che desideriate fare dei passi di **5 istruzioni** dovrete procedere come segue.

- Portate il cursore sulla scritta **Run** visibile in fig.71 e cliccate e nella finestra che appare andate sulla scritta **Multiple Step Value** e cliccate.

Fig.70 Icona per fare passi multipli.

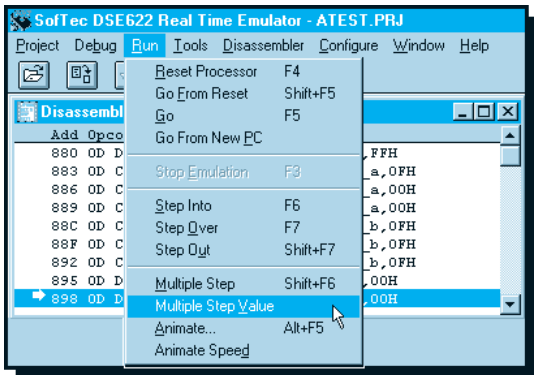


Fig.71 Finestra per variare i passi/multipli.

- Nella finestra di dialogo che appare (vedi fig.72) andate nella casella **Value** e sostituite il numero **2** con il **5**, poi andate su **OK** e cliccate.

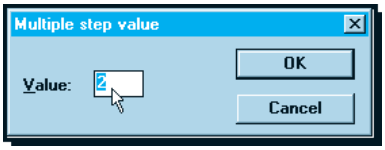


Fig.72 Digitate il numero dei passi multipli.

- Tutte le volte che cliccherete sull'icona in cui sono disegnate due orme (vedi fig.73), il programma avanzerà di **5 istruzioni**.
Cliccando nuovamente sull'icona, il programma avanzerà di altre **5 istruzioni**.

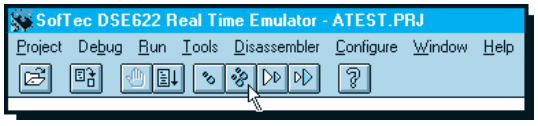


Fig.73 Icona per i passi multipli.

FUNZIONE DEBUG

La funzione **debug** è molto utile per vedere la **mappa** del micro utilizzato e su quali piedini sono posizionate le **porte A - B - C**, inoltre potete vedere quali **livelli logici** sono presenti sui piedini **d'ingresso** o di **uscita** durante l'esecuzione del programma.

Per entrare nella funzione **debug** cliccate sulla scritta **Debug**, visibile nella fascia superiore del menu, e quando appare la maschera di fig.74 cliccate nella riga **Test I/O**.
Apparirà così la finestra di fig.75.
In alto potete leggere la sigla dell'integrato prescelto, nel nostro caso **ST6210 - ST6220**, in basso a sinistra potete vedere le connessioni del micro e su quali **piedini** sono posizionate le **porte A** e **B**, infine sul lato destro è visibile la **mappa** di configurazione logica di queste due porte.

Per la **porta A**, che dispone di soli 4 piedini **PA0 - PA1 - PA2 - PA3**, troverete a destra **4** caselle **grigie** (questo perché i piedini della porta **A** sono solo 4) e **4** caselle indicate **3 - 2 - 1 - 0** che possono essere **vuote** o contrassegnate da una **V**.

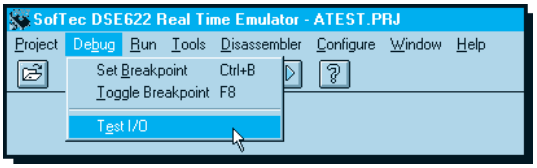


Fig.74 Cliccate su Test I/O per la fig.75.

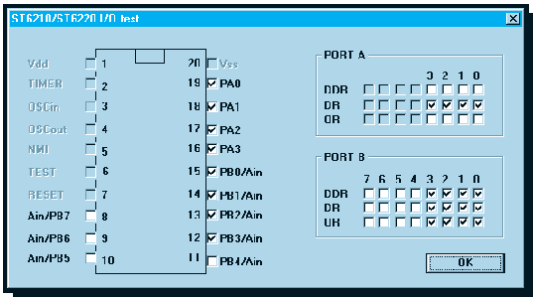


Fig.75 Mappa di configurazione del micro.

caselle DDR - (Data Direction Register) Se queste caselle risultano **vuote** significa che abbiamo definito i piedini **input** (ingressi), quelle contrassegnate con una **V** indicano che li abbiamo definiti **output** (uscite).

caselle DR - (Data Register) Se queste caselle risultano **vuote** significa che sui piedini è presente un **livello logico 0**, se sono contrassegnate da una **V** significa che è presente un **livello logico 1**.

caselle OR - (Opzion Register) Queste caselle servono per selezionare le varie opzioni delle porte. Se con il **DDR** abbiamo predefinito il piedino come **input**, combinandolo con il **DR** e l'**OR** otterremo queste selezioni:

DDR	DR	OR	opzione come ingressi
0	0	0	pull-up senza interrupt
0	1	0	senza pull-up e senza interrupt
0	0	1	con pull-up e con interrupt
0	1	1	ingresso analogico (vedi nota)

Nota: L'ingresso **analogico** non è consentito per i piedini **PA0 - PA1 - PA2 - PA3**.

Per la **porta B**, che dispone di 8 piedini **PB0 - PB1 - PB2 - PB3 - PB4 - PB5 - PB6 - PB7**, vedrete **8 caselle** su **3 file** indicate **7 - 6 - 5 - 4 - 3 - 2 - 1 - 0** che possono essere **vuote** o contrassegnate da una **V**.

caselle DDR - (Data Direction Register) Se queste caselle risultano **vuote** significa che abbiamo definito i piedini **input** (ingressi), quelle contrassegnate con una **V** indicano che li abbiamo definiti **output** (uscite).

caselle DR - (Data Register) Se queste caselle risultano **vuote** significa che sui piedini è presente un **livello logico 0**, se sono contrassegnate da una **V** significa che è presente un **livello logico 1**.

caselle OR - (Opzion Register) Queste caselle servono per selezionare le varie opzioni delle porte. Se con il **DDR** abbiamo predefinito il piedino come **output**, combinandolo con il **DR** e l'**OR** otterremo queste selezioni:

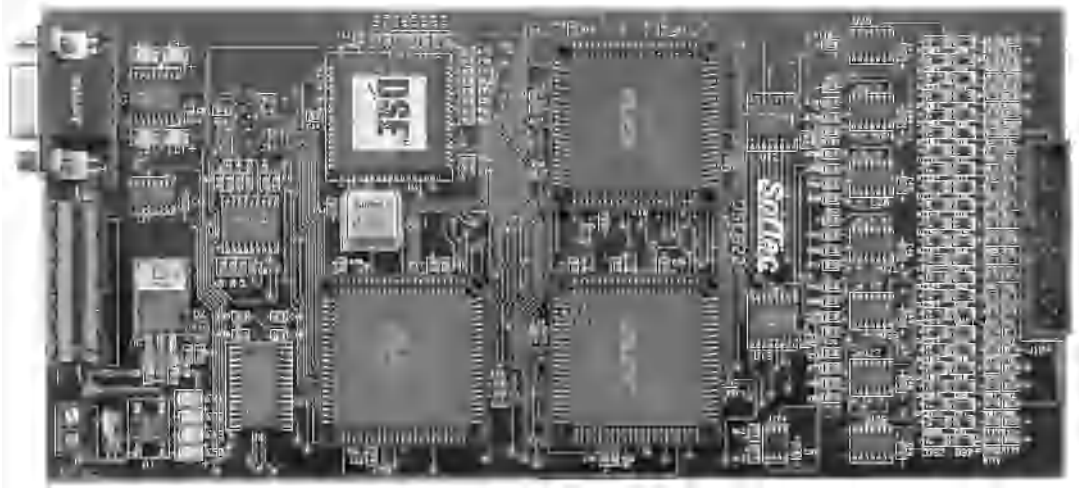
DDR	DR	OR	opzione come uscite
1	0	0	Collettore aperto
1	1	0	Collettore aperto
1	0	1	uscita in Push-Pull

Poiché in molti programmi non si usano i termini **DDR - DR - OR**, ma **PDIR - PORT - POTP**, riportiamo le loro equivalenze:

PDIR	corrisponde a **DDR**
PORT	corrisponde a **DR**
POTP	corrisponde a **OR**

Se rileggerete tutti gli articoli precedenti riguardanti l'**ST6** (vedi Riviste N.172/173 - 174 - 175/176) troverete molti esempi su come procedere per **settare** le porte come **ingressi** e **uscite**.

Potendo **vedere** tramite la finestra visibile in fig.75 tutti i **livelli logici** presenti su questi piedini, potete comprendere quanto risulti semplice accorgersi degli errori, anche perché proseguendo **passo x passo** potete subito verificare come cambiamo i livelli logici sia sugli ingressi sia sulle uscite.



Questo software vi permetterà di vedere nella finestra Disassembler tutte le istruzioni in formato leggibile; nella finestra Register tutti i registri, lo stack level e gli stati dei flags; nella finestra Data il contenuto delle variabili dei registri e della data rom windows ecc. Chi desiderasse acquistare la scheda emulatrice (vedi foto) può rivolgersi a:

SOFTEC MICROSYSTEMS V.le Rimembranze, 19/C 33082 AZZANO DECIMO (PN)
fax 0434-631598 – tel. 0434-640113 – BBS 0434-631904

Nel disegno grafico visibile sulla sinistra di fig.75, potete vedere i piedini che hanno una **casella grigia**, ad esempio:

1 = Vdd (tensione positiva di alimentazione)
2 = Timer
3 = Oscillatore input
4 = Oscillatore uscita
5 = NMI (Interrupt non mascherato)
6 = TEST (piedino di programmazione)
7 = Reset
20 = Vss (tensione negativa di alimentazione)

Le **caselle grigie** non possono essere direttamente testate nella **simulazione** tramite **software**, perché manca la **tensione di alimentazione** ed il **quarzo**, quindi dovrete attivarle con alcuni accorgimenti.

A esempio, non disponendo della **frequenza** di **clock** potrete simulare le funzioni di **timer** solo attivando da programma la **subroutine** legata all'**interrupt del timer**.

La funzione di **reset** può essere invece attivata con un comando presente nella **barra** dei menu.

Chi si procurerà la **scheda emulatrice** sarà in grado di testare in modo automatico anche queste funzioni, perché ha la stessa funzione del **micro**.

In ogni caso risolverete molti problemi già con il solo **software**.
Ad esempio se avete scritto un programma che deve portare a **livello logico 1** il **piedino 6** e simulandolo vi accorgete che è rimasto a **livello logico 0**, vi sarà molto più facile, avanzando **passo** per **passo** e controllando ogni istruzione, trovare quella che, per un banale **errore**, non ha provveduto a modificare il livello logico su tale piedino.

Per uscire dal **debug** è sufficiente portare il cursore sulla scritta **OK** poi cliccare.
Il programma ripartirà **automaticamente** dalla prima **istruzione eseguibile**.
Se lancerete l'esecuzione **automatica** vedrete il programma ruotare all'infinito sulle etichette:

ripeti
main00 - mains1 - main01 - mains2
main02 - mains3 - main03 - mains4

perché non trova premuto nessuno dei quattro interruttori presenti sulla porta **A**.

COME SIMULARE L'INTERRUTTORE

Nel programma **ATEST** tutte le uscite rimangono a **livello logico 0** fino a quando non mettete a **livello logico 1** uno dei quattro **ingressi**.
Poiché siamo in **simulazione** e non avete né un "interruttore" né una tensione "positiva", per portare a **livello logico 1** uno di questi ingressi dovrete forzare l'ingresso desiderato come ora vi spiegheremo.

Cliccate sull'icona di **Stop** visibile in fig.76 rappresentata da una **mano aperta**.

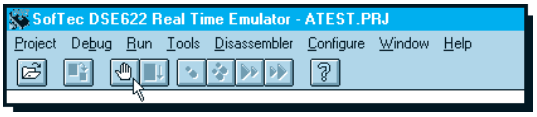


Fig.76 Icona di Stop.

Ammesso di voler portare a **livello logico 1** l'ingresso del piedino **PA2** di **port_a** affinché sull'uscita del piedino **PB2** di **port_b** appaia lo stesso livello logico, per prima cosa dovrete andare nella finestra **Watch**, visibile in fig.77, per controllare l'indirizzo di **port_a**, che risulta essere **C0** (vedi sotto **ADD**).

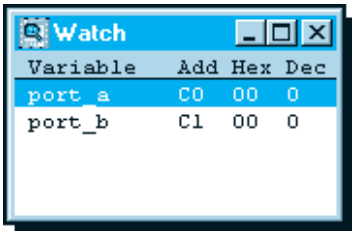


Fig.77 Finestra di Watch.

Per forzare a **livello logico 1** l'ingresso del piedino **PA2** dovete andare nelle finestra **DATA** (vedi fig.78) poi ricercare nella colonna degli indirizzi di memoria il valore esadecimale **C0**.
Trovato **C0** guardate nel righello in alto in cui appaiono i valori esadecimali:

00-01-02-03-04 ——- 08-09-0A-0B-0C-0D-0E-0F

Poiché l'indirizzo di **port_a** è **C0**, scendete dal valore esadecimale **00** fino ad incontrare la riga **C0** e così troverete il valore **00** (vedi fig.78).
Portate il cursore su **00** e cliccate **2 volte**.

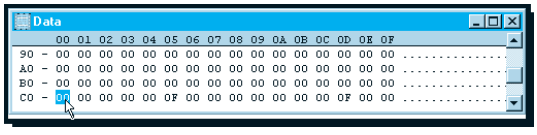


Fig.78 Cliccate 2 volte per vedere la fig.79.

Quando appare la finestra **Edit Data** (fig.79) dovete cliccare sulla scritta **Bits** in modo da far apparire la finestra di fig.80.

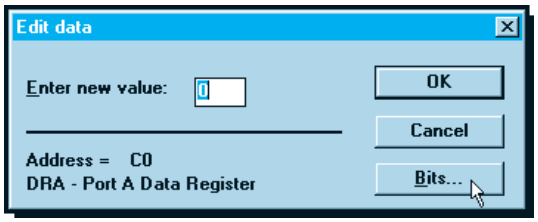


Fig.79 Scegliete Bits per vedere la fig.80.

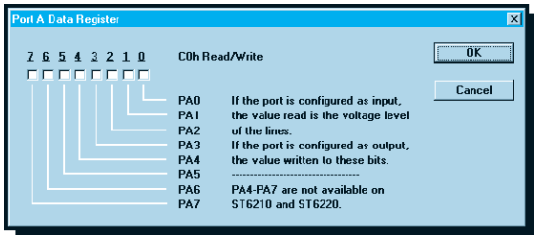


Fig.80 Qui potete forzare i livelli logici.

In questa finestra potete vedere lo stato logico presente su tutti i piedini della porta **A** e poiché tutte le caselle risultano **vuote**, è ovvio che su tutti i piedini è presente un **livello logico 0**.

Volendo **forzare** a **livello logico 1** il piedino d'ingresso **PA2** dovrete portare il cursore nella casella posta sotto il **numero 2** e cliccare.

Comparirà così **V** a conferma del fatto che sul piedino d'ingresso **PA2** è ora presente un **livello logico 1**.

Nota: se cliccherete una seconda volta tornerà a **livello logico 0**.

A questo punto cliccate su **OK** e nella finestra che appare ritornate a cliccare su **OK**: apparirà così la finestra di fig.81.

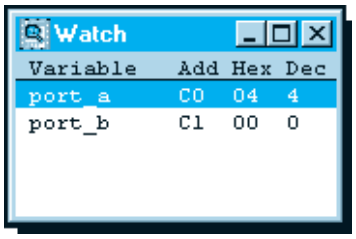


Fig.81 Nel Watch vedrete il nuovo livello.

Se ora guardate all'interno della finestra **Watch** vedrete che qualcosa è cambiato, infatti sotto la colonna **.Hex** troverete **04** e sotto la colonna **Dec.** il numero **4**.

Poiché avevamo "bloccato" il programma pigiando **Stop** (icona con mano) per farlo ripartire dovete ricercare l'etichetta **ripeti** procedendo come segue:

- Attivate la finestra **Disassembler** cliccando all'interno della finestra, quindi cliccate sulla scritta **Disassembler** del menu e poi cliccate sulla scritta **Set New PC** (vedi fig.82).

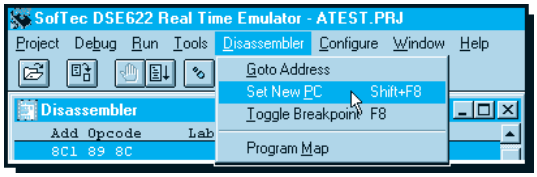


Fig.82 Selezionate Set New PC.

- Apparirà così la finestra di dialogo **New program counter** (vedi fig.83) e cliccando sulla freccia posta a destra della finestra **PC value** dovrete ricercare l'etichetta **ripeti** (vedi fig.84).

Nota: tutte le **etichette** sono in ordine alfabetico.

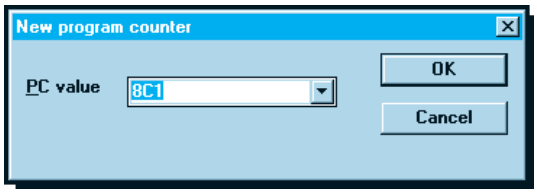


Fig.83 Finestra New Program Counter.

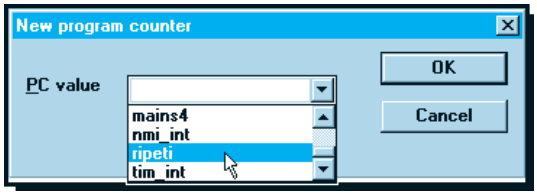


Fig.84 Cliccate sull'etichetta ripeti.

- Ponete il cursore su **ripeti** e cliccate, poi cliccate su **OK** e vedrete apparire la finestra di fig.85. Noterete che la prima riga in alto è ferma sull'etichetta **ripeti**.

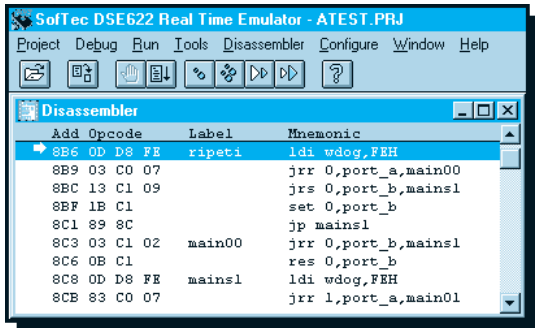


Fig.85 Il programma è fermo su "ripeti".

- A questo punto posizionate il cursore sull'icona di esecuzione **passo** per **passo** e cliccate più volte fino ad arrivare all'istruzione:

8DD 43 C0 07 jrr 2,port_a,main02

Poiché questo bit è a **livello logico 1**, il programma **non salterà** più a **main02** ma proseguirà all'istruzione:

8E0 53 C1 09 jrs 2,port_b,mains3

Poiché **PB2** è a **livello logico 0** non salterà a **mains3**, ma, cliccando sul tasto **passo** per **passo**, proseguirà fino alla successiva istruzione che sarà:

8E3 5B C1 set 2,port_b

Per eseguire questa istruzione cliccate ancora sull'icona **passo passo** ed il piedino **PB2** cambierà il suo livello logico da **0** a **1**.

Per vedere se questa condizione si è verificata potrete guardare nella finestra **Watch** dove leggerete:

port_b C1 04 4

Se volete avere un'ulteriore conferma visiva andate nella finestra **Data** (fig.86), cercate l'indirizzo **C0**, poi andate sotto la colonna **01** e scendendo incontrerete la casella **04**.
Portate il cursore su questa casella e cliccate **due volte**.

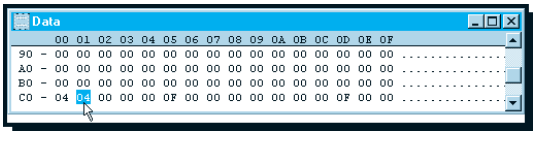


Fig.86 Sulla casella 04 cliccate 2 volte.

Quando appare la maschera **Edit Data** (vedi fig.87) cliccate su **Bits** e comparirà la finestra di dialogo di fig.88 dove potrete vedere che nella casella sotto il numero **2** del piedino **PB2** c'è una **V** ad indicare che questo piedino è a **livello logico 1**.

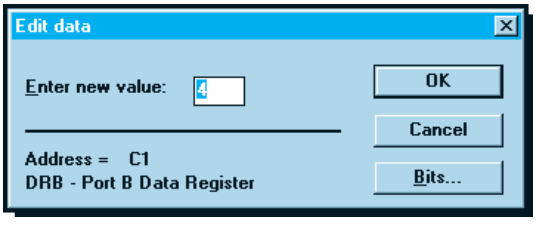


Fig.87 Nell'Edit data cliccate su Bits.

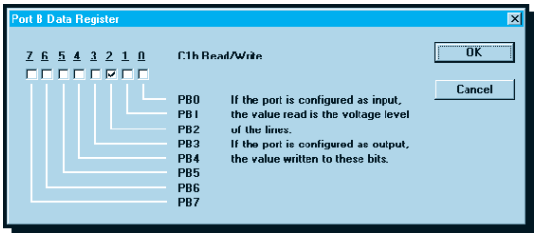


Fig.88 Sul piedino PB2 appare una V.

Per uscire cliccate su **OK** e nella successiva maschera cliccate nuovamente su **OK**: apparirà così la maschera di fig.85.

Ora che avete vi abbiamo spiegato come sia possa modificare un ingresso da **livello logico 0** a **livello logico 1** o viceversa, potete fare un po' di pratica portando a **livello logico 1** anche il piedino di un altro **ingresso** per poi riportarlo a **livello logico 0**, poi verificate se i piedini d'**uscita** sono passati da **livello logico 0** a **livello logico 1**.
Per verificarlo dovrete sempre far ripartire il programma dell'etichetta **ripeti** (vedi fig.85).

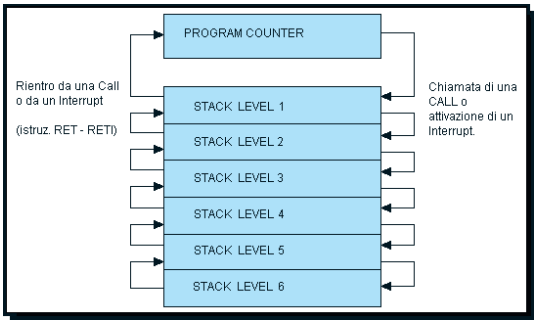


Fig.89 In ogni programma si possono eseguire fino a 6 subroutine nidificate, cioè una all'interno dell'altra.

NON ABBIAMO FINITO

Con questo articolo abbiamo riportato solo una condensata panoramica di quello che riesce a fare questo sofisticato **software di simulazione**.
Per spiegarvi tutto, cioè insegnarvi a capire come scoprire gli **errori**, come **correggerli** ecc., dovremo portarvi tanti altri esempi, e per questo vi rimandiamo al **prossimo** numero.
Comunque quando avrete a disposizione questo **software**, scoprirete voi stessi molte cose ed anche facilmente tramite la funzione di **Help**.

COSTO del dischetto SOFTWARE

Tutti i softwaristi e hobbisti che volessero entrare in possesso di questo dischetto di simulazione per **ST6** siglato **DF622.03** potranno richiederlo alla nostra Direzione. Costo del dischetto ..... € 7,75

tilizzata la direttiva **.else** per la gestione della condizione di "**non vero**". Infatti, in questi due casi ci interessava solo che venisse evidenziata la condizione "**vero**" delle direttive **.ifc**.

Proseguiamo dunque con le istruzioni successive:

```
            .ifc        eq t_baud – 12
set_tcr     .set        140
set_psc     .set        2
            .display    "1200 BAUD"
            .else
            .ifc        eq t_baud – 24
set_tcr     .set        140
set_psc     .set        1
            .display    "2400 BAUD"
            .else
            .ifc        eq t_baud – 48
set_tcr     .set        140
set_psc     .set        0
            .display    "4800 BAUD"
            .else
            .ifc        eq t_baud – 96
set_tcr     .set        70
set_psc     .set        0
            .display    "9600 BAUD"
            .else
            .error      "Errore Selez. Baud"
            .mexit
            .endc
            .endc
            .endc
            .endc
```

Ci troviamo di fronte ad un esempio un po' complesso di compilazione condizionata (**.ifc**) dove per condizione "**vero**" viene eseguita la direttiva **.display**, mentre per "**non vero**" viene posta una nuova condizione **.ifc**, che a sua volta ha una gestione per "**vero**" e rimanda a una nuova condizione di **.ifc** per "**non vero**" e così via.

Vediamo però passo passo cosa succede e analizziamo la prima sequenza:

```
            .ifc        eq t_baud – 12
set_tcr     .set        140
set_psc     .set        2
            .display    "1200 BAUD"
            .else
```

Il compilatore confronta il valore ricavato dalla espressione **t_baud – 12** con **zero** (condizione **eq**) e se risulta "**vero**" definisce la costante **set_tcr** e le associa il valore **140**, inoltre definisce la costante **set_psc** e le associa il valore **2**, infine esegue la direttiva **.display**.
Quest'ultima direttiva si utilizza essenzialmente per visualizzare dei messaggi a video durante la fase di Compilazione del programma.
Nel nostro caso se la condizione fosse vera, a video comparirebbe "**1200 BAUD**", per segnalarci che il programma **SERIAL.ASM** utilizza una velocità di trasmissione di **1200 baud**.

È dunque ora necessario verificare qual è il risultato dell'espressione **t_baud – 12** e per farlo bisogna prima ricavare il valore di **t_baud**.
Se ricordate, la prima istruzione di **SERIAL.ASM** che abbiamo visto era:

```
t_baud      .set        96
```

che assegna a **t_baud** il valore **96**.
Pertanto l'espressione **t_baud – 12** dà come risultato:

96 –12 = 84.

A questo punto è chiaro che l'istruzione diventa:

```
            .ifc        eq 84
```

e poiché l'oggetto della condizione, cioè **84**, non è uguale a **zero**, si attiva la condizione di "**non vero**", e quindi le istruzioni:

```
set_tcr     .set        140
set_psc     .set        2
            .display    "1200 BAUD"
```

non vengono eseguite. Il compilatore passa dunque alle istruzioni poste dopo **.else**:

```
            .ifc        eq t_baud – 24
set_tcr     .set        140
set_psc     .set        1
            .display    "2400 BAUD"
            .else
```

e confronta nuovamente il valore ricavato dalla espressione **t_baud – 24** con **zero** e se "**vero**" definisce la costante **set_tcr** e le associa il valore **140**, definisce **set_psc** e le associa il valore **1**, infine esegue la direttiva **.display**.
Siccome però l'espressione **t_baud – 24** dà come risultato un valore **non** uguale a **0**:

96 – 24 = 72

anche in questo caso viene attivata la condizione di "**non vero**". Il compilatore ignora dunque:

```
set_tcr     .set        140
set_psc     .set        1
            .display    "2400 BAUD"
```

e passa alle istruzioni successive a **.else**:

```
            .ifc        eq t_baud – 48
set_tcr     .set        140
set_psc     .set        0
            .display    “4800 BAUD”
            .else
```

Anche in questo caso il risultato dell’espressione **t_baud – 48** è un valore diverso da **zero**:

96 – 48 = 48

pertanto il compilatore ignora:

```
set_tcr     .set        140
set_psc     .set        0
            .display    “4800 BAUD”
```

e passa alle istruzioni dopo **.else**:

```
            .ifc        eq t_baud – 96
set_tcr     .set        70
set_psc     .set        0
            .display    “9600 BAUD”
            .else
```

In questo caso invece l’espressione **t_baud – 96**:

96 – 96 = 0

soddisfa la condizione per “**vero**” e perciò il compilatore esegue le istruzioni:

```
set_tcr     .set        70
set_psc     .set        0
            .display    “9600 BAUD”
```

definisce così la costante **set_tcr** e le associa il valore **70**, definisce **set_psc** e le associa il valore **0**, infine esegue la direttiva **.display** e a video comparirà la scritta “**9600 BAUD**”.
A questo punto il compilatore ignora l’istruzione **.else** e quelle che seguono:

```
            .error      “Errore Selez. Baud”
            .mexit
```

e passa alla prima delle quattro **.endc** che chiude l’ultima **.ifc** vista, cioè:

```
            .ifc        eq t_baud – 96
```

Poi va alla seconda **.endc** che chiude:

```
            .ifc        eq t_baud – 48
```

Poi va alla terza **.endc** che chiude:

```
            .ifc        eq t_baud – 24
```

Poi va alla quarta **.endc** che chiude:

```
            .ifc        eq t_baud – 12
```

Per facilitarvi nella comprensione della sequenza logica delle istruzioni, alla destra del listato visibile in fig.3 abbiamo legato con dei trattini le condizioni **.ifc** alle rispettive **.endc**.
Si vede così abbastanza chiaramente che si tratta di una serie di **.ifc** racchiuse una dentro l’altra, dove la prima del listato è l’ultima ad essere “chiusa”.
Si parla in questo caso di **.ifc** “**nested**” che tradotto vuol dire “nidificate”.
Vi ricordiamo che è importantissimo “chiudere” sempre ogni **.ifc** con una **.endc**.
Il compilatore segnala infatti errore nel caso che siano state inserite un numero maggiore o minore di **.endc** rispetto alle **.ifc** inserite.
Segnala inoltre errore anche quando si inseriscono più **.else** rispetto alle **.ifc**.

Dopo l’ultima **.endc** il compilatore trova la direttiva **.endm** che gli segnala la fine della macro.

A questo punto prosegue con la compilazione delle rimanenti istruzioni del programma **SERIAL.ASM** e quando arriva alla routine che predispone il **timer** per gestire la velocità di trasmissione, carica nel registro **tcr** (Contatore del Timer) il valore corrispondente alla costante **set_tcr** (nel nostro esempio **70**) e nel **Prescaler** del registro **tscr** il valore corrispondente alla costante **set_psc** (nel nostro esempio **0**).
Questo permetterà di gestire i tempi strettamente legati alla velocità di trasmissione.
Vi ricordiamo che trattandosi di esempi, i valori **70** e **0** che abbiamo utilizzato sono indicativi, poiché quello che ci premeva farvi capire è il meccanismo con cui si ottengono questi valori.
A fine compilazione comparirà a video il messaggio visibile in fig.5.

Notate la dicitura “**9600 BAUD**” visibile prima della scritta ***** SUCCESS ***** che testimonia che è stata selezionata la velocità di **9600** baud per la trasmissione.

A questo punto vi starete chiedendo cosa succede se nel definire **t_baud**, anziché utilizzare uno dei valori numerici gestiti dalla macro **setbaud** (cioè 12 o 24 o 48 o 96) inseriamo un valore diverso, ad esempio 75.
Aiutandovi con il listato di fig.3 che abbiamo appena descritto provate a simulare il compilatore.

Tutte e quattro le espressioni che utilizzano **t_baud** danno un risultato diverso da zero; l'ultima dà addirittura un risultato negativo.
Ne consegue che verranno eseguite sempre le condizioni per "**non vero**" arrivando a:

```
.error      "Errore Selez. Baud"
.mexit
```

La direttiva **.error** viene utilizzata per fare apparire a video la segnalazione di errore seguita, dove ci sia, dalla frase inserita tra virgolette.
Quando il compilatore incontra questa direttiva, visualizza il messaggio a video e continua comunque la compilazione del programma, ma non genera nessun programma eseguibile (**.HEX**).
Questa direttiva si utilizza perciò per segnalare un caso di errore grave.
La direttiva **.mexit** che abbiamo inserito di seguito viene utilizzata per uscire forzatamente dalla compilazione di una macro senza dover arrivare alla sua fine naturale, cioè all'istruzione **.endm**.

Nella fig.6 potete vedere il messaggio che sarebbe apparso dopo la compilazione di **SERIAL.ASM** con **t_baud** non valido.
Viene infatti mostrato a video il messaggio di errore e la dicitura finale "**No object created**".

Torniamo ora all'esempio corretto dove **t_baud** vale **96** e la compilazione dà esito positivo.
Qualcuno potrebbe obiettare che sono state inserite molte istruzioni, con una conseguente perdita di spazio e tempo di esecuzione, per ottenere la configurazione di due costanti: **set_tcr** e **set_psc**.

Vorremmo però farvi osservare che se in futuro si presenterà la necessità di scrivere più di un programma che esegua una trasmissione e/o una ricezione seriale asincrona, ognuno a una diversa velocità di trasmissione tra le 4 proposte nella macro, sarà sufficiente definire in maniera corretta il valore di **t_baud** per avere già tutto predisposto.
Inoltre se siete dei corretti osservatori, avrete notato che la macro **setbaud** è composta esclusivamente da direttive dell'Assembler, e voi dovreste sapere che queste non occupano spazio di memoria, non vengono eseguite in fase di esecuzione del programma e non generano nessuna opcode.
A riprova di quanto detto abbiamo lanciato l'esecuzione del programma **SERIAL.HEX** tramite il simulatore **SimST626** (presentato sulla rivista **N.197**) e come visibile in fig.7, dopo l'istruzione:

```
call        init_p
```

viene eseguita l'istruzione:

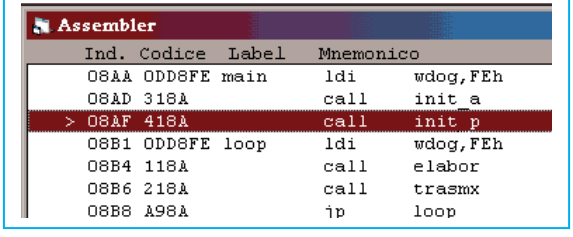
```
loop        ldi         wdog,FEh
```

e non vi è più traccia di:

```
setbaud     t_baud
```

come invece riportato nel **SERIAL.ASM** di fig.2.

Fig.7 Esecuzione del file SERIAL.HEX



Se non disponete di un simulatore, per sapere se i dati sono stati correttamente inseriti nel registro **tcr** e nel registro **tscr** del Timer, vi dovete fidare di ciò che appare a video alla fine della compilazione e cioè di un messaggio simile a quello visibile in fig.5.
Esiste però un altro controllo che si può effettuare quando non si dispone di un simulatore.
È infatti sufficiente compilare il programma inserendo l'opzione **–S** per ottenere così anche il file con estensione **.SYM**.
Come già spiegato sulla rivista **N.194** relativamente alle opzioni del compilatore assembler, questo file contiene tutte le etichette e tutte le costanti utilizzate nel programma con a fianco il loro valore espresso in **esadecimale**.
Vediamo dunque, tramite la fig.8, il listato del file **SERIAL.SYM** e andiamo a verificare i valori di:

```
t_baud   :   EQU  00060H  C
```

dove appunto **60h** espresso in decimale è **96**.

```
set_psc  :   EQU  00000H  C
```

dove il valore **00h** espresso in decimale è **0**.

```
set_tcr  :   EQU  00046H  C
```

dove il valore **46h** espresso in decimale è **70**.

SECONDO ESEMPIO

Per il secondo esempio abbiamo realizzato una **macro** che abbiamo chiamato **ritardo** e che abbiamo salvato nel file **RITARDO.LMA**.

Abbiamo quindi scritto un semplice programma che abbiamo chiamato **PROVA2.ASM** e in due diversi punti abbiamo utilizzato la **macro ritardo**.

Fig.8 LISTATO del file SERIAL.SYM

Per ottenere un file con estensione .SYM, bisogna compilare il programma sorgente inserendo l'opzione –S. In questo modo si ottiene l'elenco delle etichette definite in Program Space e delle costanti simboliche utilizzate nel programma sorgente. Come potete vedere in queste righe, accanto a ogni etichetta (definita con P) o costante (definita con C), è espresso l'indirizzo in valore esadecimale.

```
t_baud    :   EQU 00060H C
ad_int    :   EQU 008a5H P
init_a    :   EQU 008a3H P
elabor    :   EQU 008a1H P
init_p    :   EQU 008a4H P
inizio    :   EQU 00880H P
trasmx    :   EQU 008a2H P
main      :   EQU 008aaH P
loop      :   EQU 008b1H P
nmi_int   :   EQU 008a9H P
set_psc   :   EQU 00000H C
set_tcr   :   EQU 00046H C
tim_int   :   EQU 008a6H P
A_int     :   EQU 008a8H P
BC_int    :   EQU 008a7H P
```

Fig.9 LISTATO del PROGRAMMA RITARDO.LMA

```
                .macro    ritardo time,?lop1
                .ifc      ndf freqz
                .error    "Frequenza quarzo non definita"
                .mexit
                .endc
                .ifc      gt time*freqz/6/13-256
                .error    "Tempo troppo lungo"
                .mexit
                .endc
                .ifc      le time*freqz/6/13-1
                .error    "Tempo troppo corto"
                .mexit
                .endc

                ldi       carmat,time*freqz/6/13-1
      lop1      dec       carmat
                jrnz      lop1
                .endm
```

Fig.10 LISTATO del PROGRAMMA PROVA2.ASM

```
carmat  .def      084h            ;Variabile per avere un ritardo
freqz   .set      8               ;Segnala 8MHz di frequenza
        .input    "RITARDO.LMA"

main
        ldi       wdog,0feh       ;ricarica il Watchdog
        call      set_pin         ;configura le porte
        call      elab1           ;prima elaborazione
        call      delay1          ;esegui un ritardo
        call      elab2           ;seconda elaborazione
        call      delay2          ;esegui un ritardo
        jp        main            ;ripeti

delay1  ritardo   1200            ;Esegue un ritardo di 1200 us
        ret

delay2  ritardo   1500            ;Esegue un ritardo di 1500 us
        ret
```

In fig.9 potete vedere il listato della **macro** chiamata **ritardo**, mentre in fig.10 potete vedere il listato del programma **PROVA2.ASM** relativo alle sole istruzioni che ci interessano ai fini dell'esempio.

La macro riportata in fig.9 ci serve per generare un **ritardo variabile**, il cui valore andrà inserito all'interno del programma **PROVA2.ASM** in corrispondenza delle istruzioni che richiamano questa macro, cioè:

```
delay1      ritardo     1200

delay2      ritardo     1500
```

I valori numerici **1200** e **1500** sono i valori che verranno passati dal programma sorgente alla macro durante la compilazione e corrispondono al ritardo espresso in **microsecondi** che verrà generato.
Nelle istruzioni della macro è inoltre previsto un controllo sui valori numerici passati alla macro stessa, in modo che se il ritardo è minore di **10 microsecondi** o maggiore di **2496 microsecondi**, venga segnalato **errore**.

Una macro come quella da noi chiamata **ritardo** può risultare molto utile quando si devono inserire dei ritardi in determinati programmi, perché eviterà di dover calcolare di volta in volta il tempo dei **cicli** delle istruzioni.

Adesso vediamo cosa avviene quando **compiliamo** il programma **PROVA2.ASM**.

Seguendo il listato di fig.10 troviamo subito la prima istruzione:

```
carmat      .def        084h
```

dove la variabile **carmat** viene associata all'area di Data Space **084h**. Questa variabile è quella che verrà utilizzata dalla macro **ritardo**.

La seconda istruzione:

```
freqz       .set        8
```

definisce la costante **freqz** associandole il valore **8**. Questo valore corrisponde alla frequenza di oscillazione del quarzo da **8 MHz** utilizzato per il clock. E' ovvio che se si utilizzasse un quarzo da **4 MHz**, l'istruzione dovrebbe cambiare in:

```
freqz       .set        4
```

La terza istruzione che incontriamo riguarda la direttiva **.input**. Come abbiamo già avuto modo di ricordare con il 1° esempio, questa direttiva informa il **compilatore** che deve caricare la macro **ritardo** nel programma principale **PROVA2.ASM**, prelevandola dal file **RITARDO.LMA**.

Tralasciamo le istruzioni successive perché non strettamente inerenti all'argomento di questo articolo e andiamo direttamente a:

```
delay1      ritardo     1200
            ret
```

Questa sub-routine ha il compito di effettuare un ritardo di **1200 microsecondi**.

Il **compilatore** associa l'etichetta **delay1** alla istruzione **ritardo 1200** e, poiché ha riconosciuto che **ritardo** è una **macro**, inizia a compilare le istruzioni contenute nella stessa macro, che, come abbiamo già ricordato, si trovano in fig.9.

La prima istruzione di fig.9:

```
            .macro      ritardo time,?lop1
```

identifica la **macro ritardo** e informa il compilatore che in questa macro verrà passato il parametro **time** e che verrà utilizzata l'etichetta interna **?lop1**.

Ora il **compilatore** prende in esame il blocco di istruzioni:

```
.ifc        ndf freqz
.error      "frequenza quarzo non definita"
.mexit
.endc
```

che equivale a: se la **freqz** del quarzo **non** è stata definita, segnala a video un messaggio di errore con la scritta **"Frequenza quarzo non definita"**, quindi esci dalla macro senza generare il programma eseguibile (istruzione **.mexit**).
Poiché però nel programma sorgente **freqz** è stata definita **.set 8**, questo blocco di istruzioni viene totalmente ignorato.

Il compilatore passa quindi al successivo blocco di istruzioni:

```
.ifc        gt time*freqz/6/13–256
.error      "Tempo troppo lungo"
.mexit
.endc
```

che equivale a: se il risultato dell'espressione **time*freqz/6/13–256** è maggiore (**gt**) di **0** allora segnala a video il messaggio di errore **"Tempo troppo lungo"** ed esci dalla macro senza generare

nessun programma eseguibile.

Nota: vi ricordiamo che le **espressioni** sono state spiegate nella rivista **N.189**.

Il compilatore esegue automaticamente il calcolo di questa espressione, ma noi possiamo verificare, procedendo passo passo, se la condizione è **vera** o **non vera**.

Poiché **time** è il parametro della **macro ritardo** che viene passato nel programma **PROVA2.ASM**, quando si richiama la macro con l'istruzione:

```
delay1      ritardo      1200
```

noi sappiamo che **time** equivale a **1200**, quindi l'espressione **time*freqz/6/13–256** diventa:

1200*freqz/6/13–256

Poiché la costante **freqz** è stata definita associandola al valore **8** del quarzo, la nostra espressione diventa:

1200*8/6/13–256

Come prima operazione eseguiamo la moltiplicazione:

1200*8 = 9600

poi eseguiamo la prima divisione:

9600/6 = 1600

quindi la seconda divisione:

1600/13 = 123,0769

e infine, dopo aver scartato i **decimali**, eseguiamo l'ultima operazione con il solo numero intero:

123–256 = –133

Quindi l'istruzione:

```
.ifc        gt time*freqz/6/13–256
```

diventa in pratica:

```
.ifc        gt –133
```

Poiché il valore **–133** non è **maggiore** di **0**, la condizione posta da **.ifc** non viene soddisfatta e quindi il blocco di istruzioni viene ignorato e non viene segnalato **errore**.

Il **compilatore** passa poi al successivo blocco di istruzioni:

```
.ifc        le time*freqz/6/13–1
.error      "Tempo troppo corto"
.mexit
.endc
```

che equivale a: se il risultato dell'espressione **time*freqz/6/13–1** è minore o uguale (**le**) a **0**, segnala a video il messaggio di errore **"Tempo troppo corto"**, quindi esci dalla macro senza generare il programma eseguibile.

Il compilatore esegue automaticamente il calcolo di questa seconda espressione, ma noi possiamo verificare passo passo se questa condizione risulta **vera** o **non vera**.

Poiché abbiamo già visto che **time** vale **1200**, mentre a **freqz** si associa il valore **8**, l'espressione **time*freqz/6/13–1** diventa:

1200*8/6/13–1

Come prima operazione eseguiamo la moltiplicazione:

1200*8 = 9600

poi eseguiamo la prima divisione:

9600/6 = 1600

poi eseguiamo la seconda divisione:

1600/13 = 123,0769

e infine, dopo aver scartato i **decimali**, eseguiamo l'ultima operazione con il solo numero intero:

123–1 = 122

Quindi l'istruzione:

```
.ifc        le time*freqz/6/13–1
```

diventa in pratica:

```
.ifc        le 122
```

e poiché il valore **122** è **maggiore** di **0**, anche questo blocco di istruzioni verrà ignorato senza segnalare nessun **errore**, perché il valore di **1200 microsecondi** che vogliamo utilizzare come ritardo è un valore ammesso dalla macro.

A questo punto il compilatore passa a:

```
ldi     carmat,time*freqz/6/13–1
```

e dopo aver fatto il calcolo, che darà come risultato sempre **122**:

```
ldi     carmat,122
```

lo carica nella variabile **carmat** e lo trasforma in formato eseguibile.

Continuando la compilazione trova:

```
lop1  dec     carmat
      jrnz    lop1
      .endm
```

Con la direttiva **.endm**, il compilatore sa che la macro **ritardo** è finita e torna al programma sorgente **PROVA2.ASM** per proseguire la compilazione delle istruzioni, dove trova:

```
ret
```

che serve per rientrare dalla **call delay1** (vedi fig.10). Quindi può proseguire con:

```
delay2    ritardo    1500
```

e riconoscendo la macro **ritardo**, ricompila nuovamente le istruzioni della macro sostituendo il **time 1200** con il nuovo tempo **1500**.

Quindi l'espressione:

```
.ifc    gt time*freqz/6/13–256
```

viene semplificata in:

```
1500*8/6/13–256
```

il cui risultato è:

```
1500*8 = 12000
12000/6 = 2000
2000/13 = 153
153–256 = –103
```

L'istruzione diventa pertanto:

```
.ifc    gt –103
```

Poiché il valore **–103** non è **maggiore** di **0**, questo blocco di istruzioni viene ignorato e non viene segnalato nessun **errore**.

Il **compilatore** passa poi al secondo blocco di istruzioni, dove l'espressione:

```
.ifc    le time*freqz/6/13–1
```

viene semplificata in:

```
1500*8/6/13–1
```

il cui risultato è:

```
1500*8 = 12000
12000/6 = 2000
2000/13 = 153
153–1 = 152
```

L'istruzione diventa pertanto:

```
.ifc    le 152
```

e poiché il valore **152** è **maggiore** di **0** anche questo blocco di istruzioni verrà ignorato senza segnalare nessun **errore**, perché il valore di **1500 microsecondi** che vogliamo utilizzare come ritardo è un valore ammesso dalla macro.

A questo punto il compilatore passa a:

```
ldi     carmat,time*freqz/6/13–1
```

e dopo aver fatto il calcolo, che da come risultato sempre **152**:

```
ldi     carmat,152
```

lo carica nella variabile **carmat** e lo trasforma in formato eseguibile.

Continuando la compilazione trova:

```
lop1  dec     carmat
      jrnz    lop1
      .endm
```

Con la direttiva **.endm** il compilatore sa che la macro **ritardo** è finita e torna al programma sorgente **PROVA2.ASM** per proseguire la compilazione delle istruzioni, dove trova:

```
ret
```

che serve per rientrare dalla **call delay2** (vedi fig.10).

Ora proviamo a simulare il programma ottenuto con la compilazione, cioè **PROVA2.HEX**, e in fig.11 vediamo la parte relativa al nostro esempio.

Osservate le righe evidenziate in giallo che si rife-

riscono alla sub-routine **delay1** ricavata dalla macro **ritardo**:

```
delay1      ldi       carmat,7Ah
L0$         dec       carmat
            jrnz      L0$
            ret
```

Trasformando il valore esadecimale **7Ah** nel suo decimale corrispondente, otteniamo **122**, che, come desiderato, ci permetterà di ottenere un ritardo di **1200 microsecondi**.
Le righe evidenziate in verde si riferiscono invece alla sub-routine **delay2** ricavata sempre dalla macro **ritardo**:

```
delay2      ldi       carmat,98h
L1$         dec       carmat
            jrnz      L1$
            ret
```

Trasformando il valore esadecimale **98h** nel suo decimale corrispondente, otteniamo **152**, che, come desiderato, ci permetterà di ottenere un ritardo di **1500 microsecondi**.

Prima però di verificare se effettivamente si ottengono i ritardi voluti, apriamo una parentesi per ricordarvi che, quando il compilatore, come nel nostro caso, incontra nelle macro delle etichette o labels interne (**?lop1** in fig.9) le genera automaticamente nel file **.HEX** assegnandole un numero consecutivo. Ecco perché le istruzioni della macro:

```
lop1        dec       carmat
            jrnz      lop1
```

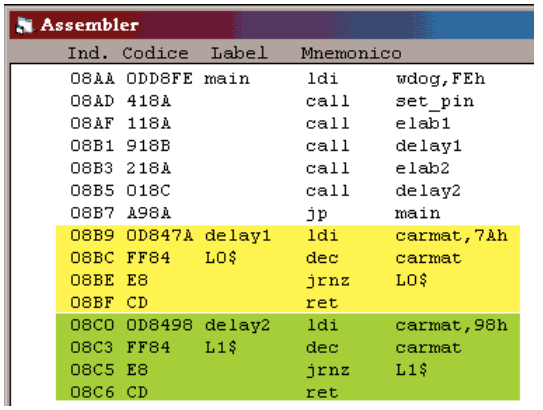
nella simulazione del programma sono diventate rispettivamente:

```
L0$         dec       carmat
            jrnz      L0$

L1$         dec       carmat
            jrnz      L1$
```

Chiudiamo la parentesi e andiamo a fare un piccolo controllo per verificare se effettivamente vengono ottenuti i ritardi voluti.
Sommiamo dunque i **cicli** delle istruzioni delle due sub-routine e moltiplichiamo il risultato per il tempo di un **ciclo macchina** che corrisponde a **1,625 microsecondi**.
Nell'articolo relativo al **software simulatore** per testare i micro **ST6** pubblicato sulla rivista **N.185**, abbiamo fornito l'elenco completo delle istruzioni Assembler indicando, tra le altre cose, il **numero dei cicli macchina**.

Fig.11 Esecuzione del file PROVA2.HEX



Per un ritardo di **1200** abbiamo:

```
            call      delay1      1 x 4 cicli     =    4
delay1      ldi       carmat,7Ah  1 x 4 cicli     =    4
L0$         dec       carmat      122 x 4 cicli   = 488
            jrnz      L0$         122 x 2 cicli   = 244
            ret                   1 x 2 cicli     =    2
```

Sommando i **cicli macchina** di queste sub-routine otteniamo **742**.
Poiché un ciclo macchina corrisponde a **1,625 microsecondi** noi otteniamo un effettivo ritardo di:

742 x 1,625 = 1205,75 microsecondi

La differenza di **5,75 microsecondi** in più rispetto al ritardo impostato nel file sorgente non è un errore, ma, in questo caso, è dovuto al necessario arrotondamento operato sui decimali nell'espressione calcolata.

Per un ritardo di **1500** abbiamo:

```
            call      delay2      1 x 4 cicli     =    4
delay2      ldi       carmat,98h  1 x 4 cicli     =    4
L1$         dec       carmat      152 x 4 cicli   = 608
            jrnz      L1$         152 x 2 cicli   = 304
            ret                   1 x 2 cicli     =    2
```

Sommando i **cicli macchina** di queste sub-routine otteniamo **922**.
Poiché un ciclo macchina corrisponde a **1,625 microsecondi** noi otteniamo un effettivo ritardo di:

922 x 1,625 = 1498,25 microsecondi

La differenza di **1,75 microsecondi** in meno rispetto al ritardo impostato nel file sorgente non è un errore, ma, in questo caso, è dovuto al necessario arrotondamento operato sui decimali nell'espressione calcolata.

IL programma LINKER

Con l'articolo sul linker LST6 di cui ci occupiamo in queste pagine, proseguiamo l'esposizione dei diversi aspetti della programmazione dei microcontrollori della serie ST6. Non vi nascondiamo che l'argomento non è dei più semplici, ma con l'aiuto di qualche esempio, siamo certi che anche questa materia non avrà più segreti.

Fino ad oggi nella realizzazione di un programma in **Assembler** per i micro **ST6** ci siamo sempre posti l'obiettivo di scrivere un programma **sorgente**, cioè un file in formato **.ASM** dal quale ottenere un file in formato eseguibile **.HEX**.

Infatti, in tutti gli articoli pubblicati e nei diversi esempi di programmi che vi abbiamo fornito, abbiamo sempre pensato al programma come a una cosa unica, a sé stante, risultato della compilazione in Assembler di un unico file sorgente con tutt'al più l'inserimento, tramite la direttiva **.input**, di subroutine, macro o definizioni di variabili esterne, ma sempre in **formato sorgente**.

L'articolo di oggi si propone invece di illustrarvi un secondo **metodo** per la realizzazione dei vostri programmi, non necessariamente migliore dell'altro, ma sicuramente differente perché presuppone il conseguimento di un altro scopo.

Con il **linker**, termine che possiamo rendere in italiano con **programma di collegamento**, si può ottenere un programma finale eseguibile **.HEX** senza avere il corrispondente programma in formato sorgente, ma **collegando** programmi diversi assemblati in formato oggetto **.OBJ**.
Per semplicità possiamo dunque definire il **linker** come un programma che concatena moduli software al fine di realizzare un programma eseguibile completo.

Il primo passo per usare il **linker** è quello di disporre di una serie di programmi assemblati singolarmente in formato oggetto **.OBJ** utilizzando le opportune opzioni del programma compilatore **Ast6**.

Il secondo passo è quello di lanciare il programma **Lst6** di linkaggio dei file **.OBJ** con le opportune opzioni, in modo da ottenere il programma definitivo eseguibile in formato **.HEX**.

I PROGRAMMI in formato .HEX

Sulla base di quanto fin qui detto, qualcuno potrebbe domandarsi perché non usare il **linker** direttamente con i programmi in formato **.HEX**, invece di utilizzare dei programmi in formato **.OBJ**.

Quando si lancia la compilazione Assembler di un programma, ad esempio **PIPPO.ASM**, a compilazione conclusa, se non vi sono errori, si genera un programma in formato Intel eseguibile, nel nostro caso **PIPPO.HEX**.

Nel file in formato **.HEX**, le singole istruzioni del programma sorgente **.ASM**, sono tradotte in codice binario direttamente eseguibile e soprattutto vi è una **corrispondenza diretta** tra le locazioni di memoria, sia RAM che ROM, attribuite durante la stesura del programma sorgente e quelle ottenute dalla compilazione dell'eseguibile **.HEX**.

I PROGRAMMI in formato .OBJ

I programmi in formato oggetto **.OBJ** si ottengono utilizzando l'opzione **–O** quando si lancia la compilazione di un programma.
Ad esempio, se compiliamo il file sorgente **PIPPO.ASM** con le opzioni:

Ast6 –L –O PIPPO.ASM

otteniamo il programma **PIPPO.OBJ**.

Nota: ricordiamo ai lettori che le opzioni del compilatore Assembler e il loro utilizzo sono state ampiamente trattate nella rivista **N.194**.

Il programma generato in formato **.OBJ** ha due caratteristiche:

1 – non è direttamente **eseguibile**, pertanto non può essere simulato né caricato su un micro.

per i microprocessori ST6

Infatti, all'interno di ogni programma, dopo la definizione dei registri e della variabili, viene posta la direttiva **.org 0800h** o **0880h** che serve a posizionare in maniera **assoluta** le istruzioni da quell'indirizzo di memoria ROM in poi.
La stessa cosa si ottiene alla fine con la direttiva **.org 0FF0h**, che posiziona le eventuali gestioni dei vettori di interrupt da quell'indirizzo di memoria assoluta in poi.

Comprenderete quindi che se si tentasse di "**unire**" tramite il linker parti di più programmi in formato **.HEX**, essendo ognuna di esse posizionata a un **indirizzo fisso** di **memoria**, si dovrebbe realizzare un programma ad **incastro**, in maniera che la routine che ci interessa inserire dopo le istruzioni del programma principale iniziasse esattamente ad una ben precisa locazione di memoria successiva a quella già occupata, in caso contrario si correrebbe il rischio di "sovrascrivere" porzioni di programma.
Unire moduli software diventerebbe così un lavoro estremamente difficile, se non impossibile.

A facilitare il nostro compito, ci viene in aiuto il formato **.OBJ**, che essendo **rilocabile** e **non eseguibile**, meglio si presta ad essere linkato.
Vediamo dunque cosa sono i programmi in formato **.OBJ** e come ottenerli.

2 – le istruzioni contenute non sono in formato assoluto, bensì in formato "**rilocabile**".
In altre parole le istruzioni hanno un indirizzamento di memoria e di Program Counter **relativo** (e non assoluto come nel formato **.HEX**) e quindi possono essere "ricollocate" o, utilizzando un termine specifico, **rilocate**.

E' dunque utile chiarire cosa si intende per indirizzamento relativo e indirizzamento assoluto.

Pensate ad esempio alla numerazione delle pagine di un libro qualsiasi o di una rivista.
Ogni numero specifica la posizione della pagina rispetto alle altre, per cui il numero 10 specifica che quella pagina è la decima della rivista, il numero 128 specifica che quella pagina è la centoventottesima della rivista, e così via.
In questo caso si può parlare di **indirizzamento assoluto** e, a patto di non intervenire in maniera cruenta con tagli o strappi, questo indirizzamento **non cambierà** mai.

Se però decidete di raccogliere insieme gli articoli riguardanti un unico argomento, la numerazione delle pagine non sarà più consecutiva, cioè non avrà una progressione numerica, ma sarà **relativa** alla rivista dalla quale proveniva l'articolo.

Solo quando “concatenerete” uno all’altro gli articoli rinumerando le pagine, darete un nuovo indirizzamento assoluto alla vostra raccolta.

Chiusa questa parentesi, torniamo ai programmi **.OBJ** per precisare che non basta inserire l’opzione **–O** nella compilazione Assembler per ottenere questo formato.
Se provate a compilare un vostro programma inserendo questa opzione, vedrete che il compilatore vi segnalerà un certo numero di errori.

Proprio per le sue peculiarità, nei programmi sorgente bisogna inserire alcune precise direttive e modificarne o toglierne altre prima di generare il formato **.OBJ**.

Le direttive specifiche che servono per generare il formato **oggetto** e quindi anche per **linkare** i programmi **.OBJ**, sono:

.pp_on
.extern
.section
.window
.windowend
.global
.notransmit
.transmit

Nell’esempio che vi proponiamo di seguito cercheremo di chiarire in quale modo e perché vanno utilizzate queste direttive per ottenere un programma **.OBJ** senza errori.

I programmi PLEXER.ASM e PCONT.ASM

Per il nostro esempio abbiamo utilizzato un nostro datato, ma semplice programma dimostrativo che esegue un conteggio e lo visualizza su due display. In fig.1 abbiamo riportato il listato del programma **CONTA.ASM** così come lo avevamo realizzato.

Dal programma **CONTA.ASM** abbiamo estratto le istruzioni che vedete evidenziate in azzurro in fig.1 e le abbiamo inserite in un nuovo programma che abbiamo chiamato **PLEXER.ASM**.
Questo programma ci mette a disposizione una serie di subroutine che gestiscono l’incremento o il decremento di un contatore e la visualizzazione a due cifre del risultato su 2 display in multiplexer.

Abbiamo quindi cancellato dal programma **CONTA.ASM** le istruzioni inserite in **PLEXER.ASM** e abbiamo salvato ciò che rimaneva con il nome **PCONT.ASM** per non confonderlo con l’originale.

LISTATO del programma CONTA.ASM

```
;* Programma per fare un conteggio *

        .title    “CONTA”
        .vers     “ST62E25”
        .w_on
        .romsize  4
        .input    “ST62X.DEF”

;VARIABILI usate da questo PROGRAMMA

lsb     .def      084h
msb     .def      085h
del1    .def      086h
del2    .def      087h
up_dw   .def      088h

        .org      0800h

inizio
        ldi       wdog,0ffh

        ldi       port_a,00000000b
        ldi       pdir_a,00001100b
        ldi       popt_a,00001100b

        ldi       port_b,00000000b
        ldi       pdir_b,11111111b
        ldi       popt_b,11111111b

        ldi       port_c,00000000b
        ldi       pdir_c,00000000b
        ldi       popt_c,00000000b

;***   Disabilita gli Interrupt
        ldi       adcr,0
        ldi       tscr,0
        ldi       ior,0

        reti

        jp        main

;***     GESTORI di INTERRUPT      ***

ad_int   reti
tim_int  reti
BC_int   reti
A_int    reti
nmi_int  reti
```

Fig.1 Dal programma CONTA.ASM, di cui vi forniamo il listato, abbiamo estratto le istruzioni evidenziate in azzurro e le abbiamo salvate nel file PLEXER.ASM (vedi fig.3). Le istruzioni rimaste sono state salvate nel file PCONT.ASM (vedi fig.2).

```
;***           SUBROUTINE           ***

;- multiplexa le 2 cifre sui display
mulplx
        ld        a,lsb
        addi      a,40h
        ld        x,a
        ld        a,(x)
        ldi       port_a,00001100b
        ld        port_b,a
        ldi       port_a,00000100b

        ld        a,msb
        addi      a,40h
        ld        x,a
        ld        a,(x)
        ldi       port_a,00001100b
        ld        port_b,a
        ldi       port_a,00001000b

        ret

;- incremento delle 2 cifre
;- con controlli
increm
        inc       lsb
        ld        a,lsb
        cpi       a,10
        jrnz      incr1
        ldi       lsb,0
        inc       msb
        ld        a,msb
        cpi       a,10
        jrnz      incr1
        ldi       msb,0
incr1
        ret

;- decremento delle 2 cifre
;- con controlli
decrem
        ld        a,lsb
        cpi       a,0
        jrnz      decr1
        ldi       lsb,9

        ld        a,msb
        cpi       a,0
        jrnz      decr2
        ldi       msb,9
        ret

decr2   dec       msb
        ret

decr1   dec       lsb
        ret
```

```
;***     PROGRAMMA PRINCIPALE      ***

main
        ldi       wdog,0feh

        ldi       lsb,0
        ldi       msb,0
        ldi       up_dw,1

        ldi       drw,digit.w

loop    ldi       del1,17
main1   ldi       del2,255
main2   ldi       wdog,0feh

        call      mulplx

        dec       del2
        jrz       main3
        jp        main2
main3   dec       del1
        jrz       main6

        jrs       0,port_a,main4
        ldi       up_dw,0
main4   jrs       1,port_a,main5
        ldi       up_dw,1
main5
        jp        main1
main6
        ld        a,up_dw
        cpi       a,0
        jrz       main7

        call      increm
        jp        loop

main7   call      decrem
        jp        loop

;*** tabella con i segmenti per far
; apparire sui display le cifre ***

        .block    64-$%64
digit   .byte     192,249,164,176,153
        .byte     146,130,248,128,144

;*** VETTORI DI INTERRUPTS ***

        .org      0ff0h
        jp        ad_int
        jp        tim_int
        jp        BC_int
        jp        A_int
        .org      0ffch
        jp        nmi_int
        jp        inizio

        .end
```

LISTATO del programma PCONT.ASM

```
;* Programma per fare un conteggio *

        .title    "PCONT"
        .vers     "ST62E25"
        .w_on
        .romsize  4
        .pp_on
        .input    "ST62X.DEF"

;VARIABILI usate da questo PROGRAMMA

del1    .def      084h
del2    .def      085h
up_dw   .def      086h
lsb     .def      087h
msb     .def      088h

        .extern decrem,increm,mulplx
        .section 1

inizio
        ldi       wdog,0ffh

        ldi       port_a,00000000b
        ldi       pdir_a,00001100b
        ldi       popt_a,00001100b

        ldi       port_b,00000000b
        ldi       pdir_b,11111111b
        ldi       popt_b,11111111b

        ldi       port_c,00000000b
        ldi       pdir_c,00000000b
        ldi       popt_c,00000000b

;***   Disabilita gli Interrupt

        ldi       adcr,0
        ldi       tscr,0
        ldi       ior,0

        reti

        jp        main

;***     GESTORI di INTERRUPT      ***

ad_int   reti
tim_int  reti
BC_int   reti
A_int    reti
nmi_int  reti
```

Fig.2 Listato del programma PCONT.ASM. Per compilarlo in formato oggetto .OBJ abbiamo dovuto inserire le direttive evidenziate in giallo.

```
;***      PROGRAMMA PRINCIPALE     ***

main
        ldi       wdog,0feh
        ldi       lsb,0
        ldi       msb,0
        ldi       up_dw,1
        ldi       drw,digit.w

loop    ldi       del1,17
main1   ldi       del2,255
main2   ldi       wdog,0feh

        call      mulplx

        dec       del2
        jrz       main3
        jp        main2
main3   dec       del1
        jrz       main6

        jrs       0,port_a,main4
        ldi       up_dw,0
main4   jrs       1,port_a,main5
        ldi       up_dw,1
main5
        jp        main1
main6
        ld        a,up_dw
        cpi       a,0
        jrz       main7

        call      increm
        jp        loop

main7   call      decrem
        jp        loop

;*** tabella con i segmenti per far
; apparire sui display le cifre ***

        .window
digit   .byte     192,249,164,176,153
        .byte     146,130,248,128,144
        .windowend

;*** VETTORI DI INTERRUPTS ***

        .section 32
        jp        ad_int
        jp        tim_int
        jp        BC_int
        jp        A_int
        .block    4
        jp        nmi_int
        jp        inizio

        .end
```

LISTATO del programma PLEXER.ASM

```
;* Modulo per gestire un multiplexer
;* a due cifre

        .title    "PLEXER"
        .vers     "ST62E25"
        .w_on
        .romsize  4
        .pp_on
        .input    "ST62X.DEF"

;VARIABILI usate da questo PROGRAMMA

lsb     .def      084h
msb     .def      085h
        .section  1

;***          SUBROUTINE              ***

;- multiplexa le 2 cifre sui display
mulplx
        ld        a,lsb
        addi      a,40h
        ld        x,a
        ld        a,(x)
        ldi       port_a,00001100b
        ld        port_b,a
        ldi       port_a,00000100b

        ld        a,msb
        addi      a,40h
        ld        x,a
        ld        a,(x)
        ldi       port_a,00001100b
        ld        port_b,a
        ldi       port_a,00001000b

        ret
```

```
;- incremento delle 2 cifre
;- con controlli
increm
        inc       lsb
        ld        a,lsb
        cpi       a,10
        jrnz      incr1
        ldi       lsb,0
        inc       msb
        ld        a,msb
        cpi       a,10
        jrnz      incr1
        ldi       msb,0
incr1
        ret



;- decremento delle 2 cifre
;- con controlli
decrem
        ld        a,lsb
        cpi       a,0
        jrnz      decr1
        ldi       lsb,9

        ld        a,msb
        cpi       a,0
        jrnz      decr2
        ldi       msb,9
        ret

decr2   dec       msb
        ret

decr1   dec       lsb
        ret
```

Fig.3 Listato del programma PLEXER.ASM. Queste istruzioni sono state tratte dal programma CONTA.ASM (vedi in fig.1 le istruzioni evidenziate in azzurro), e per compilarle in formato .OBJ abbiamo inserito le direttive evidenziate in giallo.

A questo punto abbiamo due programmi, **PLEXER.ASM** e **PCONT.ASM**, che dobbiamo modificare e compilare separatamente per ottenere rispettivamente **PCONT.OBJ** e **PLEXER.OBJ**.
Vedremo così come, linkando questi programmi, si ottenga un terzo programma in formato **.HEX**.

Per generare in formato **.OBJ** il programma **PCONT**, abbiamo dovuto modificare il listato come visibile in fig.2. Per generare in formato **.OBJ** il programma **PLEXER**, abbiamo dovuto modificare il listato come visibile in fig.3.
In entrambe le figure abbiamo evidenziato in **giallo** le direttive inserite e ora analizzeremo nei dettagli queste modifiche via via che le incontreremo.

La direttiva .pp_on

Rispetto al programma originario, e cioè CONTA.ASM, nel programma **PCONT.ASM** dopo la direttiva **.romsize 4** abbiamo inserito la direttiva **.pp_on**, che abilita la paginazione della memoria del micro.

Normalmente questa direttiva va inserita quando si realizzano programmi per i microprocessori ST6 che dispongono di più di **4 kbyte** di memoria **Program Space** (**ROM**).

In questi modelli di micro infatti, esiste una memoria ROM che possiamo definire **primaria** di 4096

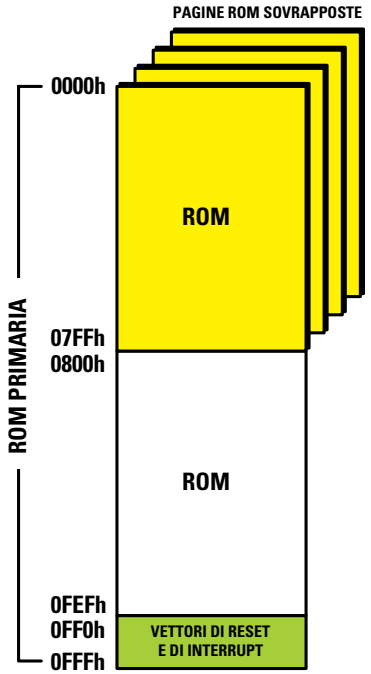


Fig.4 I micro con più di 4 kbytes di memoria Program Space hanno una ROM primaria suddivisa in tre pagine di grandezza diversa: da 0000h a 07FFh, da 0800h a 0FEFh e da 0FF0h a 0FFFh. Inoltre hanno un massimo di 30 pagine sovrapposte con locazione da 0000h a 07FFh.

bytes che va da locazione 0000h a 0FFFh e un massimo di 30 "pagine" sovrapposte di 2048 bytes di area ROM, tutte con locazione da 0000h a 07FFh, come visibile nel disegno di fig.4.

La stessa memoria **primaria** viene ulteriormente suddivisa in tre **pagine** di area ROM di grandezza diversa: la prima ha locazione 0000h – 07FFh, la seconda ha locazione 0800h – 0FEFh e la terza ha locazione 0FF0h – 0FFFh.

A ciascuna per comodità viene virtualmente associato un numero di pagina che va da 0 a 32 (vedi tabella di fig.5) e ogni pagina deve essere indirizzata tramite un'altra direttiva chiamata **.section**.
Utilizzando **.pp_on**, e quindi segnalando al compilatore che deve virtualmente suddividere la memoria in pagine, bisognerà utilizzare anche la direttiva **.section** che serve a indirizzare queste pagine.

Nel nostro esempio noi utilizziamo un micro ST62E25 che non supera i 4 kbyte di memoria (vedi **.vers** in fig.2), ma volendo generare un programma in formato **.OBJ** siamo obbligati ad inserire la direttiva **.section** e, di conseguenza, a definire anche **.pp_on**, altrimenti il compilatore segnalerà errore.

La direttiva .extern

Sempre rispetto al programma **CONTA**, la successiva istruzione che abbiamo inserito nel programma **PCONT** è la direttiva **.extern** con a fianco l'indicazione di tre etichette:

.extern decrem,increm,mulplx

La direttiva **.extern** va obbligatoriamente inserita ogniqualvolta si intende assemblare in formato **.OBJ** un programma contenente istruzioni che richiamano o saltano a labels di routine che non si trovano all'interno del programma stesso, ma sono inserite in altri programmi. In questo modo si avverte il compilatore di non segnalare errore quando non trova le routine richiamate.

Nel programma **PCONT** (vedi listato in fig.2) ci sono infatti tre routine chiamate con le istruzioni:

```
call mulplx
call increm
call decrem
```

che **non** vengono assolutamente **definite**, perché inserite nel programma **PLEXER** (vedi fig.3).

Inserendo la direttiva **.extern**, abbiamo avvertito il compilatore che le routine sono **esterne**, e che quindi non deve segnalare errore quando incontra le istruzioni che le richiamano.

Per avere una riprova di ciò, abbiamo provato a togliere l'istruzione:

.extern decrem,increm,mulplx

e abbiamo compilato **PCONT**.
In fig.6 è visibile la segnalazione di errore del compilatore, in cui queste tre etichette vengono indicate come "undefined symbol".

E' importante annotare che quando si utilizza questa direttiva per definire **labels** di **routine esterne** al programma, conviene sempre porla all'inizio così da rendere visibile già in fase di edit, che il programma contiene salti o richiami a routine esterne.

Possono essere definite come .**extern** solamente labels di Program Space e di Data Rom Windows.

Non possono essere definite come .**extern** i registri, le variabili di **Data space** o le costanti (**.def**, **.equ**, **.set**).

Pagina N°	INDIRIZZO VIRTUALE	INDIRIZZO REALE
0	0000 - 07FF	0000 - 07FF
1	0800 - 0FEF	0800 - 0FEF
2	1000 - 17FF	0000 - 07FF
3	1800 - 1FFF	0000 - 07FF
da 4 a 31	[n*800]-[(9n*80)+7FF]	0000 - 07FF
32	0FF0 - 0FFF	0FF0 - 0FFF

Fig.5 A ogni pagina di memoria, che ha un suo preciso indirizzo reale, viene associato per comodità un indirizzo virtuale rappresentato da un numero da 0 a 32.

Per finire, questa direttiva può essere inserita solamente nei programmi che verranno compilati con l'opzione **–O**. In caso contrario il compilatore segnalerà errore.

La direttiva .section

Confrontate ancora il programma **PCONT.ASM** di fig.2 all'altezza della label **inizio** con il programma **CONTA.ASM** di fig.1 alla stessa altezza.
Nel programma originale **CONTA.ASM** prima della label **inizio** avevamo inserito l'istruzione **.org 0800h**, mentre in **PCONT.ASM** l'abbiamo sostituita con **.section 1**.

Quando si assembla un programma in formato **.OBJ** si deve sostituire la direttiva **.org** con la direttiva **.section** seguita da un numero da **0** a **32**, altrimenti verrà segnalato errore.

In relazione a quanto detto precedentemente a proposito della direttiva **.pp_on**, che attiva la "paginazione" o, se preferite, la suddivisione in pagine della memoria ROM, inserendo nel programma la direttiva **.section** noi indichiamo al compilatore in quale "pagina" di memoria ROM deve inserire le istruzioni del programma da compilare in formato **.OBJ**.

Nel nostro caso, noi indichiamo al compilatore che le istruzioni del programma **PCONT** devono essere inserite nella Program Space di pagina **1**, e cioè nell'area ROM con locazione **0800h – 0FEFh** come visibile in fig.7.

All'interno dello stesso programma è possibile inserire più direttive **.section** per indirizzare pagine diverse ed inserire perciò le istruzioni in punti diversi di Program Space.
Poiché però vi sono alcune limitazioni sull'utilizzo delle istruzioni di salto da una pagina di memoria all'altra, bisogna fare attenzione alle caratteristiche di "salto" legate al numero di pagina utilizzato.

In fig.7 riportiamo la tabella illustrativa di queste limitazioni.

Nella colonna "**salto a...**", in corrispondenza delle righe di pagina **1** e di pagina **32** è indicato **tutte le pagine**, mentre nelle restanti è indicato **pagina 1**. Questo significa che nelle pagine **1** e **32** di Program Space si possono inserire istruzioni di salto incondizionato (**jp, call**) a tutte le altre pagine di memoria, mentre nelle pagine **0** e da **2** a **31** si possono inserire solamente istruzioni di salto incondizionato alla pagina **1**.

```
C:\ST6\LX1208>ast6 -l -o PCONT.ASM
ST6 MACRO-ASSEMBLER version 4.00  - August 1992
Error  PCONT.ASM 94: (106) undefined symbol: decrem
Error  PCONT.ASM 91: (106) undefined symbol: increm
Error  PCONT.ASM 72: (106) undefined symbol: mulp1x
Execution time: 0 second(s)
3 errors detected
No object created
```

Fig.6 Errore segnalato dal compilatore quando non si usa correttamente la direttiva .extern.

Pagina N°	INDIRIZZO VIRTUALE	INDIRIZZO REALE	SALTO A
0	0000 - 07FF	0000 - 07FF	Pagina 1
1	0800 - 0FEF	0800 - 0FEF	tutte le Pag.
2	1000 - 17FF	0000 - 07FF	Pagina 1
3	1800 - 1FFF	0000 - 07FF	Pagina 1
da 4 a 31	[n*800]-[(9n*80)+7FF]	0000 - 07FF	Pagina 1
32	0FF0 - 0FFF	0FF0 - 0FFF	tutte le Pag.

Fig.7 Esistono delle limitazioni sull'utilizzo dell'istruzione di salto da una pagina di memoria all'altra, per cui nelle pagine 0 e da 2 a 31 si possono inserire solo istruzioni di salto a pagina 1.

Facciamo un esempio. Compilando il programma:

```
.section 1
inizio  .................
        .................
        .................
        jp  letsta
rien1   .................
        .................
.section 2
letsta  ldi  a,23
        .................
        .................
        call  storx
        .................
.section 3
storx   addi  a,23
        .................
        ret
```

non verrà segnalato errore, perché le istruzioni sono formalmente corrette.
Quando però tenteremo di **linkare** questo programma, il **linker** segnalerà un errore simile a quello di fig.8, perché non sono state rispettate le condizioni. Infatti, da **pagina 1** con l'istruzione **jp letsta** si può passare alla **pagina 2**, ma poi l'istruzione **call storx** non può essere eseguita perché **storx** si trova nella **pagina 3**.

La giusta sequenza del nostro esempio è dunque la seguente:

```
.section 1
inizio  .................
        .................
        .................
        jp  letsta
rien1   call  storx
        .................
.section 3
letsta  ldi  a,23
        .................
        .................
        jp  rien1
.section 4
storx   addi  a,23
        .................
        ret
```

L'esempio appena riportato si riferiva a più **section** inserite in un unico programma, ma è evidente che si pone un problema analogo quando diverse **.section** sono inserite in più programmi che andranno concatenati con il **linker**.

Chiusa questa parentesi, torniamo al listato di **PCONT** (vedi fig.2) e soffermiamoci sull'istruzione **.section 32** e sulla successiva **.block 4**.

```
        reference to <increm> external
        reference to <mulplx> external
lst6: ** illegal jump inside program section #2, offset 0x0, file <PCONT.OBJ>
```

Fig.8 Il controllo sul rispetto delle condizioni necessarie all'esecuzione dell'istruzione di salto viene fatto dal programma linker Lst6. In questa figura è segnalato errore perché l'istruzione di salto da pagina 2 può essere eseguita solo verso pagina 1 (vedi fig.7), e non a pagina 3 come scritto nel programma a sinistra sopra questa figura.

Nelle stesse righe del programma originale **CONTA.ASM** vi erano le istruzioni **.org 0FF0h** e **.org 0FFCh**
Con **.section 32** si attiva la pagina di memoria relativa alla gestione dei vettori di reset e di interrupt. La direttiva **.block 4** sostituisce **.org 0FFCh**, ma ha la stessa funzione di posizionare correttamente i vettori di **nmi** e di **reset**.

Le direttive .window e .windowend

Mettendo ancora una volta a confronto le righe del programma originale CONTA con quelle di **PCONT**, potete vedere che l'istruzione:

.block 64-$%64

è stata sostituita dalla direttiva **.window**, mentre dopo il secondo **.byte** è stata inserita la direttiva **.windowend**.

Come abbiamo avuto occasione di ripetere più volte (vedi soprattutto la rivista **N.190**), normalmente l'istruzione **.block 64-$%64** precede l'inserimento di dati in Program Space (**.byte, .ascii**, **.asciz**) che verranno caricati a blocchi di 64 bytes tramite la Data Rom Windows.

```
PROGRAM SECTIONS:

number   start    end      size
------   -----    ---      ----
1        0800     0F9F     00D3
32       0FF0     0FFF     0010


MODULE PCONT.OBJ:

section     type     start     size
-------     ----     -----     ----
1           P        0800      008A
32          P        0FF0      0010


MODULE PLEXER.OBJ:

section     type     start     size
-------     ----     -----     ----
1           P        088A      0049
```

Fig.9 Mappa della memoria risultante dal link ottenuto con PCONT.OBJ e PLEXER.OBJ. Non avendo inserito le direttive .window e .windowend i due programmi sono stati accodati.

Compito principale di **.block 64-$%64** è di "ottimizzare" l'utilizzo di Program Space.

Compilando il programma **PCONT** in formato **.OBJ** avremmo anche potuto lasciare l'istruzione **.block 64-$%64**, però i dati definiti con le due direttive **.byte** sarebbero stati allocati con allineamento al primo blocco di **64** byte di **Program Space** successivo all'ultima istruzione di **PCONT** e cioè **jp loop**. Linkando i due programmi **PCONT.OBJ** e **PLEXER.OBJ**, il **linker** avrebbe "accodato" al programma **PCONT** le istruzioni del programma **PLEXER**, che quindi si sarebbero venute a trovare dietro a quest'area dati.
Avremmo pertanto avuto un programma finale **.HEX** non bene ottimizzato, sia come utilizzo di memoria Program Space sia come "leggibilità".

Per provarvi quanto detto, abbiamo linkato **PLEXER.OBJ** e **PCONT.OBJ** lasciando al suo interno l'istruzione **.block 64-$%64** e senza inserire la direttiva **.windowend**.

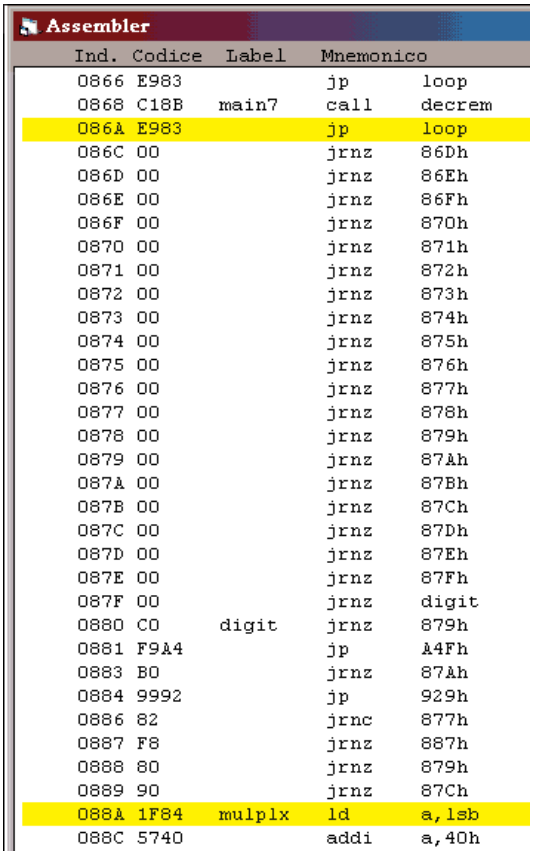
In fig.9 potete vedere la mappa della memoria del programma **.HEX** risultante.
Il programma **PCONT.OBJ** (vedi **Module**) inizia all'indirizzo di memoria **0800h** e termina all'indirizzo **0800h + 008Ah**, cioè a **088Ah**, mentre il programma **PLEXER.OBJ** inizia proprio da **088Ah** e termina a **088Ah + 0049h**, cioè a **08D3**.

Abbiamo poi simulato l'esecuzione del programma **.HEX** con un Simulatore Software e in fig.10 potete avere la riprova di quanto affermato poco sopra.

In alto è evidenziata l'ultima istruzione eseguibile di **PCONT** e cioè **jp loop** seguita da una serie di istruzioni **jrnz** che indirizzano sempre al byte successivo. Questo è il risultato dell'inserimento dell'istruzione **.block 64-$%64** che il compilatore traduce appunto in tanti salti di **1 byte** fino a quando non arriva ad un blocco di memoria divisibile esattamente per 64.
Infatti, quasi in fondo alla figura compare la label **digit** che identifica il punto di memoria esatto in cui sono stati inseriti i dati con i **.byte** e alla sua sinistra compare l'indirizzo di memoria relativo e cioè **0880h** che è appunto un indirizzo divisibile esattamente per **64**.

Spostate lo sguardo più sotto e nella riga evidenziata vedrete l'istruzione **mulplx ld a,lsb**, che è la prima istruzione del programma **PLEXER** e si trova effettivamente all'indirizzo di memoria **088Ah**.

Vediamo invece cosa succede inserendo **.window** al posto di **.block 64-$%64** e aggiungendo **.win-**



```
Assembler
    Ind. Codice  Label   Mnemonico
    0866 E983            jp      loop
    0868 C18B    main7   call    decrem
    086A E983            jp      loop
    086C 00              jrnz    86Dh
    086D 00              jrnz    86Eh
    086E 00              jrnz    86Fh
    086F 00              jrnz    870h
    0870 00              jrnz    871h
    0871 00              jrnz    872h
    0872 00              jrnz    873h
    0873 00              jrnz    874h
    0874 00              jrnz    875h
    0875 00              jrnz    876h
    0876 00              jrnz    877h
    0877 00              jrnz    878h
    0878 00              jrnz    879h
    0879 00              jrnz    87Ah
    087A 00              jrnz    87Bh
    087B 00              jrnz    87Ch
    087C 00              jrnz    87Dh
    087D 00              jrnz    87Eh
    087E 00              jrnz    87Fh
    087F 00              jrnz    digit
    0880 C0      digit   jrnz    879h
    0881 F9A4            jp      A4Fh
    0883 B0              jrnz    87Ah
    0884 9992            jp      929h
    0886 82              jrnc    877h
    0887 F8              jrnz    887h
    0888 80              jrnz    879h
    0889 90              jrnz    87Ch
    088A 1F84    mulplx  ld      a,lsb
    088C 5740            addi    a,40h
```

Fig.10 L'istruzione .block 64-$%64 è stata tradotta dal compilatore in salti di 1 byte fino ad un blocco di memoria divisibile per 64. Infatti i dati .byte, identificati dalla label digit, vengono inseriti all'indirizzo 0880h, che è divisibile per 64.

dowend. Ricompiliamo in Assembler il programma **PCONT** in formato **.OBJ** con il comando:

ast6 –L –O PCONT.ASM.

Abbiamo inserito anche l'opzione **–L** perché vogliamo generare anche **PCONT.LIS**.
Quando il compilatore incontra la direttiva **.window** prosegue fino a che non trova **.windowend** (che deve sempre essere inserita) e "memorizza" i dati (**.byte**, **.ascii**, ecc.) definiti tra questi estremi in una area rilocabile particolare definita come **Window section**.

In fig.11 abbiamo riprodotto la parte del file **PCONT.LIS** che riguarda queste direttive.
All'altezza della riga **119** potete notare la scritta **W00** che appunto rappresenta l'assegnazione alla Window section dei nostri 10 byte di data space identificati dalla label **digit**, visibili a destra nella stessa riga.
Notate inoltre che a fianco di **W00** c'è il numero **0000**: normalmente questo numero rappresenta la locazione di memoria in cui verrà memorizzata l'istruzione e in questo caso i nostri 10 bytes verranno "memorizzati" a partire dall'indirizzo **0** della **Window section**.

A questo punto possiamo linkare **PCONT.OBJ** e **PLEXER.OBJ** per ottenere l'eseguibile **.HEX** e in fig.12 riportiamo la mappa di memoria risultante.
Notate subito che rispetto alla mappa precedente (vedi fig.9) vi è una **Window section** che inizia a **08B5h** ed è lunga **000Ah** (cioè i 10 byte di **digit**).
La stessa Window section è poi richiamata più in basso, nel programma **PCONT.OBJ**, nella terza riga della seconda colonna (vedi type W).

```
 113 S01 0068 0100    S01 0068        91  main7   call    decrem
 file PCONT.lis  page 4

PCONT
 114 S01 006A E903    S01 006A        92          jp      loop
 115                                  93
 116                                  94  ;**********************************
 117                                  95  ;tabella con i segmenti per far appa
 118                                  96          .window
 119 W00 0000 C0      W00 0000        97  digit   .byte   192,249,164,176,153
 120 W00 0001 F9      W00 0001        97
 121 W00 0002 A4      W00 0002        97
```

Fig.11 Ricompilando il programma PCONT dopo aver inserito le direttive .window e .windowend, il compilatore "memorizza" le istruzioni racchiuse tra queste due direttive in un'area rilocabile definita window section: notate la scritta W00 all'altezza della riga 119. Il numero che segue (0000) rappresenta la locazione di memoria in cui vengono memorizzate le istruzioni racchiuse tra le direttive .window e .windowend.

```
WINDOW SECTIONS:

number  start   end     size
------  -----   ---     ----
0       08B5    08BE    000A


MODULE PCONT.OBJ:

section     type     start     size
-------     ----     -----     ----
1           P        0800      006C
32          P        0FF0      0010
0           W        08B5      000A


MODULE PLEXER.OBJ:

section     type     start     size
-------     ----     -----     ----
1           P        086C      0049
```

Fig.12 Mappa della memoria risultante dal link dei programmi PCONT.OBJ e PLEXER.OBJ dopo aver inserito le direttive .window e .windowend. Rispetto alla fig.9, c'è una window section lunga esattamente 10 byte (000Ah), il cui inizio non è più a 0000, ma a 08B5, perché il linker ha posizionato la window section in coda a tutte le istruzioni.

Notate però che l'indirizzo della Window section non è più 0000 come era in PCONT.LIS di fig.11 ma è diventato come già detto **08B5h**.
Il linker infatti ha unito in sequenza le istruzioni dei programmi **PCONT** e **PLEXER** e solo dopo, in coda a tutto, ha "rilocato" la Window section.

Per fare questo calcola innanzitutto la grandezza dell'area dati che si vuole inserire in Program Space tramite Window section (nel nostro esempio **digit** sono 10 bytes), poi si posiziona alla prima locazione di Program Space divisibile esattamente per 64 successiva all'ultima istruzione del programma finale .HEX.

Se la differenza fra questa locazione e quella relativa all'ultima istruzione del programma è maggiore della grandezza dell'area dati da inserire (**digit**), inserisce i dati prima di questa locazione (vedi fig.13), se invece è minore, li inserisce dopo (vedi fig.14).

Per concludere, con le direttive **.window** e **.windowend**, i dati da inserire in Program Space vengono automaticamente posizionati in coda a tutte le istruzioni, in un area già ottimizzata evitando così inutili sprechi di memoria e soprattutto predisponendoli ad essere caricati in maniera corretta nella Data Rom Window.

Nota: nella rivista **N.190** abbiamo spiegato il corretto utilizzo della Data Rom Window.

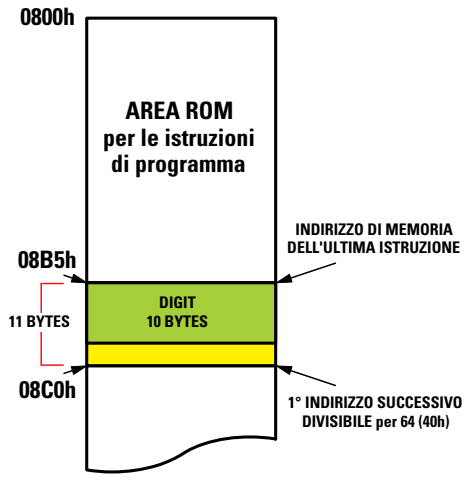


Fig.13 Se la differenza tra la prima locazione di memoria ROM divisibile per 64 e la locazione dell'ultima istruzione è maggiore dell'area window section, i dati vengono inseriti prima di questa locazione.

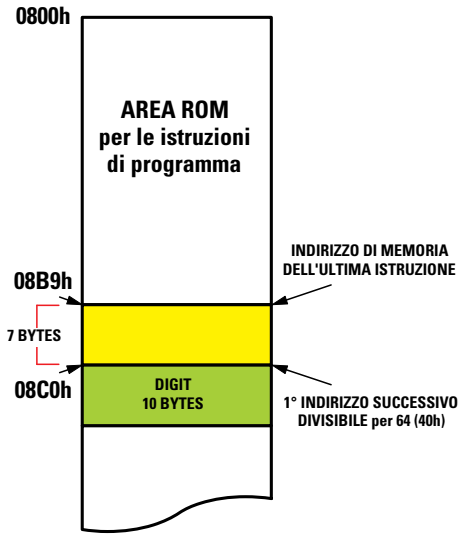


Fig.14 Se la differenza tra la prima locazione di memoria ROM divisibile per 64 e la locazione dell'ultima istruzione è minore dell'area window section, i dati vengono inseriti dopo questa locazione.

L'unica restrizione all'uso di queste direttive è che tra **.window** e **.windowend** si possono inserire un massimo di 64 byte di dati altrimenti il compilatore Assembler segnalerà questo errore:

Error current program section overflow (fatal)

In questo caso dovrete spezzare i vostri dati in blocchi di massimo 64 byte e utilizzare più volte le direttive **.window .windowend** come riportato nell'esempio che segue:

```
        .window
dig01   .byte           .............
        .................
        .windowend
        .window
dig02   .ascii          .............
        .byte           .............
        .windowend
```

A questo punto abbiamo terminato l'analisi del programma **PCONT.ASM** e possiamo dedicarci al programma **PLEXER.ASM**.

Innanzitutto potete notare che davanti alle istruzioni che abbiamo estratto dal programma originale **CONTA.ASM** (vedi il listato in fig.3), sono state inserite le direttive necessarie al programma **PLEXER** per essere compilato:

```
        .title      "plexer"
        .vers       "st62e25"
        .w_on
        .romsize    4
```

Abbiamo quindi aggiunto la direttiva **.pp_on** (vedi riga evidenziata in giallo) che, come abbiamo già detto, attiva la "paginazione" e abbiamo ripetuto la definizione dei registri con la direttiva **.input** e la definizione delle variabili **lsb** e **msb**, che avevamo già definito in **PCONT.ASM**.

Questa ripetizione si è resa necessaria dal momento che queste variabili e alcuni registri vengono utilizzati nel programma e perciò se non li avessimo segnalati, il compilatore avrebbe dato errore e non avrebbe compilato il programma nel formato oggetto **.OBJ**.

Se state pensando che si potevano evitare queste istruzioni definendo i registri e le variabili come esterni con la direttiva **.extern**, siete in errore.

Come infatti abbiamo già detto, ma forse è utile ripetere, possono essere **definite** come **.extern** solamente **labels** di Program Space e di Data Rom Windows, mentre i registri, le variabili di **Data space** e le costanti (**.def**, **.equ**, **.set**) **non** possono essere **definite** con questa direttiva.

E' però importante farvi notare che le variabili **lsb** e **msb** sono state definite ad un indirizzo di memoria differente da quello che avevano nel programma **PCONT.ASM** (vedi di fig.2).
Torneremo più avanti su questo argomento.

In conclusione sottolineiamo che anche in questo programma è stata inserita la direttiva **.section 1**, che oramai conoscete.

A questo punto assembliamo in formato **.OBJ** i programmi **PCONT.ASM** e **PLEXER.ASM** digitando al prompt di DOS:

```
ast6 –L –O PCONTA.ASM
ast6 –L –O PLEXER.ASM
```

Otteniamo così **PCONTA.OBJ** e **PLEXER.OBJ**, che ora possiamo "unire" con il linker **lst6** per ottenere un programma eseguibile al quale diamo nome **XCONTA.HEX**.

Ottenere il formato .HEX con il linker lst6

Finora abbiamo sempre parlato di "unire" più programmi **.OBJ** per ottenere un programma eseguibile **.HEX**.

In realtà è meglio utilizzare il termine **collegare**, perché i programmi vengono collegati insieme e ogni indirizzo di memoria, che prima era relativo ad un singolo programma, diventa **indirizzo assoluto** nel programma finale **.HEX**.

A questo punto penserete che essendo i programmi correttamente compilati in formato **.OBJ**, linkandoli non incontreremo alcun ostacolo.

In realtà le cose non stanno proprio così, ma poiché non sarebbe utile anticipare i problemi, vediamo per ora come si lancia il linker Lst6 per collegare **PCONT.OBJ** e **PLEXER.OBJ** e ottenere **XCONTA.HEX**.
Al prompt di DOS digitiamo:

```
lst6 –S –I –T –V –M –O XCONTA PCONT PLEXER
```

Le scritte **–S –I –T –V –M** sono opzioni specifiche del linker Lst6 che verranno spiegate in maniera completa nel prossimo articolo.
Per non appesantire questo articolo, ci soffermiamo solo su **–O XCONTA PCONT PLEXER**.

Nota: attenzione a non confondere l'opzione **–O** del linker con l'opzione **–O** dell'Assembler.

L'opzione **–O** del linker seguita dal nome del programma finale, nel nostro caso **XCONTA**, serve ad indicare al linker come dovrà chiamare il programma eseguibile **.HEX**.

Come potete notare noi ci siamo limitati a scrivere **XCONTA**, perché l'estensione **.HEX** viene messa automaticamente dal programma **lst6**.

Se avessimo voluto ottenere un programma con una diversa estensione avremmo dovuto scrivere il nome per esteso: ad esempio **–O XCONTA.PGM**.

Dopo il nome dell'eseguibile, scriviamo in successione il nome dei programmi da concatenare, cioè **PCONT** e **PLEXER**, omettendo anche stavolta l'estensione **.OBJ**, perché assunta di default.

E' invece molto **IMPORTANTE** l'ordine in cui vengono definiti i programmi da linkare, perché il linker seguirà quell'ordine per collegarli.

Nel nostro esempio i programmi sono due, ma potrebbero essere molti di più.

CONTROLLO delle CONDIZIONI

Lanciamo quindi il linker e, come già anticipato, a video compaiono le segnalazioni di **errore** visibili in fig.15. Dopo la visualizzazione della versione del Linker e la segnalazione del copyright c'è la scritta:

pass1 :

Il linker o, come più correttamente sarebbe giusto chiamarlo, il Linkage Editor, agisce infatti in due fasi o passi.
Il primo passo o **pass1** è quello di controllare che in tutti i programmi **.OBJ** da linkare ci siano le condizioni per poterli collegare segnalando eventuali errori.
Il secondo passo o **pass2** è quello specifico di collegare ogni singola istruzione e locazione di memoria dei vari programmi in modo da ottenere un unico programma eseguibile. E' in questa seconda fase che le locazioni di memoria dei singoli programmi vengono in un certo senso sistemate una in "coda" all'altra con la conseguente "rilocazione" o "rimappatura" degli indirizzi.

In tutti gli articoli sul linguaggio di programmazione Assembler usato dai microprocessori ST6, vi abbiamo sempre spiegato le procedure per scrivere i programmi unendo la teoria, della quale non si può fare a meno, alla pratica, con esempi che fossero semplici e immediati. Chi ha avuto la costanza di seguirci non ne è rimasto deluso e con questo articolo sul linker Lst6 potrà acquisire ulteriori elementi per programmare senza problemi.

Sotto **pass 1** leggiamo:

```
<PCONT.obj>: program section(s) size is 0x7C (124), window(s) size is 0xA (10)
```

Il linker calcola e segnala l'occupazione di Program Space (**124** bytes) e l'occupazione di window section (**10** bytes: ricordate la definizione di **digit** tra **.window** e **.windowend**) del programma **PCONT**.

Di seguito c'è:

```
<PLEXER.obj>: program section(s) size is 0x49 (73), window(s) size is 0x0 (0)
```

Il calcolo della memoria di Program Space (**73** bytes) e l'eventuale presenza di window section, avviene anche per il programma **PLEXER**.

Nelle tre righe seguenti leggiamo:

```
lst6 : ** undefined symbol <decrem> ; first referenced in file  <PCONT.obj>
lst6 : ** undefined symbol <increm> ; first referenced in file  <PCONT.obj>
lst6 : ** undefined symbol <mulplx> ; first referenced in file  <PCONT.obj>
lst6 : <3> fatal error(s) occurred
```

Effettuando un controllo sulla possibilità di collegare **PCONT** e **PLEXER**, il linker rileva tre anomalie relative alle routine identificate dalle labels **decrem**, **increm** e **mulplx** e pertanto termina senza generare il programma eseguibile.
Segnala queste routine come indefinite (undefined symbol) e ci informa che il primo riferimento (first referenced) è nel programma PCONT.

La prima cosa che dobbiamo fare è controllare il programma **PCONT.ASM** dove però le tre labels sono state correttamente definite **esterne** con la direttiva **.extern decrem,increm,mulplx**.

A questo punto controlliamo anche il programma **PLEXER.ASM**, ma anche qui **decrem**, **increm** e **mulplx** sono definite e usate correttamente.
Dovrebbe perciò essere tutto a posto, ma nonostante ciò il linker le segnala come indefinite.

L'errore deriva dal fatto che nel programma **PLEXER** non è stata inserita la direttiva **.global**.

```
C:\>LST6 -S -I -T -V -M -O XCONTA PCONT PLEXER

ST6 Linkage Editor  version 3.40
Copyright (C)  SGS-THOMSON Microelectronics  May 1995

pass1:
<PCONT.obj>: program section(s) size is 0x7C (124), window(s) size is 0xA (10)
<PLEXER.obj>: program section(s) size is 0x49 (73), window(s) size is 0x0 (0)
lst6: ** undefined symbol <decrem> ; first referenced in file  <PCONT.obj>
lst6: ** undefined symbol <increm> ; first referenced in file  <PCONT.obj>
lst6: ** undefined symbol <mulplx> ; first referenced in file  <PCONT.obj>
lst6: <3> fatal error(s) occured
```

Fig.15 Il programma linker agisce in due fasi o passi. Nella prima fase controlla se nei programmi .OBJ da linkare ci sono i presupposti per il loro collegamento. In questo caso non passa alla seconda fase perché rileva delle anomalie sull'uso delle labels decrem, increm e mulplx segnalandoci che il loro primo riferimento si trova in PCONT.OBJ.

La direttiva .global

Questa direttiva è assolutamente ininfluente in fase di compilazione in formato **.OBJ** e la prova è data dal fatto che il compilatore non ha segnalato nessun errore assemblando **PLEXER.ASM**.

Quando però si devono linkare programmi che contengono la direttiva **.extern** per segnalare l'utilizzo di routine esterne, nel programma che effettivamente contiene queste routine bisogna inserire la direttiva **.global** seguita dalla definizione delle labels di queste routine.

In questo modo segnaliamo al linker che queste routine sono richiamate in altri programmi e, in un certo senso, le rendiamo "disponibili".
E' importante ricordare che **.global** deve essere **obbligatoriamente** inserita prima della definizione delle routine che vogliamo rendere utilizzabili in altri programmi.

Il listato visibile in fig.3 va perciò modificato inserendo nel programma **PLEXER.ASM**, prima di **.section 1** l'istruzione:

.global decrem,increm,mulplx

Ovviamente il programma va ricompilato per generare **PLEXER.OBJ** e poi va rilanciato il linker.

RILOCAZIONE degli INDIRIZZI

Nella fig.16 abbiamo riportato la videata che appare dopo aver lanciato per la seconda volta il linker.

Questa volta sotto **pass1** non vengono segnalati errori, ma appare: **window #0 (10 bytes) mapped in program page #1, at offset 0xb5**.

Questa scritta ci informa che, grazie alle direttive **.window** e **.windowend** inserite in **PCONT**, il linker ha rilocato (mapped) all'indirizzo **0B5h** di Program page 1 un'area dati di **10** bytes.

Il linker passa quindi alla seconda fase e ne dà il resoconto sotto la scritta:

pass2 :

Il collegamento vero e proprio di **PCONT** e **PLEXER** è stato effettuato e segnala che in **PCONT** ha rilevato l'utilizzo delle tre routine esterne e che in **PLEXER** ha rilevato le stesse routine definite con **.global** e ha assegnato loro un indirizzo assoluto di memoria Program Space:

decrem	**89Eh**
increm	**889h**
mulplx	**86Ch**

```
ST6 Linkage Editor  version 3.40
Copyright (C)  SGS-THOMSON Microelectronics  May 1995

pass1:
<PCONT.obj>: program section(s) size is 0x7C (124), window(s) size is 0xA (10)
<PLEXER.obj>: program section(s) size is 0x49 (73), window(s) size is 0x0 (0)
window #0 (10 bytes) mapped in program page #1, at offset 0xb5
pass2:
    <PCONT.obj>
        reference to <decrem> external
        reference to <increm> external
        reference to <mulplx> external
    <PLEXER.obj>
        definition of <decrem> global program  89E(2206)
        definition of <increm> global program  889(2185)
        definition of <mulplx> global program  86C(2156)
program section(s) size is 0xCF (207)
<XCONTA.hex>: hexadecimal image
<XCONTA.dsd>: dsd file
<XCONTA.sym>: namelist
```

Fig.16 Il linker dà un resoconto scritto anche della 2° fase, che consiste nel collegare ogni singola istruzione e locazione di memoria così da ottenere un eseguibile .HEX.

Inoltre segnala che la grandezza del programma eseguibile sarà di **0CFh bytes** di Program Space e cioè di 207 bytes a partire da program section 1, e cioè dall'indirizzo di memoria **0800h** (vedi la tabella in fig.5). Infine segnala che ha generato:

XCONTA.hex
XCONTA.dsd
XCONTA.sym

Nota: non ci soffermiamo sulle peculiarità dei programmi con estensione **.dsd** e **.sym** ai quali abbiamo dedicato l'articolo apparso sulla rivista **N.194**.

Questa volta il linkaggio è andato a buon fine quindi non ci resta che effettuare una semplice prova di simulazione per verificare se **XCONTA.HEX** funziona correttamente.

Se vi ricordate, in entrambi i programmi **PCONT** e **PLEXER** avevamo definito le variabili **lsb** e **msb**, ma in locazioni di memoria diverse.
Poiché il linker non ha segnalato nessuna anomalia, siamo un po' curiosi di vedere cosa succede nella simulazione.
Carichiamo perciò il software simulatore, il cui uso è stato spiegato nelle riviste **N.184** e **N.185**, ed eseguiamo la simulazione istruzione per istruzione fino ad arrivare al punto visibile in fig.17, dove in **giallo** sono evidenziate le istruzioni che nel programma **PCONT** riguardavano le variabili **lsb** e **msb**, cioè:

ldi lsb,00h
ldi msb,00h

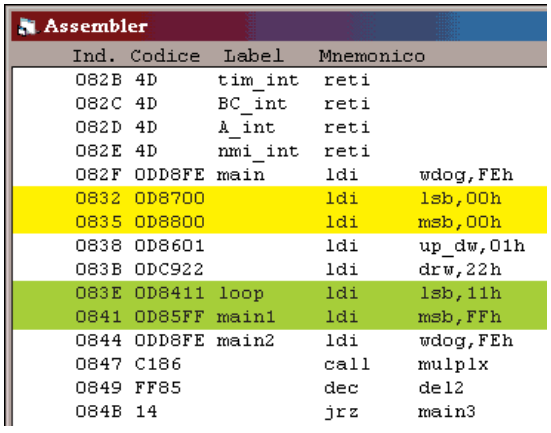


Fig.17 In giallo sono evidenziate le istruzioni ldi delle variabile lsb e msb del programma PCONT; in verde altre istruzioni che non rispettano il listato di PCONT. Vi facciamo notare (vedi colonna opcode) che le locazioni di memoria sono differenti.

Confrontando il loro **opcode** (vedi colonna **codice** in fig.17) con il listato di fig.2, si può notare che sono corrette. L'operazione **ldi** infatti, avviene esattamente nelle due locazioni di memoria definite in **PCONT**, cioè **087h** e **088h**.

Sempre in fig.17 abbiamo evidenziato in **verde** altre due istruzioni, cioè:

loop ldi lsb,11h
main1 ldi msb,FFh

che sono invece sbagliate. Infatti, verificando il listato di **PCONT** dovevano essere:

loop ldi del1,11h
main1 ldi del2,FFh

Verificando il loro **opcode**, possiamo vedere che l'operazione di **ldi** avviene nelle locazioni **084h** e **085h**, che corrispondono alle locazioni di **lsb** e **msb** definite nel programma **PLEXER**.

Il simulatore che, come sapete benissimo, utilizza il file con estensione **.dsd** per assegnare le etichette delle variabili e dei registri e rendere così leggibile il programma, quando ha decodificato le due ultime **opcode**, ha visualizzato le labels corrispondenti agli indirizzi **084h** e **085h**, che in questo file corrispondono alle etichette **del1** e **del2** del programma **PCONT**.

In fig.18 riportiamo il contenuto del file **XCONTA.DSD**, dove potete vedere che **lsb** e **msb** sono infatti definite 2 volte e in locazioni di memoria diverse.
Inoltre, **del1** e **del2** hanno la stessa locazione di memoria della seconda "serie" di **lsb - msb**.
Se però guardate più attentamente, vedrete che anche tutti i registri, l'accumulatore **a**, le porte logiche sono definite due volte, anche se in questo caso nella stessa locazione di memoria.
Questo sta a significare che nonostante il linker non abbia segnalato errore, c'è un problema.

Per poter assemblare in formato **.OBJ** sia **PCONT** che **PLEXER**, abbiamo dovuto inserire in entrambi i programmi le definizioni standard dei registri, dell'accumulatore, delle porte logiche e delle etichette utilizzate, perché altrimenti il compilatore avrebbe segnalato errore.
Quando però il linker ha unito i due **.OBJ**, ha come "sdoppiato" questi campi, generando una evidente confusione.

Per impedire che questo si verifichi ci vengono in aiuto due direttive: **.notransmit** e **.transmit**.

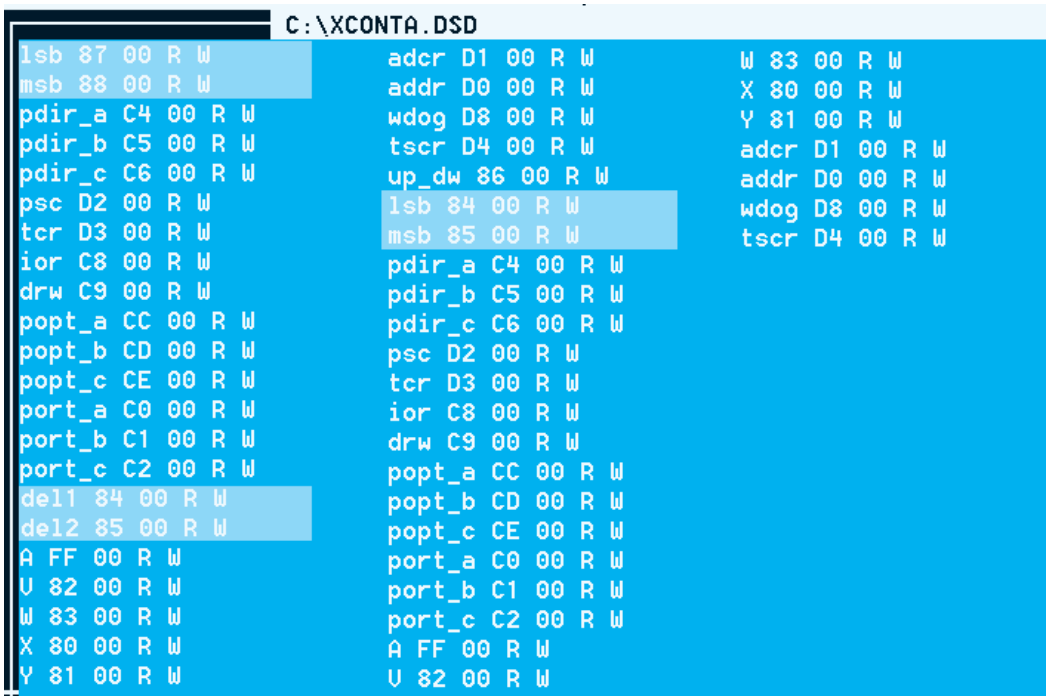


Fig.18 Il programma XCONTA.DSD riferito alle variabili lsb e msb definite due volte in due differenti locazioni di memoria. Come potete notare, le istruzioni del1 e del2 hanno le stesse locazioni di memoria della seconda serie di variabili lsb e msb. Anche le definizioni dei registri, dell'accumulatore, delle etichette ecc., sono state sdoppiate provocando confusione. Per ovviare a ciò si utilizzano le direttive .notransmit e .transmit.

```
        .title  "PLEXER"
        .vers   "ST62E25"
        .w_on
        .romsize 4
        .pp_on

        .notransmit
        .input  "ST62X.DEF"

;VARIABILI usate da questo PROGRAMMA

lsb     .def    084h
msb     .def    085h
        .transmit

        .global decrem,increm,mulplx
        .section 1
```

Fig.19 Parte del listato del programma PLEXER.ASM in cui abbiamo inserito le direttive .notransmit e .transmit.

Le direttive .notransmit e .transmit

Come abbiamo già visto per la direttiva **.global**, anche le direttive **.notransmit** e **.transmit** non sono strettamente necessarie nella fase di compilazione in formato **.OBJ**, ma vanno assolutamente inserite quando i programmi da linkare contengono le definizioni delle stesse variabili, degli stessi registri, delle stesse etichette ecc.

In questi casi è sufficiente che in uno dei programmi venga inserita **.notransmit** prima delle definizioni delle variabili comuni, e **.transmit** immediatamente dopo l'ultima variabile comune.

In questo modo il **linker** utilizza le variabili, i registri ecc. di un solo programma e collega tutte le istruzioni relative a queste locazioni.

Nel nostro caso, abbiamo inserito le direttive nel programma **PLEXER.ASM** come riportato in fig.19, e poi abbiamo ricompilato il programma per avere **PLEXER.OBJ** e abbiamo rilanciato il linker.
In fig.20 riportiamo il file **XCONTA.DSD** corretto.

```
lsb 87 00 R W
msb 88 00 R W
pdir_a C4 00 R W
pdir_b C5 00 R W
pdir_c C6 00 R W
psc D2 00 R W
tcr D3 00 R W
ior C8 00 R W
drw C9 00 R W
popt_a CC 00 R W
popt_b CD 00 R W
popt_c CE 00 R W
port_a C0 00 R W
port_b C1 00 R W
port_c C2 00 R W
del1 84 00 R W
del2 85 00 R W
A FF 00 R W
V 82 00 R W
W 83 00 R W
X 80 00 R W
Y 81 00 R W
adcr D1 00 R W
addr D0 00 R W
wdog D8 00 R W
tscr D4 00 R W
up_dw 86 00 R W
```

Fig.20 Il file XCONTA.DSD ottenuto dopo aver inserito correttamente le direttive .no-transmit e .transmit.

ULTIME CONSIDERAZIONI

Nell'esempio che vi abbiamo illustrato, abbiamo linkato due soli programmi, quindi è stato relativamente facile ricordare come scrivere le giuste istruzioni e la giusta sequenza per il linker.

Quando però i programmi diventano tanti e tante sono le routine da utilizzare, potrebbe risultare difficile gestire i programmi senza commettere nessun errore.
Per questo motivo vi suggeriamo un semplice metodo, utilizzato da molti programmatori, che vi consente di avere a disposizione anche il listato del programma eseguibile che si ottiene con il linker.
In questo modo potrete facilmente controllare l'opcode delle istruzioni e il loro indirizzamento nella memoria.

Prendiamo ancora una volta ad esempio i files **PCONT** e **PLEXER**.

Apriamo un editor qualsiasi e digitiamo:

ast6 –L –O PCONT
ast6 –L –O PLEXER

lst6 –S –I –T –V –M –O XCONTA PCONT PLEXER

quindi salviamo il file chiamandolo **XCONTA.BAT**.

A questo punto, ogni volta che dovremo compilare o linkare i due programmi, sarà sufficiente scrivere al prompt di DOS:

XCONTA

e automaticamente verranno lanciate in cascata prima le due compilazioni in formato **.OBJ** e poi il linker **lst6**.

Poiché nei comandi Assembler prima dell'opzione **–O** abbiamo inserito l'opzione **–L**, che genera anche il formato **.LIS** dei programmi, quando viene lanciato il linker, oltre a essere generato il programma eseguibile, nei files **.LIS** vengono sostituiti gli indirizzamenti **relativi** con gli indirizzamenti **assoluti** del programma finale.

Avremo così a disposizione anche il listato definitivo di **XCONTA**, che potremo leggere in **PCONT.LIS** e **PLEXER.LIS**.

Sebbene questa parte vi possa essere sembrata alquanto complicata, non dovete sottovalutare il fatto che ottenere dei programmi collegando tra loro programmi già esistenti è una **pratica comune** ad altri linguaggi di programmazione.

Pertanto coloro che intendessero approfondire anche lo studio di **altri linguaggi** software, non potranno che trarre **vantaggio** dalla lettura degli articoli dedicati al **linker** per i microprocessori **ST6**.

I PROGRAMMI LST6 E AST6

Informiamo tutti i nostri lettori che è possibile scaricare il programma **linker lst6** unitamente alla **versione 4.50** dell'**Assembler ast6** dal nostro sito:

WWW.NUOVAELETTRONICA.IT

Entrambi i programmi sono **gratuiti**.

QUALCOSA in più sul TIMER

La funzione **timer** dei microprocessori ST6 e il suo corretto utilizzo nella programmazione con linguaggio Assembler, è uno degli argomenti che abbiamo trattato fin dalle nostre prime lezioni.

In quegli esempi però, non si è tenuto volutamente conto del fatto che alcuni tipi di micro, oltre i tre normali registri del timer, hanno un **piedino**, grazie al quale è possibile attivare alcune modalità di funzionamento particolari e molto interessanti.

A seconda del modello di microprocessore, questo piedino può avere una differente numerazione e può essere indicato con la scritta **TIM1** (vedi fig.1) oppure con la scritta **TIMER** (vedi fig.2).

Abbiamo quindi ritenuto opportuno scrivere una piccola appendice per spiegarvi brevemente i casi in cui va utilizzato questo piedino, che per comodità chiameremo semplicemente **TIMX** (vedi fig.3).

Con le sigle **TSCR**, **TCR** e **PSC** abbiamo invece chiamato i tre registri utilizzati dal timer.

In fig.3 è riportato lo schema a blocchi interno che fa capo al timer del microprocessore **ST6**.

Passiamo ora a spiegare a grandi linee come funziona il **timer** dei **micro ST6**.

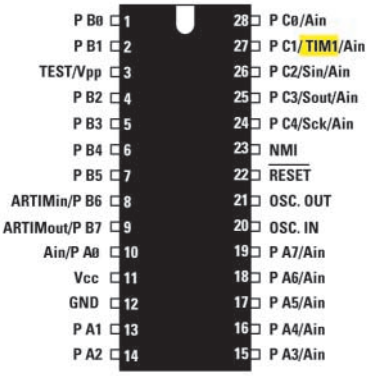
Il timer non è nient'altro che un **contatore** (**TCR**) che si **decrementa** di **1** ogni volta che un altro contatore, chiamato **prescaler** (**PSC**), decrementandosi a sua volta ad ogni impulso generato da un clock interno o esterno, arriva a **zero**.
Quando il **tcr** arriva a **zero**, setta alcuni valori all'interno del registro di controllo **TSCR**.

Ricaricando il contatore **TCR** e ripristinando il registro **TSCR**, è possibile far ripartire il timer tutte le volte necessarie al suo utilizzo.

E' possibile inoltre impostare un **divisore** sul **prescaler**, di modo che questo contatore si decrementi solamente ogni **1**, **2**, **4**, **8**, **16**, **32**, **64** o **128** impulsi di clock (interni o esterni).
Ma di questo abbiamo già parlato abbondantemente e spiegato con numerosi esempi negli articoli dedicati ai microprocessori ST6.

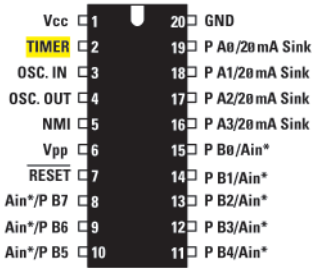
In questo articolo, parleremo solamente del **registro** di **controllo** (vedi fig.4) e cioè del:

TSCR Timer Status Control Register



ST 62T65C-ST 62E65C

Fig.1 I microprocessori serie ST62T65C e ST62E65C hanno un piedino dedicato all'attivazione del timer. Questo piedino è il 27 e viene solitamente chiamato TIM1.



ST 6210C-ST 6220C

Fig.2 I microprocessori serie ST6210C e ST6220C hanno un piedino dedicato all'attivazione del timer. Questo piedino è il 2 e viene solitamente chiamato TIMER.

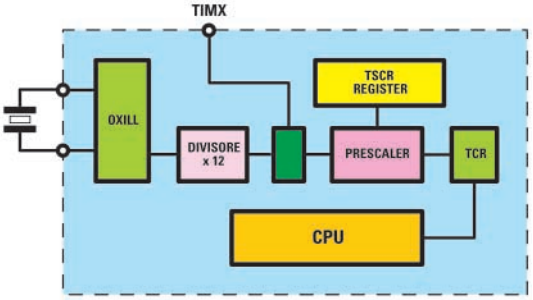


Fig.3 Al piedino del timer, che per comodità abbiamo chiamato nei nostri esempi TIMX, fanno capo tre registri chiamati TSCR - PRESCALER - TCR.

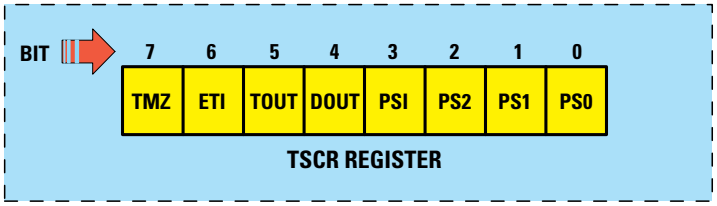


Fig.4 Il registro TSCR, abbreviazione di Timer Status Control Register, consente il controllo del Timer. In figura potete vedere il suo formato.

Con questo registro si effettua il controllo completo del timer. Il formato del registro è:

7	6	5	4	3	2	1	0
TMZ	ETI	TOUT	DOUT	PSI	PS2	PS1	PS0

Abbiamo contraddistinto i singoli bit con delle sigle e ora, bit per bit, ne diamo il significato, ma soprattutto vediamo come vanno gestiti questi bit nei nostri programmi per ottenere le diverse funzioni.

– bit 7 TMZ Timer Zero Bit

Questo bit viene gestito in maniera automatica dal timer, ma può essere anche settato o resettato da programma. Quando il registro contatore del timer, cioè il registro **tcr**, decrementandosi raggiunge lo **zero**, questo bit viene posto a **1**.
Per questo motivo, ogni volta che bisogna iniziare un nuovo conteggio del timer, questo bit va resettato a **0**, altrimenti il timer non riparte.

– bit 6 ETI Enable Timer Interrupt

Questo bit viene gestito attraverso il programma. Quando è posto a **0**, l'interrupt del timer **non** viene **abilitato**, quando invece è posto a **1**, la richiesta di interrupt del timer è **abilitata**. In questo caso, quando **TMZ**, cioè il bit **7**, diventa **1**, il programma salta al vettore relativo (definito nei nostri programmi-esempio con **tim_int** o **tad_int**).
Naturalmente dobbiamo avere correttamente abilitato il registro **ior** all'inizio del nostro programma.

I due bit che interessano direttamente il piedino **TIMX** del timer sono:

– bit 5 TOUT Timers Output Control
– bit 4 DOUT Data Output

Entrambi questi bit vengono gestiti dal programma e consentono di selezionare una delle **quattro** differenti **modalità** del timer.

Nota: in realtà le modalità sono **tre**, perché per le funzioni **Output Mode**, la modalità è la stessa e cambia solo il tipo di segnale in uscita, che può essere **0** o **1**.

Nella tabella successiva riportiamo le combinazioni possibili e le relative funzioni attivate.

TOUT	DOUT	TIMER PIN	MODALITÀ
0	0	Input	Event Counter
0	1	Input	Gated Mode
1	0	Output	Output Mode "0"
1	1	Output	Output Mode "1"

Vediamo ora insieme i singoli casi.

– Event Counter: TOUT = 0 DOUT = 0

In questa modalità il piedino **TIMX** del micro viene configurato in **input** e diventa l'input clock del registro prescaler **psc**.
Il **decremento** del registro prescaler **psc** avviene solo quando viene rilevato un **fronte** di **salita** (rising edge) su questo piedino.
Questa modalità viene ad esempio utilizzata per ottenere un **conta impulsi** o un **generico contatore** (vedi più avanti il primo programma-esempio).

– Gated Mode: TOUT = 0 DOUT = 1

In questa modalità il piedino **TIMX** viene configurato in **input** e il prescaler **psc** si decrementa sul clock del timer, cioè con la **frequenza interna divisa** per **12**.
Questo avviene solamente quando sul piedino **TIMX** viene rilevato uno stato logico "**high**", cioè **1**. Se lo stato è "**low**", cioè **0**, il registro prescaler **psc** non si decrementa mai ed il timer è bloccato fino al prossimo cambio di stato del piedino **TIMX**.
Questa modalità viene ad esempio utilizzata per ottenere un semplice **periodimetro** o per **misurare** la **durata** di un **impulso** (vedi più avanti il secondo peogramma-esempio).

– Output Mode "0": TOUT = 1 DOUT = 0

In questa modalità il piedino **TIMX** viene configurato in **output** e il prescaler **psc** si decrementa sul clock del timer, cioè con la **frequenza interna divisa** per **12**.
Quando il bit **7 TMZ** diventa **1**, sul piedino **TIMX** viene riportato il valore di **DOUT** e cioè **0**.

Questa modalità viene ad esempio utilizzata per ottenere un **generatore** di **frequenza**.

– Output Mode "1": TOUT = 1 DOUT = 1

Anche in questa modalità il piedino **TIMX** viene configurato in **output** e il prescaler **psc** si decrementa sul clock del timer, cioè con la **frequenza interna divisa** per **12**.
Quando il bit **7 TMZ** diventa **1**, sul piedino **TIMX** viene riportato il valore di **DOUT** e cioè **1**.
Come la precedente, anche questa modalità viene utilizzata per ottenere un **generatore** di **frequenza** (vedi più avanti il terzo programma-esempio).

– bit 3 PSI Prescaler Initialize Bit

Questo bit viene gestito attraverso il programma.
Quando è posto a **0**, il registro prescaler **psc** viene inizializzato al valore **7Fh** (in binario 01111111) e il suo conteggio viene bloccato.
In questa condizione quindi il timer non funziona.
Quando è posto a **1**, il conteggio (decremento) del prescaler è attivo ed il timer funziona.

– bit 2-1-0 PS2-1-0 Presc. Div. Factor

Questi bit vengono gestiti attraverso il programma.
La combinazione di questi tre bit permette di configurare il **fattore** di **divisione** del registro prescaler **psc** come si vede nella tabella proposta di seguito.

Grazie all'utilizzo corretto di questi tre bit, è possibile ottenere una vasta gamma di **temporizzazioni**.

PS2	PS1	PS0	DIVISORE
0	0	0	1
0	0	1	2
0	1	0	4
0	1	1	8
1	0	0	16
1	0	1	32
1	1	0	64
1	1	1	128

Alla luce di quanto fin qui detto, è opportuno fare alcune piccole e semplici constatazioni.
Quando si intende utilizzare il **timer**, è evidente che se il micro ST6 scelto dispone del piedino **TIM1** o **TIMER**, questo diventa praticamente riservato e non deve essere assolutamente utilizzato per scopi diversi da quelli elencati sopra, come, ad esempio, per gestire un pulsante o spedire dati ad un display, perché il programma ed il relativo circuito potrebbero non funzionare correttamente.

Inoltre, occorre fare attenzione all'**Option Byte** dei **micro** della versione **C**.
E' infatti possibile che, in alcuni modelli, sia inserito un **bit** che permette di scegliere tra la modalità **Pull-Up** o la modalità **No Pull-Up** del piedino **TIMER**. Questo fatto potrebbe influenzare la gestione del timer.

Infatti, se, ad esempio, in un programma che utilizza un micro modello **ST6220C** e che prevede l'utilizzo del timer in **Gated Mode**, **non** settiamo a **1** il bit **D2** dell'**Option Byte** (vedi fig.5), bisognerà ricordarsi di farlo elettricamente collegando tra il pie-

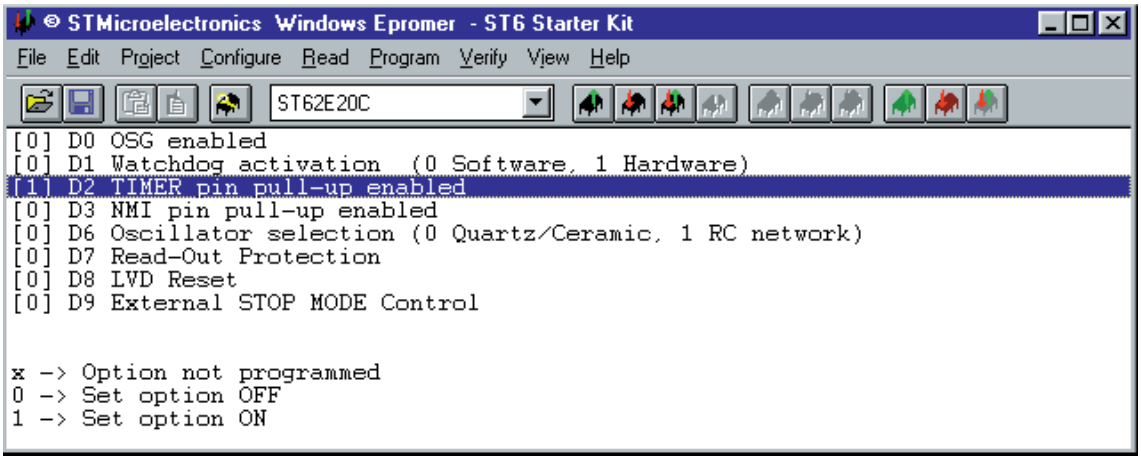


Fig.5 Quando utilizzate i micro della serie C che hanno il piedino del TIMER (vedi Tim1 e Timer nelle figg.1-2), ricordatevi di controllare se è possibile scegliere tra la modalità pull-up o no pull-up. Se così fosse, settate a 1 il piedino D2 dell'Option Byte oppure ricordatevi di collegare una resistenza esterna di pull-up da 10.000 ohm al piedino Timer.

dino **TIMX** e il positivo di alimentazione una resistenza esterna di pull-up da **10.000 ohm**, altrimenti il **timer** non **partirà mai**.
Non crediate di poterlo fare tramite il programma, perché, come abbiamo già detto, in questi tipi di micro il piedino **TIMER** è **riservato** e quindi non è configurabile tramite i tre normali registri delle porte (come, port_a, pdir_a e popt_a).

ESEMPI

Per spiegare quanto fin qui detto, abbiamo scritto tre semplici e completi programmi-esempio funzionanti:

contaimp.asm - durata.asm - ondaqua.asm

Di seguito spieghiamo con maggiori dettagli le sole istruzioni relative alla gestione del timer, ma voi trovate i sorgenti completi in questo cd-rom.

Per facilitare il riconoscimento delle istruzioni all'interno del programma, abbiamo inoltre provveduto a numerarle.

IL PROGRAMMA CONTAIMP.ASM

Per esemplificare la modalità "**Event Counter**" abbiamo scritto il programma **CONTAIMP.ASM**.

In questo programma viene utilizzato il piedino **TIMX** di un microcontrollore **ST6265** per "contare" gli impulsi. E' previsto il conteggio di un massimo di **9999 impulsi**, dopodiché si riparte.
E' inoltre prevista la visualizzazione in tempo reale di questo conteggio tramite la gestione di **4 display** numerici a **7 segmenti**.

Aprite con un normale editor, ad esempio Notepad, questo programma e posizionatevi alla riga da noi numerata **279**, dove trovate queste istruzioni:

```
ldi     tcr,1
ldi     tscr,01001000b
```

Con la prima istruzione abbiamo caricato il valore **1** nel contatore **tcr**.
Con la seconda istruzione carichiamo il valore binario **01001000** nel registro **tscr**.
Se adoperate la tabellina che abbiamo utilizzato ad inizio articolo, potete capire meglio perché abbiamo caricato questo numero nel registro **tscr**.

7	6	5	4	3	2	1	0
TMZ	ETI	TOUT	DOUT	PSI	PS2	PS1	PS0
0	1	0	0	1	0	0	0

I bit **5** e **4** denominati **TOUT** e **DOUT** sono a zero per attivare la modalità "**EVENT COUNTER**".

Il bit **6** denominato **ETI** è a **1** perché vogliamo gestire l'**interrupt** del **timer** quando il contatore **tcr** arriva a zero.

I bit **2-1-0** denominati **PS2-PS1-PS0** sono tutti a zero perché vogliamo che il divisore del **prescaler** sia uguale a **1**.
In questo modo, ogni fronte di salita rilevato sul piedino **TIMX**, va a decrementare subito il contatore del timer **tcr**.

Poiché noi abbiamo caricato il contatore **tcr** con il valore **1**, al primo impulso rilevato, il **tcr** va a zero e attiva l'interrupt (vedi **tad_int**).

Il bit **7** denominato **TMZ** viene posto a **0**, mentre il bit **3** denominato **PSI** viene posto a **1** e in questo modo il timer si attiva.

A questo punto posizionatevi alla riga da noi numerata **104**, dove trovate la routine:

tad_int

per la gestione dell'**interrupt** del timer.

Come abbiamo spiegato quando abbiamo parlato della modalità **Event Counter**, il programma attiva la routine **tad_int** quando il contatore **tcr** è a **zero**, cioè ad ogni impulso (fronte di salita) rilevato su **TIMX** e, tramite i due contatori che nel nostro programma abbiamo chiamato **unidec** e **cenmil** (vedi le istruzioni posizionate alle righe **48** e **49**), effettua un conteggio da **0** a **9999** impulsi visualizzandone il risultato su **4 display** numerici a **7 segmenti**.
Quando il conteggio degli impulsi arriva a **9999**, i contatori vengono azzerati e il conteggio riparte.

IL PROGRAMMA DURATA.ASM

Per esemplificare la modalità "**Gated Mode**" abbiamo scritto il programma **DURATA.ASM.**

In questo programma viene utilizzato il piedino **TIMX** di un microcontrollore **ST6265** per calcolare il tempo totale di accensione di una caldaia oppure di un innaffiatoio o anche di una pompa ecc., nell'arco delle **24 ore**. E' inoltre prevista la sua visualizzazione su **4 display** a **7 segmenti**.

Il tempo è calcolato in **secondi-minuti-ore** e sul display è possibile visualizzare a scelta i minuti ed i secondi oppure le ore ed i minuti, se si modificano alcune righe del programma sorgente.

Quando la caldaia (o l'innaffiatoio ecc.) si spegne, mette a **0** lo stato logico del piedino **TIMX** e il conteggio del tempo si ferma, quando si riaccende met-

te a **1** lo stato logico di **TIMX** e il conteggio riparte. Il tempo massimo del conteggio è di 24 ore.

Aprite ora con un editor, ad esempio Notepad, questo programma e posizionatevi alla riga da noi numerata **79**, dove trovate le istruzioni:

```
ldi     tcr,16
ldi     tscr,01011111b
```

Anche in questo caso adoperate la tabellina che abbiamo utilizzato ad inizio articolo, per capire meglio perché abbiamo caricato questo numero nel registro **tscr**.

7	6	5	4	3	2	1	0
TMZ	ETI	TOUT	DOUT	PSI	PS2	PS1	PS0
0	1	0	1	1	1	1	1

Notate anzitutto che il bit **5** denominato **TOUT** è a **0**, mentre il bit **4** denominato **DOUT** è a **1**.

Questa "combinazione" serve per attivare la modalità "**GATED MODE**".

I bit **2-1-0** denominati **PS2 PS1 PS0** sono tutti a **1**, quindi il divisore del prescaler è settato a **128**.
Avendo utilizzato un quarzo da **2,4576 MHz** e avendo caricato il contatore **tcr** con il valore **16** e settato il prescaler a **128**, abbiamo ottenuto una base tempi di **10 millisecondi**.

Il bit **6** denominato **ETI** è settato a **1** per attivare l'interrupt e quindi, ogni **10 millisecondi**, il programma attiva la routine **tad_int** (vedi istruzione alla riga **104**), dove gestisce sia il calcolo del tempo di utilizzo che la sua visualizzazione sui display.

Nota: nel nostro programma le istruzioni sono scritte in modo da visualizzare i **secondi** e i **minuti** di utilizzo. Infatti, alla riga **139** trovate l'istruzione:

```
ld      a,secondi
```

mentre alla riga **145** trovate l'istruzione:

```
ld      a,minuti
```

Se invece dei minuti e dei secondi, volete visualizzare i **minuti** e le **ore** dovete cambiare la riga **139** con l'istruzione:

```
ld      a,minuti
```

e la riga **145** con l'istruzione:

```
ld      a,ore
```

IL PROGRAMMA ONDAQUA.ASM

Per esemplificare la modalità "**Output Mode**" abbiamo scritto il programma **ONDAQUA.ASM**.

Con questo programma generiamo un'onda quadra della frequenza di **100 Hz** sul piedino **TIMX**.

Anche in questo caso abbiamo previsto di utilizzare un quarzo esterno che oscilli alla frequenza di **2,4576 MHz** e abbiamo opportunamente settato il contatore **tcr** e il registro di configurazione **tscr**.

Aprite dunque con un editor questo programma e posizionatevi alla riga da noi numerata **127** dove trovate le istruzioni:

```
ldi     tcr,16
ldi     tscr,01111110b
```

Adoperate anche in questo caso la tabellina che abbiamo utilizzato ad inizio articolo, per capire meglio perché abbiamo caricato questo numero nel registro **tscr**.

7	6	5	4	3	2	1	0
TMZ	ETI	TOUT	DOUT	PSI	PS2	PS1	PS0
0	1	1	1	1	1	1	0

Il bit **5** denominato **TOUT** è settato a **1** per attivare la modalità "**Output Mode**".

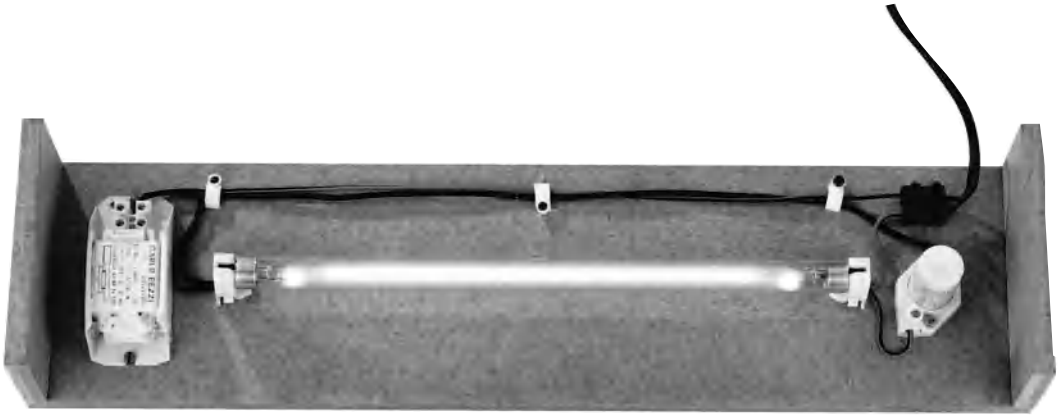
I bit **2-1-0** denominati **PS2 PS1 PS0** sono settati a **1-1-0** in modo da configurare a **64** il divisore del prescaler.

Con il quarzo a **2,4576 MHz**, il contatore **tcr** caricato con **16** e il divisore del prescaler a **64**, abbiamo ottenuto una base tempi di **5 millisecondi**.

Settando a **1** il bit **6** denominato **ETI** abbiamo attivato la richiesta di **interrupt** del **timer** e quindi ogni **5 millisecondi** il programma salterà alla routine **tad_int** visibile dalla riga **138** alla **148**.

Il bit **4** denominato **DOUT** è stato configurato a **1**. Questo significa che la prima volta che si attiva l'interrupt, nel piedino **TIMX**, che fino ad allora conteneva uno stato logico **0** (in virtù delle istruzioni che si trovano dalla riga **119** alla riga **121**), verrà "mosso" il valore del bit **DOUT** e cioè **1**.

Nelle attivazioni successive dell'interrupt del timer (vedi **tad_int** dalla riga **138** alla riga **148**), il bit **DOUT** verrà configurato alternativamente a **0** e poi a **1** e così via, in modo da generare una forma di onda quadra a **100 Hz**.



LAMPADA per CANCELLARE EPROM

Recatevi presso un qualsiasi negozio di materiale elettrico e provate a richiedere delle lampade ultraviolette che lavorino sui **2.300-2.700 Angstrom**, e come è accaduto a noi, vi verranno proposte delle comuni lampade Argon o delle lampade per abbronzarsi, che sono inadatte ai nostri scopi perché non emettono raggi ultravioletti sulla lunghezza d'onda richiesta.

Dopo aver spiegato che le lampade ci servivano per **cancellare** delle **EPROM**, ci è stato suggerito di rivolgerci presso i negozi di **computer**, dove abbiamo fatto un'amara scoperta.

La maggior parte di queste rivendite è **sprovvista** di queste lampade o, nel migliore dei casi, i pochi modelli di cui dispongono hanno prezzi esagerati.

Constatato che l'articolo sui **microprocessori** della serie **ST62** (vedi rivista **N.172/173**) ha incontrato un successo superiore alle nostre attese, tutti i lettori che hanno realizzato il **programmatore LX.1170** ci hanno tempestato di richieste chiedendoci una **lampada** per cancellare gli **ST62E**.

In passato avevamo disponibile una **lampada** per **cancellare** le memorie Eprom dei microprocessori, che si poteva direttamente collegare ai **220 volt** della rete, ma poiché è andata fuori produzione e quindi risulta anche per noi introvabile, ci siamo dati da fare per cercarne un'altra che la potesse sostituire.

La lampada che abbiamo trovato è di forma cilindrica, ha un diametro di **1,5 cm** e misura in lunghezza **30 cm**; per funzionare ha bisogno di due **zoccoli**, di un **reattore** e di uno **starter**, e poiché anche questi componenti non sono facilmente reperibili, abbiamo pensato di presentarla in Kit ai nostri lettori.

Inizialmente avevamo contattato qualche Industria per corredarla di un appropriato mobile, ma dopo aver conosciuto il prezzo per realizzarlo, abbiamo abbandonato questa idea, perché con solo quattro pezzi di legno ed un po' d'iniziativa, chiunque potrà realizzare un mobile adatto, risparmiando notevolmente.

Noi, per realizzare il contenitore che vedete nelle foto, abbiamo chiesto ad un falegname nostro amico due tavolette delle dimensioni di **47 x 12 cm**, che abbiamo adoperato come piano base e piano superiore, e due piccoli righelli alti **5,5 cm**, che abbiamo inchiodato ed incollato ai due lati delle tavolette.

A chi non piace tenere il contenitore grezzo, basterà che si procuri un pennello ed un poco di vernice **opaca** per legno, oppure una bomboletta di vernice spray, e con una spesa irrisoria avrà un elegante **cancella - Eprom**.

Con delle viti per legno abbiamo fissato sul piano superiore i due **zoccoli** per la lampada, lo **zoccolo** per lo **starter** ed il **reattore**.

Con del filo isolato di plastica abbiamo eseguito un semplice **impianto elettrico** (vedi fig.1) e per fissare le estremità del cordone di rete per i **220 volt** abbiamo usato due mammuth.

Sapendo che questa lampada può cancellare qualsiasi Eprom in circa **17-20 minuti**, le abbiamo collegato, dopo averlo programmato sui **20 minuti**, il **timer LX.1181**, che trovate pubblicato sulla rivista **N.174**.

Considerando la lunghezza della lampada, potrete cancellare contemporaneamente circa **10-12** memorie.

Collocate i microprocessori che volete cancellare sopra un cartoncino o sopra un sottile ritaglio di lamierino o compensato, quindi infilateli sotto la lampada in modo che la loro **finestra** risulti distante all'incirca **2 cm** dal bulbo in vetro.

Come abbiamo già avuto modo di accennare in precedenti articoli, prima di inserire il microprocessore sotto la lampada assicuratevi che la sua **finestra** sia pulita. Se così non fosse, sgrassatela con un batuffolo di cotone imbevuto di alcol o di acetone.

Non è conveniente fissare per molti minuti la luce **viola** emessa dalla lampada **accesa**, perché irritante per gli **occhi**.
Sul numero **172/173** l'articolista ha scritto che questa luce nuoce gravemente agli occhi, ma ha un poco esagerato, forse perché non avendola mai personalmente utilizzata, si è lasciato influenzare dalle affermazioni del tecnico.
In realtà l'occhio non viene **danneggiato**, come si potrebbe erroneamente dedurre da quanto scritto, ma si **irrita** provocando una condizione analoga (anche se molto inferiore) a quella prodotta quando si guarda l'**arco** di una saldatura elettrica senza la protezione degli occhiali.
Quindi se guardate per **5-6 minuti** di seguito i raggi ultravioletti emessi dalla lampada, vi sembrerà di avere negli occhi dei fastidiosi ed irritanti **granelli di sabbia**.
Per far scomparire il dolore, basterà fare qualche impacco con una soluzione di **acido borico**.

Per premunirvi contro questo fastidioso inconveniente, coprite, per il tempo che la lampada rimane accesa, la parte frontale del contenitore con un pezzo di cartoncino o con un panno, in modo da impedire alla luce di raggiungere gli occhi.

COSTO DI REALIZZAZIONE LX.1183

Il kit LX.1183 composto da una lampada lunga 30 cm della potenza di 8 Watt, completa di Reattore, Starter Mammuth, Zoccoli mignon e di un cordone di rete completo di spina ..... € 25,30

NOTA: Questa è la lampada che utilizziamo nel nostro laboratorio per cancellare le nostre Eprom e Microprocessori, perché con altre di potenza minore non sempre riuscivamo a cancellarle.

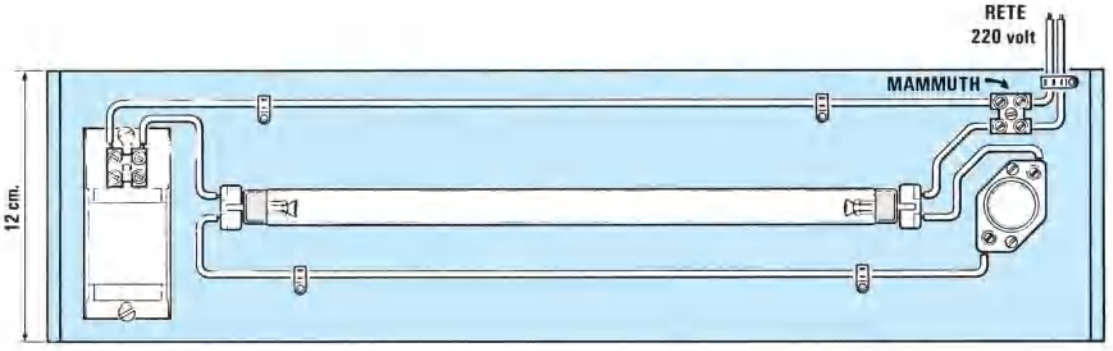


Fig.1 Su una piccola tavoletta in legno delle dimensioni di 12 x 47 cm abbiamo fissato a sinistra il reattore e a destra lo starter. Assieme alla lampada forniamo anche i due portalampada miniatura che fisserete con viti in legno sulla tavoletta alla distanza richiesta. Nel disegno si nota chiaramente come dovete effettuare il semplice schema elettrico.

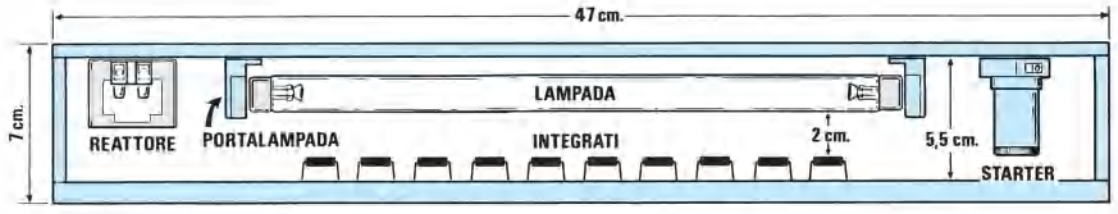


Fig.2 Sui due lati della tavoletta in legno abbiamo applicato due righelli alti circa 5,5 cm in modo da ottenere una distanza tra il bulbo della lampada ed il corpo superiore degli integrati (Eprom o Microprocessori cancellabili) che non risulti minore di 2 cm. Questa distanza non è critica, ma se superate i 4 cm dovrete aumentare i tempi d'esposizione.

INDICE ANALITICO TEORIA

A

A/D converter

C

Carry flag
Cicli macchina

D

Direttiva .ascii
Direttiva .asciz
Direttiva .block
Direttiva .byte
Direttiva .def
Direttiva .equ
Direttiva .extern
Direttiva .ifc
Direttiva .ifc - approfondimenti
Direttiva .macro
Direttiva .pp_on
Direttiva .set
Direttiva .w_on
Direttiva .window
Direttiva .windowend

E

Espressioni

F

Formato .hex
Formato .obj
Formato delle istruzioni
Funzione SPI

I

Interrupt
Istruzioni

L

Linker

M

Memoria EEprom
Memoria RAM e RAM aggiuntiva

O

Opcode delle istruzioni
Option Byte della serie C
Option Byte della serie C - funzioni attivabili
Opzioni del compilatore Assembler

P

Porte
Porte - gestione ottimale

R

Registri
Reset

T

Timer
Timer - registri
Tipi di registri

V

Variabili

W

Watchdog
Watchdog - approfondimenti

Z

Z flag

INDICE ANALITICO KIT

LX.1170 Programmatore per gli ST62/10-15-20-25

LX.1170 Consigli per migliorare il programmatore LX.1170

LX.1171 Scheda test per provare gli ST6

LX.1183 Lampada per cancellare le Eprom

LX.1202 Bus per testare i micro ST62/10-15-20-25

LX.1203 Stadio di alimentazione per BUS LX.1202

LX.1204 Scheda test con Display per provare gli ST6

LX.1205 Scheda test con Relè per provare gli ST6

LX.1206 Scheda per pilotare 4 diodi triac con un ST6

LX.1207 Scheda per pilotare un display numerico LCD con un ST6

LX.1208/N Scheda per pilotare un display alfanumerico LCD con un ST6

LX.1325 Programmatore per micro ST62/60-65

LX.1329 Bus per testare i micro ST62/60-65

LX.1380 Scheda test per la funzione SPI

LX.1381 Scheda test per la funzione SPI

LX.1382 Scheda test per la funzione SPI

LX.1430 Interfaccia per LX.1170 per gli ST6 serie C

INDICE ANALITICO SOFTWARE

Software simulatore DSE622 – sigla **DF.622**

Correzione degli errori con il simulatore DSE622

Se i programmi in DOS per ST6 non girano sotto Windows 95

Nuovo software simulatore per micro ST62/10-15-20-25 – sigla **ST622.1** e **ST622.2**

Nuovo software simulatore per micro ST62/60-65 – sigla **ST626.1** e **ST626.2**

Programma per LX.1170 – sigla **DF.1170.3**

Programma per LX.1325 e test per ST62/60 – sigla **DF.1325**

Programma per LX.1430 e per programmare gli ST6/C – sigla **DF.ST6-C**

Programmi per il TIMER

Programmi per la SPI – sigla **DF.1380**

Programmi test per LX.1202 – sigla **DF.1202.3**

Programmi test per LX.1207 – sigla **DF.1207.3**

Programmi test per LX.1208/N – sigla **DF.1208.3**