I Manuali di Giobe2000

MODULO LCD

Studio Tecnico ing. Giorgio OBER eurosito@giobe2000.it

Modulo Visualizzatore - Connettore

Controller/Driver per Visualizzatore a Matrice di Punti a cristalli liquidi

- Il **Visualizzatore a Matrice di Punti** a cristalli liquidi è, in realtà, solo la parte più appariscente di un **Modulo LCD** e, spesso, si tende a confondere il primo con il secondo...
- Di certo il **Display LCD** è assolutamente inutile senza un *circuito elettronico* in grado di gestirlo e, dato l'oneroso compito a cui è chiamato, sembra improbabile solo pensare a realizzarlo con *componenti discreti*.
- Per questo il mercato offre *strutture integrate* (**Moduli**) dotate di tutto ciò che serve, montato su un *supporto in vetroresina* (*circuito stampato*); da un lato è saldato il *display* e sul lato opposto sono in evidenza anche numerosi *integrati*.
- Certamente il più importante tra questi è il *controller*, un *componente programmabile* chiamato ad un duplice compito:
  - dispone della ***struttura*** necessaria per *gestire* fino a *50* (10*5) *pixel* per ciascuno dei possibili *caratteri alfanumerici* (*16*, *20*, *24*, *32*, *40*, ..) del *display*, organizzati su una o più *righe*
  - assicura l'***interfaccia*** intelligente tra il *visualizzatore* e il processore o il microcontrollore (single-chip) chiamato a gestirlo.
- Uno dei più diffusi *componenti* chiamati al **controllo** dei **Visualizzatori a Matrice di Punti** a cristalli liquidi (**LCD**) presenti sui **Moduli LCD** è l'**integrato HD44780**; il suo **pin-out**, di solito importante punto di partenza, in questo caso non sembra di particolare utilità (certamente a nessuno verrà la voglia di saldarne uno da qualche parte...).
- Lo stesso vale per lo **schema a blocchi** e per la **descrizione dei** suoi (numerosi) **piedini**; tutti i dettagli su questi argomenti sono disponibili al link dei Data Sheet, proposto al termine della recensione.
- Abbiamo già sottolineato che, all'atto dell'acquisto, non cercheremo un **Visualizzatore LCD** a (per esempio..) *2 righe* e *16 caratteri per riga*, ma un **Modulo LCD** che lo esibisca su uno dei suoi 2 lati.
- La scelta *semplifica* gran parte dei *problemi d'interfacciamento*, limitandoli alla conoscenza del *connettore* del **Modulo**; la figura riproduce un esemplare calzante con il mio esempio:

Display Alfanumerico da 16 caratteri per riga

n° pin	Nome	I/O	Descrizione
1	Vss	Power	Massa
2	Vcc	Power	+5V
3	Vo	Analog	Contrasto
4	RS	Input	Selezione
5	R/$\overline{W}$	Input	Read/$\overline{W}$rite
6	E	Input	Abilitazione
7	D0	I/O	Data LSB
8	D1	I/O	Data
9	D2	I/O	Data
10	D3	I/O	Data
11	D4	I/O	Data
12	D5	I/O	Data
13	D6	I/O	Data
14	D7	I/O	Data MSD
15	A	Retroilluminazione +	
16	K	Retroilluminazione -	

- Solitamente insieme all'oggetto viene fornito un *foglietto* che descrive il *nome* e la *funzione* dei ***segnali*** associati ai ciascun pin del connettore; la tabella riposta le assegnazioni più frequenti, da dare *quasi* per scontate anche in mancanza di documentazione.
- I *nomi* e la *funzione* sono gli stessi di quelli resi disponibili dal *controller* del **Modulo**, saldato sul circuito stampato in vetroresina, dal lato opposto a quello in cui è visibile il *display vero e proprio*.
- Ogni pagina di questa Sessione darà per scontata la presenza del *controller* **HD44780**, *quasi* un standard per questi oggetti, e descritto in seguito, nei minimi particolari.

- Descriviamo i 16 piedini del connettore del **Modulo**:
  - i pin **1** e **2** servono per fornire **alimentazione** (**5 volt**); di solito l'assorbimento è contenuto, dell'ordine di **pochi mA**.
  - il pin **3** è un *ingresso analogico* al quale si deve fornire una tensione compresa tra 0 e 5V al fine di **regolare il contrasto** del display, cioè la capacità di rendere più o meno scuri i pixel attivi, al fine di rendere migliore la visualizzazione, in funzione della luminosità dell'ambiente; di solito la regolazione si ottiene con un *potenziometro* da *10k*.
  - i pin **4** (**RS**), 5 (**R/$\overline{W}$**) e 6 (**E**) servono per il **controllo del dispositivo**; rispettivamente:
    - il primo, detto ***Register Selector***, indica la natura dell'informazione presente sul *bus dati*: se viene forzato a **0** il byte in ingresso sarà interpretato come un *comando* da eseguire (*istruzione*), mentre con un **1** sarà ritenuto *dato* da interpretare.
    - il livello logico sul secondo (***Read/Write***) specifica la direzione dei bytes sul *bus*: se vale **0** si sta *scrivendo* nella *memoria interna LCD*, mentre con un **1** questa memoria è sottoposta a *lettura*.
    - Il piedino di **E** (***Enable signal***) è il segnale che abilita il dispositivo: quando è a livello **alto** *sincronizza* la lettura del dato o del comando predisposto sul *bus dati*.
  - gli 8 pin (da **7** a **14**) del **Data Bus** sono il supporto per l'informazione bidirezionale tra il processore e il dispositivo, in base all'operazione che si sta svolgendo.
  - talvolta sono presenti 2 pin aggiuntivi (pin **15** e **16**), mediante i quali è possibile alimentare l'eventuale luce di **retro illuminazione**.

Controller/Driver HD44780 - Risorse Interne

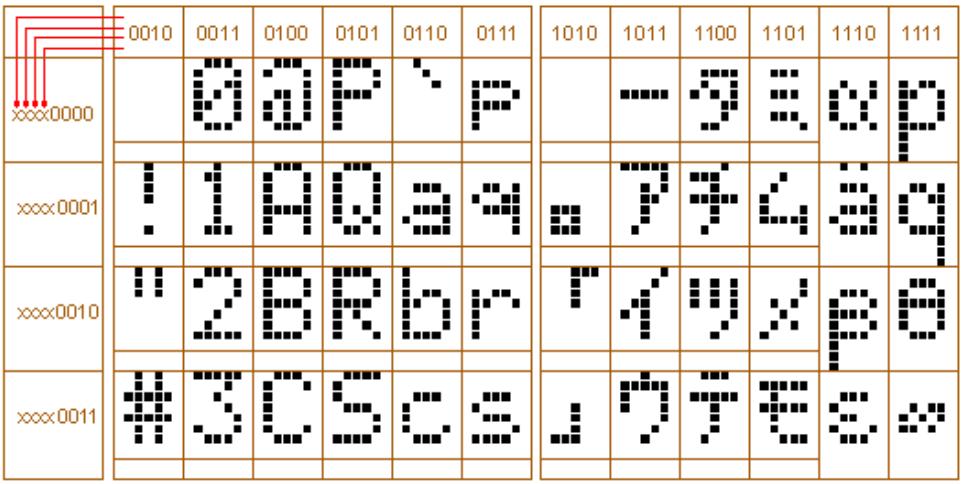
ROM e RAM del generatore di Caratteri - RAM per i dati del Display - Registri a 8 bit - Flag di Busy - Contatore di indirizzo (AC, Address Counter)

- Questa parte della *Sezione* si prefigge il compito di chiarire ogni ***aspetto hardware*** dei **Moduli LCD**, mostrando ogni dettaglio della loro struttura e dell'**integrato HD44780**, uno dei più diffusi *componenti* chiamati a governarla; lo scopo di questa recensione non è quello di descrivere in assoluto tutti i suoi dettagli, ma piuttosto quello di rendere il lettore autonomo nei confronti della sua programmazione.

- La sua presenza sui **moduli LCD** assicura da una parte (***driver***) le necessarie *caratteristiche elettriche* adatte all'accensione di ogni *carattere a matrice di punti* ospitabile dal visualizzatore, ma anche la struttura di ***segnali*** adatta a garantire l'interfaccia nei confronti del processore chiamato alla gestione software di tutto il processo di visualizzazione, in termini di *posizione e movimento* dei singoli caratteri.

- Da questo punto di vista è importante sapere quali sono le **risorse interne** disponibili.

- **ROM** del *generatore di Caratteri*: questa struttura si basa sulla presenza di una capace memoria **CGROM**, **C**haracter **G**enerator **R**ead **O**nly **M**emory, contenente i font dei **208** caratteri da 5*8 punti (8320 bit) e dei **32** caratteri da 5*10 punti (1600 bit); il *codice* del carattere che si desidera visualizzare, fornito come dato al controller, consente di localizzare la *matrice di punti* corrispondente all'interno di questa memoria a sola lettura, in accordo con la seguente tabella:



- Sono necessariamente definiti tutti i caratteri ASCII standard e numerosi altri caratteri (per lo più giapponesi...).

- **RAM** del *generatore di Caratteri*: mette a disposizione **64** bytes di **CGRAM, C**haracter **G**enerator **R**andom **A**ccess **M**emory; in queste locazioni è possibile prowedere alla *ridefinizione di caratteri* da programma (descritta con dovizia nella documentazione originale e qui non riportata); i codici di dato in grado di puntare queste nuove matrici di punti sono quelli da **00**H a **0F**H, evidenziati dalla prima colonna a sinistra, nella precedente tabella.

- **RAM per i dati del Display**: sono disponibili **80** bytes di memoria **DDRAM, D**isplay **D**ata **R**andom **A**ccess **M**emory, sufficienti per memorizzare altrettanti codici ad 8 bit, cioè caratteri, dato che il generatore interno è in grado di localizzarlo e visualizzarlo a partire da un semplice byte.
  Le locazioni di memoria non utilizzate per la visualizzazione dei caratteri possono essere usate in modo convenzionale, come normali locazioni RAM.
  Ciascun carattere occupa una posizione ben precisa dentro la memoria **DDRam**, indicata dal numero esadecimale del suo indirizzo; la loro sequenza varia in funzione della dimensione del visualizzatore presente sul modulo, o più precisamente in funzione del loro *numero di linee* e del *numero di caratteri per linea*:

Linee & Caratteri	Posizione carattere	Indirizzo nella DDRam
1 x 08	00 a 07	da 00H a 07H
1 x 16	00 a 15	da 00H a 0FH
1 x 20	00 a 19	da 00H a 13H
1 x 24	00 a 23	da 00H a 17H
1 x 32	00 a 31	da 00H a 1FH
1 x 40	00 a 39	da 00H a 27H
2 x 16	00 a 15	da 00H a 0FH *riga 0* da 40H a 4FH *riga 1*
2 x 20	00 a 19	da 00H a 13H *riga 0* da 40H a 53H *riga 1*
2 x 24	00 a 23	da 00H a 17H *riga 0* da 40H a 57H *riga 1*
2 x 32	00 a 31	da 00H a 1FH *riga 0* da 40H a 5FH *riga 1*
2 x 40	00 a 39	da 00H a 27H *riga 0* da 40H a 67H *riga 1*
4 x 16	00 a 15	da 00H a 0FH *riga 0* da 40H a 4FH *riga 1* da 14H a 23H *riga 2* da 54H a 63H *riga 3*
4 x 20	00 a 19	da 00H a 13H *riga 0* da 40H a 53H *riga 1* da 14H a 27H *riga 2* da 54H a 67H *riga 3*
4 x 24	00 a 23	da 00H a 17H *riga 0* da 40H a 57H *riga 1* da 14H a 2BH *riga 2* da 54H a 6BH *riga 3*
4 x 32	00 a 31	da 00H a 1FH *riga 0* da 40H a 5FH *riga 1* da 14H a 33H *riga 2* da 54H a 73H *riga 3*
4 x 40	00 a 39	da 00H a 27H *riga 0* da 40H a 67H *riga 1* da 14H a 59H *riga 2* da 54H a 7BH *riga 3*

- **Registri a 8 bit**: sono **2**, **IR**, **I**nstruction **R**egister, a sola scrittura, che ha il compito di memorizzare i codici delle istruzioni di controllo della visualizzazione, come quelle destinate alla gestione del cursore o all'indirizzamento delle 2 RAM; e **DR**, **D**ata **R**egister, il registro di transito delle informazioni.
  - nelle operazioni di **scrittura**: una precedente scrittura di un indirizzo in **IR** specifica in quale locazione di una delle 2 RAM verrà trasferito automaticamente il dato, non appena il processore ***scrive*** in **DR**.
  - nelle operazioni di **lettura**: una precedente scrittura di un indirizzo in **IR** prowede a trasferire automaticamente il dato in **DR** dalla specificata locazione di una delle 2 RAM; la ***lettura*** di **DR** da parte del processore attiva anche il trasferimento interno automatico di un nuovo dato dalla locazione di una delle 2 RAM puntata dall'indirizzo successivo, in attesa della prossima effettiva lettura da parte della cpu.
  - dei 2 **Registri** quello coinvolto in queste operazioni dipende dal valore del segnale **RS** (**R**egister **S**elector), che troveremo anche tra quelli presenti sul connettore dei **Moduli LCD**; con **RS** a **1** il registro coinvolto in lettura o scrittura è **DR** e l'informazione scambiata è un **dato** per o da una delle 2 RAM; con **RS** a **0** il registro coinvolto è **IR**, ma solo in scrittura (del codice di una **istruzione**); scambiata è un dato per o da una delle 2 RAM; l'eventuale lettura da parte del processore mette a sua disposizione il valore corrente del **Contatore d'indirizzo** (i 7 bit meno significativi, DB6-DB0) e quello della **flag di Busy**, cioè in sostanza dello stato del controller (entrambe queste risorse sono descritte qui di seguito...).
- **flag di Busy**: è trovata a **1** se il controller è occupato a gestire i trasferimenti interni e non è in grado di accettare l'istruzione successiva: la lettura *in polling* di questo bit consente dunque al processore di stabilire il momento giusto per fornire il comando successivo (quando il bit torna a **0**).
- **Contatore di indirizzo** (**AC**, Address Counter): consente di sapere in ogni momento la ***posizione corrente del cursore***, cioè l'indirizzo corrente dentro una delle sue memorie **DDRam** o **CGRam**:
  - nelle operazioni di **scrittura** il suo valore è copiato automaticamente da quello scritto dall'istruzione in **IR**, specificando in quale locazione di una delle 2 RAM verrà trasferito automaticamente il dato, cioè qual è il carattere da aggiornare o spostare.
  - nelle operazioni di **lettura** indica da quale locazione di una delle 2 RAM deve essere prelevato il dato.
  - dopo ogni operazione di lettura o scrittura delle 2 RAM il suo valore viene incrementato o decrementato automaticamente.

Controller/Driver HD44780 - I Diagrammi Temporali

Fase di scrittura - Fase di lettura

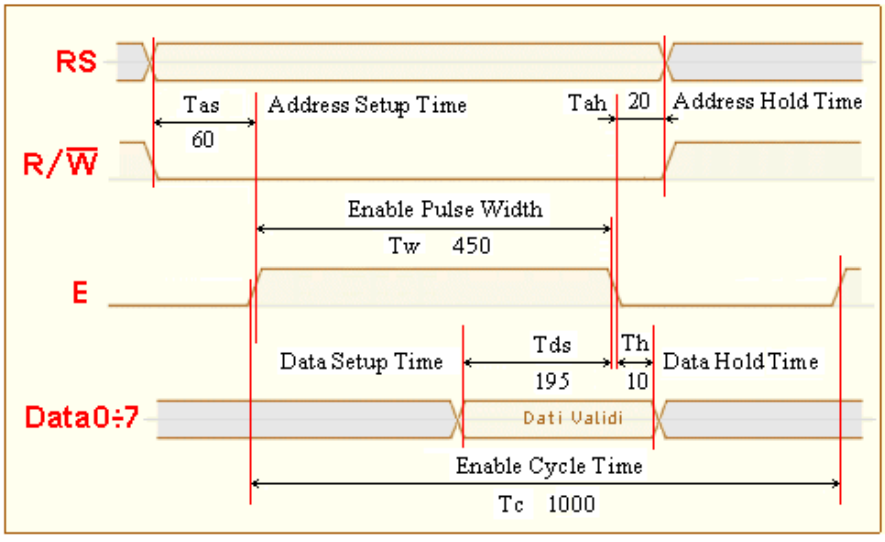
- Per interfacciare correttamente il controller **HD44780** con i processori chiamati a gestire il visualizzatore associato è necessario conoscere con grande dettaglio anche le **temporizzazioni** necessarie al componente per interpretare ed eseguire le sue istruzioni.
- Il processore controllore deve, in questo caso, prowedere all'attivazione di **3 segnali**, di solito riportati pari pari anche sul connettore del modulo visualizzatore che ospita l'**HD44780**; insieme alle 8 linee di **dato** vanno correttamente programmate anche le linee:
  - *Read/Write* (**R/W**, da porre a **0** se la CPU scrive il dato verso il controller, e a **1** se il dato è letto dal controller.

- *Register Select* (**RS**), da porre a **0** se l'informazione presente sul bus dati è il *codice operativo* di un comando da eseguire (istruzione), oppure a **1** se si tratta del codice di un carattere (dato effettivo).
- *Enable signal* (**E**), il vero e proprio segnale di sincronismo, da forzare a **1** quando il dispositivo è pronto a gestire un *dato* o un *comando* predisposto sul bus dati (in funzione del valore della ***flag di busy***, descritta in questa pagina: finchè il suo valore è **1** il controller è occupato a gestire i trasferimenti interni e non è in grado di accettare l'istruzione successiva.

Le 2 figure seguenti mostrano in dettaglio i ***Diagrammi Temporali*** relativi alle **fasi di scrittura** verso il controller e **di lettura** dal controller; il tempo minimo previsto per un intero ciclo di lettura o scrittura (**Tc**, Enable Cycle Time) di **1 ms**, tutti i tempi di transizione (fronti di salita e di discesa) sono da ritenere dell'ordine di **25 ns**, al massimo.

La fase di **scrittura** verso il controller **HD44780** è la più ricorrente:

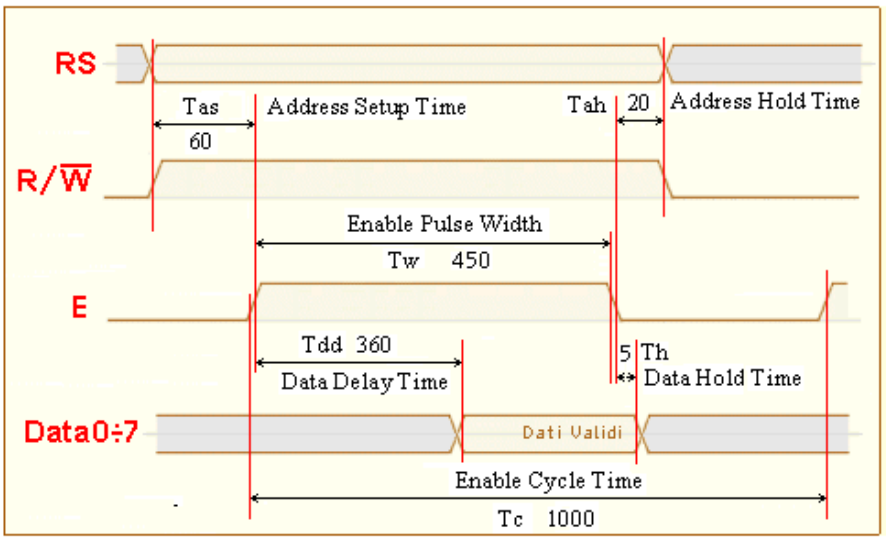
- il codice operativo di una istruzione per il controller o il codice del carattere da scrivere deve essere posto sul ***bus dati***.
- nel primo caso (istruzione) il segnale **RS** deve essere posto a **0**; nel secondo (dato) deve essere forzato a **1**.
- il segnale **R/$\overline{W}$** è posto a **0**.
- i segnali **RS** e **R/$\overline{W}$** devono essere posti (pressocchè simultaneamente) almeno **60 ns** prima di portare alto il segnale di abilitazione **E**; questo tempo minimo è definito **Tas**, Address Setup Time.
- Il segnale **E** è ancora a **0**; bisogna attendere almeno **60 ns** prima di portarlo a **1**.
- dopo questo tempo l'impulso di abilitazione **E** deve rimanere a **1** al minimo di **450 ns** (**Tw**, Enable Pulse Width), per lasciare al visualizzatore il tempo necessario per interpretare (o fornire) il dato; al termine è necessario riportare l'impulso a **0**: il valore presente sul *bus dati* viene **scritto** sul fronte di discesa di **E** [si da per scontato che esso sia presente sul bus almeno **195 ns** prima che **E** torni a **0**, tempo detto **Tds**, Dat Setup Time].
- il dato deve essere mantenerlo sul bus almeno altri **10 ns** (cioè per il **Th**, Data Hold Time) dopo che il segnale **E** è tornato basso.
- Non appena il segnale di abilitazione **E** viene riportato a **0** devono passare almeno **20 ns** prima di rilasciare i segnali **RS** e **R/$\overline{W}$**; questo tempo minimo è definito **Tah**, Address Hold Time.



La fase di **lettura** dal controller **HD44780**, intesa come assunzione dei *dati* contenuti in una delle 2 memorie **CGRam** e **DDRam**, è poco probabile; essenziale è, invece, la necessità di leggere lo *stato* del componente, cioè il bit7 del dato assunto con segnale **RS** a **0**: si tratta della **flag di Busy**, frequentemente citata in questa recensione, che deve essere letta *in polling* per stabilire il momento giusto per fornire il comando successivo (quando il bit vale **0**). In dettaglio:

- per leggere lo stato del controller (flag di busy e valore corrente del contatore d'indirizzo, **AC**) il segnale **RS** deve essere posto a **0**; per leggere i valori dalle 2 memorie (dati) deve essere forzato a **1**.
- il segnale **R/$\overline{W}$** è posto a **1**.
- i segnali **RS** e **R/$\overline{W}$** devono essere posti (pressocchè simultaneamente) almeno **60 ns** prima di portare alto il segnale di abilitazione **E**; questo tempo minimo è definito **Tas**, Address Setup Time.
- Il segnale **E** è ancora a **0**; bisogna attendere almeno **60 ns** prima di portarlo a **1**.
- non appena **E** è a **1 il controller metterà il dato sul bus** in un tempo (detto **Tdd**, Data Delay Time) al massimo di **360 ns**.
- dopo che l'impulso **E** dell'abilitazione è passato a **1** è necessario attendere almeno altri **450 ns** prima di riportarlo a **0**: il valore presente sul *bus dati* sarà **letto** sul fronte di discesa di **E**.

- Non appena il segnale di abilitazione **E** viene riportato a **0** devono passare almeno **20 ns** prima di rilasciare i segnali **RS** e **R/W**; questo tempo minimo è definito **Tah**, Address Hold Time.



- I diagrammi temporali si riferiscono ad un'**interfaccia a 8 bit**, ma la loro validità è mantenuta anche con **quella a 4 bit**: in questo caso l'informazione è gestita ponendo sui 4 bit più significativi del bus (DB7 - DB4) prima la **parte alta** del dato e poi **quella bassa**. Penserà un multiplexer dedicato, integrato nel chip, a ricostruire l'originario dato ad 8 bit.

Controller/Driver HD44780 - Conclusioni e Links

Fase di scrittura - Fase di lettura

- La tabella raccoglie le principali **caratteristiche elettriche** del componente (da notare le differenze tra l'impiego a bassa, **3V**, o a (relativamente) alta tensione, **5V**):

Caratteristiche Elettriche	Valori
potenza dissipata massima	3.0 mW (clock 270 Khz, assorbe 0,6 mA/Vcc=5V) 0,9 mW (clock 270 Khz, assorbe 0,3 mA/Vcc=3V)
corrente erogata tipica sul Bus Dati	1,2 mA (uscita a "0", massimo 0,4V con Vcc=5V) 0,1 mA (uscita a "0", massimo 0,2V con Vcc=3V)
corrente assorbita tipica dal Bus Dati	0,2 mA (uscita a "1", minimo 2,4V con Vcc=5V) 0,1 mA (uscita a "1", minimo 0,75V con Vcc=3V)
frequenza di clock tipica	270 kHz (Rt=91k/Vcc=5V e Rt=75k/Vcc=3V)
assestamento alimentaz. (*Powe Supply Rise Time*) massimo	15 ms (Vcc=5V) / 40 ms (Vcc=3V)
tempo di Enable totale (*Enable Cycle Time*) minimo	500 ns (Vcc=5V) /1000 ns (Vcc=3V) in R/$\overline{W}$
tempo di Enable alto (*Enable Pulse Width*) minimo	230 ns (Vcc=5V) / 450 ns (Vcc=3V) in R/$\overline{W}$
ritardo di Enable dopo RS, R/$\overline{W}$ (*Address SetUp Time*) minimo	40 ns (Vcc=5V) / 60 ns (Vcc=3V) in R/$\overline{W}$
ritardo di RS, R/$\overline{W}$ dopo Enable (*Address Hold Time*) minimo	10 ns (Vcc=5V) / 20 ns (Vcc=3V) in R/$\overline{W}$
anticipo dei Dati prima di Enable a 0 (*Data SetUp Time*) minimo	80 ns (Vcc=5V) / 195 ns (Vcc=3V) in R/$\overline{W}$
tenuta dei Dati dopo di Enable a 0 (*Data Hold Time*) minimo	10 ns (Vcc=5V) / 10 ns (Vcc=3V) in $\overline{W}$
tenuta dei Dati dopo di Enable a 0 (*Data Hold Time*) minimo	5 ns (Vcc=5V) / 5 ns (Vcc=3V) in R
ritardo dei Dati dopo di Enable a 1 (*Data Delay Time* massimo	160 ns (Vcc=5V) / 360 ns (Vcc=3V) in R

- I seguenti riferimenti rendono disponibili i link ai siti di alcuni fornitori di **Display LCD**: i produttori offrono, oltre alle specifiche dei loro visualizzatori, interessanti *recensioni* e *application notes*:

HANTRONIX	http://www.hantronix.com/2_2.html
SHELLY	http://www.shellyinc.com
Scott Edwards Electronics, Inc.	http://www.seetron.com/slcds.htm

- I seguenti link rendono disponibili i **Data Sheet originali**, in **formato PDF**:

HITACHI Semiconductor	**http://semiconductor.hitachi.com/hd44780.pdf** **http://shellyinc.com/conchips/HD44780U.pdf**

Controller/Driver HD44780 - Software - Descrizione delle Istruzioni

Nop - Clear Display - Cursore a capo - Modo d'accesso dei caratteri

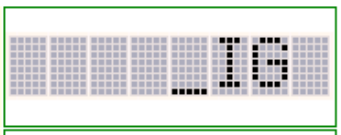
- Non è possibile controllare la scrittura su un **Visualizzatore a Matrice di Punti** a cristalli liquidi se non si conosce a fondo il *set delle istruzioni* previste dal suo controller e il *modo* di organizzarle.
- Questa parte della *Sezione* si prefigge il compito di chiarire ogni ***aspetto software*** dei **Moduli LCD**, mostrando ogni dettaglio sulla programmazione dell'**integrato HD44780**, uno dei più diffusi *componenti* chiamati a governarla.
- Nell'ambito di ciascuna delle applicazioni software, previste a supporto, verranno discusse e progettate le **Procedure** chiamate a organizzare ed eseguire ciascuna delle 3 fasi della ***Programmazione***, previste dall'elenco (*Fase iniziale*, *Scelte gestionali* ed *Erogazione caratteri*).
- Il componente **HD44780** è nato per costituire l'interfaccia con qualunque microprocessore o controllore di processo in grado di fornire i 2 segnali **R/$\overline{W}$** e **RS** che regolano il traffico d'informazioni del suo *bus dati* a 8bit.
- Anche una ***porta parallela*** può simulare con estrema facilità questa gestione, e sarà utilizzata per l'*interfacciamento* dei **Moduli LCD** in tutti i miei progetti.
- Naturalmente diventa importante imparare le ***istruzioni*** che consentono il controllo alla CPU esterna; alcune sono dedicate al controllo del visualizzatore, altre alla gestione dei dati sulle 2 RAM
- Riprendiamo un ***concetto importante***: se il controller è occupato a gestire i trasferimenti interni e non è in grado di accettare l'istruzione successiva, per cui è necessario leggere *in polling* la flag di **Busy** (bit7 del dato letto sul bus con **R/$\overline{W}$=1** e **RS=0**) prima di fornire il comando successivo; questo bit è forzato internamente a **1** se il controller è occupato ad eseguire una precedente istruzione, per cui è necessario aspettare che torni a **0**.
- Le **istruzioni** (*bytes* di comando) necessarie per controllare il **Modulo** controllato dal **HD44780** sono raccolte nella **tabella** seguente:

Istruzione	Input		Data Bus								Descrizione	Tempo max
	RS	R/W	D7	D6	D5	D4	D3	D2	D1	D0		
Nop	0	0	0	0	0	0	0	0	0	0	Nessuna operazione	0
Cancella Display			0	0	0	0	0	0	0	1	Cancella il visualizzatore e fissa Address DDRam Counter a 0	1,52 ms
Cursore a Capo			0	0	0	0	0	0	1	x	fissa Address DDRam Counter a 0 senza modificare la DDRAM display spostato alla posizione iniziale	1,52 ms
Modo d'accesso dei caratteri			0	0	0	0	0	1	I	S	Address >> I=0 Decremen I=1 Incremen display >> S=0 è bloccato S=1 scorre	37 us
Controllo Display			0	0	0	0	1	D	C	B	display >> D=0 spento D=1 acceso cursore >> C=0 invisibile C=1 visibile carattere>>B=0 fisso B=1 lampeggiante	37 us
Scorrimento cursore e display			0	0	0	1	S/C	R/L	x	x	S/C=0 il cursore si muove S/C=1 il display scorre R/L=0 verso sinistra R/L=1 verso destra (non modifica la DDRAM)	37 us
Impostazioni Parametri			0	0	1	DL	N	F	x	x	interfaccia >> DL=0 a 4 bit DL=1 a 8 bit Display >> N=0 a 1 linea N=1 a 2 linee Matrice carattere >> F=0 5x7 F=1 5x10	37 us
Indirizzo CGram			0	1	Character Generator RAM						Imposta indirizzo CGRAM per R/W dati	37 us
Indirizzo DDram			1	Display Data RAM Address							Imposta l'indirizzo DDRAM per R/W dati	37 us
Lettura Bit di Stato Lettura Indirizzo		1	BF	Address Counter							Legge Flag di Busy e'Address Counter BF=0 comando eseguito BF=1 comando in esecuzione	1 us
Scrittura Dato	1	0	Dato da Scrivere								Scrive dati dalla CGRAM o DDRAM	37 us
Lettura Dato		1	Dato da Leggere								Legge dati nella CGRAM o DDRAM	37 us

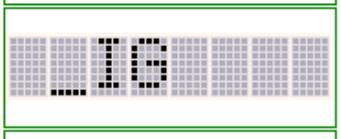
- I *tempi di esecuzione* (valore massimo) sono indicativi e si riferiscono ad una frequenza di oscillatore di **270 kHz**; se essa cambia nel valore $f_{\#}$ i nuovi valori si calcolano moltiplicando quelli della tabella per il fattore $270/f_{\#}$ (per esempio, con $f_{\#}$=**250kHz** >> ***37us*** diventa 37*270/250= ***40us***).
- Sebbene la tabella sia sinteticamente molto precisa, vediamo di proporre in dettaglio ***tutti i possibili codici operativi***:

- OpCode **00**H **Nop** : nessuna operazione, classica istruzione che no produce effetto
- OpCode **01**H **Clear Display** : scrive il codice 20H (cioè il carattere "spazio") in tutte le locazioni **DDRam** ed azzera il Contatore d'indirizzo **AC**, riportando il visualizzatore nelle condizioni originali; il cursore lampeggia nell'angolo in alto a sinistra (primo carattere della prima riga); in pratica esegue lo stesso servizio che la procedura ***ClearScreen*** (Pascal) impone al nostro monitor.
- OpCode **02**H o **03**H **Cursore a capo** : forza a 0 il contatore d'indirizzo **AC**, senza cambiare il contenuto della **DDRam**; il cursore lampeggia nell'angolo in alto a sinistra (primo carattere della prima riga); si comporta come il cursore del nostro monitor quando si preme il tasto ***Home***.
- **Modo d'accesso dei caratteri** : specifica la direzione verso la quale si sposterà il cursore (se i caratteri sono mantenuti al loro posto) o i caratteri visualizzati (se è il cursore che rimane fisso); in sostanza stabilisce se le 2 memorie **DDRam** o **CGRam** saranno lette o scritte in locazioni con indirizzo crescente o decrescente, (cioè incrementando o decrementando l'indirizzo contenuto in **AC**); naturalmente qualora l'istruzione sia riferita alla **CGRam** l'eventuale effetto di shift sul display non viene imposto.
  Le seguenti animazioni aiutano a capire l'effetto di queste 4 istruzioni: si suppone di scrivere la sequenza dei caratteri "GIOBE" nella **DDRam**:

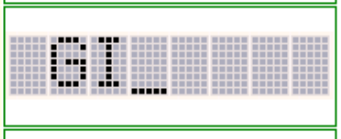
OpCode **04**H : si scrive in locazioni con indirizzo decrescente (cioè **AC** viene decrementato); il cursore si muove verso sinistra, cioè il testo creato rimane fisso: l'effetto è quello di una frase che si sviluppa *alla rovescia* all'indietro, *estratta* dal cursore, a partire dalla sua posizione iniziale.



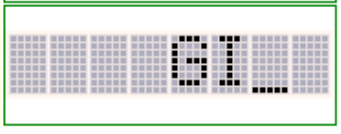
OpCode **05**H : si scrive in locazioni con indirizzo decrescente (cioè **AC** viene decrementato); il cursore è mantenuto fisso, cioè il testo creato si muove verso destra: l'effetto è quello di una frase che si sviluppa *alla rovescia* in avanti, *estratta* dalla posizione iniziale del cursore.



OpCode **06**H : si scrive in locazioni con indirizzo crescente (cioè **AC** viene incrementato); il cursore si muove verso destra (cioè il display è fisso): l'effetto è quello di una frase che si sviluppa *diritta* in avanti, *estratta* dal cursore, a partire dalla sua posizione iniziale.



OpCode **07**H: si scrive in locazioni con indirizzo crescente (cioè **AC** viene incrementato); il cursore è mantenuto fisso, cioè il testo creato si muove verso sinistra: l'effetto è quello di una frase che si sviluppa *diritta* indietro, *estratta* dalla posizione iniziale del cursore..



Controller/Driver HD44780 - Software - Descrizione delle Istruzioni

Controllo Display - Scorrimento Cursore/Testo - Predisposizione dell'indirizzamento della CGRam e della DDRam

- **Controllo Display** : esercita 3 tipi di controllo del visualizzatore:
  - decide se il cursore deve essere visibile o invisibile: quando è visibile il cursore ha la forma di un trattino (*underscore*) di 5 pixel sulla linea più bassa; quando non è visibile i caratteri sul display rimangono congelati, anche se nel frattempo sono state eseguite operazioni di scrittura o lettura.
  - attiva o meno l'effetto lampeggio (*blinking*) sul carattere puntato dal cursore, alternando alla sua consueta matrice di punti quella composta da pixel tutti spenti.
  - accende o spegne tutti i pixel del display: pur mantenendo i dati in memoria DDRam, i caratteri corrispondenti non vengono visualizzati; quando il display è spento ogni controllo sul cursore può ancora essere esercitato (cursore visibile/invisibile, con o senza effetto blink) ma risulta ovviamente inutile: per questo per spegnere il display può essere utilizzato indifferentemente uno dei 4 codici operativi corrispondenti.

In dettaglio:

OpCode **08**H ÷ **0B**H	***spegne*** il display
OpCode **0C**H OpCode **0D**H	***rende invisibile*** il cursore
OpCode **0E**H	***rende visibile*** il cursore *come* ***trattino***
OpCode **0F**H	***rende visibile*** il cursore *come* ***blocco lampeggiante***

- **Scorrimento Cursore/Testo** : impone lo spostamento di tutti i caratteri visualizzati o del solo cursore, verso destra o verso sinistra, **senza intervenire** sul contenuto delle locazioni DDRam (come invece imponeva il **Modo d'accesso dei caratteri** visto in precedenza, legato alla effettiva scrittura o lettura).
  Il servizio torna utile quando si deve intervenire sui caratteri già scritti, per effettuare una correzione, o semplicemente per cercare un determinato carattere; owiamente il cursore passa automaticamente da una linea all'altra e lo scorrimento è imposto simultaneamente su tutti i caratteri di entrambe le linee.
  Lo spostamento a sinistra (a destra) del cursore decrementa (incrementa) il **Registro** d'indirizzo **AC**.
  Lo spostamento a sinistra (a destra) del display da la sensazione che il cursore si muova dalla parte opposta, a destra (sinistra); lo spostamento del display non comporta modifiche di **AC**.
  Poiché i 2 bit meno significativi non sono coinvolti nella definizione di questi comandi, ci sono ben 4 codici operativi per ogni funzione; in dettaglio:

OpCode **10**H÷**13**H	sposta il ***cursore*** di un carattere a sinistra
OpCode **14**H÷**17**H	sposta il ***cursore*** di un carattere a destra
OpCode **18**H÷**1B**H	sposta il contenuto dell'*intero* ***display*** di un carattere a sinistra
OpCode **1C**H÷**1F**H	sposta il contenuto dell'*intero* ***display*** di un carattere a destra

- **Predisposizione funzionale** : predispone il componente per garantire l'interfaccia desiderata dall'utente; in sostanza il funzionamento viene programmato:
  - il numero di bit (*Data Lenght*) da trattare in scrittura o lettura (in pratica il numero di linee coinvolte sul bus dati, di solito DB7÷DB0 con 8 bit e DB7÷DB4 con 4 bit); con rapporti a 4 bit sono necessarie 2 letture o 2 scritture.
  - il numero di linee coinvolte nella visualizzazione (se il display ne prevede più di una).
  - la quantità di pixel utilizzata per le matrici del carattere, 5x7 o 5x10.

  Anche in questo caso i 2 bit meno significativi non sono coinvolti nella definizione di questi comandi, per cui ci sono ben 4 codici operativi per ogni funzione.
  In dettaglio per ***interfaccia a 4 bit***:

OpCode **20**H÷**23**H	predispone display a **1** linea, con matrice di carattere **5x7**
OpCode **24**H÷**27**H	predispone display a **1** linea, con matrice di carattere **5x10**
OpCode **28**H÷**2B**H	predispone display a **2** linee, con matrice di carattere **5x7**
OpCode **2C**H÷**2F**H	predispone display a **2** linee, con matrice di carattere **5x10**

  ..... mentre per ***interfaccia a 8 bit***:

OpCode **30**H÷**33**H	predispone display a **1** linea, con matrice di carattere **5x7**
OpCode **34**H÷**37**H	predispone display a **1** linea, con matrice di carattere **5x10**
OpCode **38**H÷**3B**H	predispone display a **2** linee, con matrice di carattere **5x7**
OpCode **3C**H÷**3F**H	predispone display a **2** linee, con matrice di carattere **5x10**

- **Predisposizione dell'indirizzamento della CGRam** : le locazioni della **CGRam** (memoria del ***Generatore di Caratteri***) sono **64**, per cui sono necessari **6** (2^6=**64**) codici operativi: poichè sono a 8 bit, come gli altri, si ottengono anteponendo il valore binario **01** (bit7, bit6, più significativi) a quello dell'indirizzo, da **00000** a **111111** (bit5-bit0, meno significativi). L'effettivo dato sarà scritto o letto solo dopo aver eseguito una di queste istruzioni.
  In conclusione:

OpCode **40**H÷**7F**H	puntano le singole locazioni **CDRam**, dalla prima (**01**000000 = **40**H) fino all'ultima (**01**111111 = **7F**H).

- **Predisposizione dell'indirizzamento della DDRam** : le locazioni della **DDRam** (memoria ***Dati del Display***) sono **80**, per cui i codici operativi necessari richiedono l'utilizzo di almeno **7** bit (2^7=**128**), visto che **6** (2^6=**64**) non sono sufficienti. Dunque non tutte le 128 combinazioni sono necessarie: i 7 bit d'indirizzo dovranno esprimere solo i numeri da 0 (**0000000**) a 79 (**1001111**; per formattare a 8 bit il corrispondente codice operativo, ai 7 bit (bit6-bit0, meno significativi) si antepone un bit a **1** (bit7, il più significativo).
  E' comunque chiaro che il controller **HD44780** è in grado di immagazzinare più caratteri di quanti il visualizzatore possa mostrare; per esempio, quelli ad una linea gestiscono fino a 40 caratteri, per cui solo una metà (da 0 a 39, da 00h a 27H) delle locazioni è copiata sul display; la rimanente metà **può** però **essere fatta scorrere** nelle medesime 40 posizioni del visualizzatore, con le istruzioni descritte in precedenza...
  In conclusione:

OpCode **80**H÷**CF**H	con visualizzatori ad **1** linea (N=0) gli indirizzi possibili vanno da 00H (=00H) a 4FH (=79); per cui i codici che puntano le locazioni **DDRam**, sono 10000000 = **80**H per la prima, fino a 11001111 = **CF**H, per l'ultima, l'ottantesima. con visualizzatori a **2** linee (N=1) gli indirizzi *per la prima* vanno da 00H (=0) a 27H (=39) e *per la seconda* vanno da 40H (=64) a 67H (=103) per cui i codici che puntano le locazioni **DDRam** sono:
OpCode **80**H **A7**H	*sulla prima* linea, a partire dalla prima (10000000 = **80**H) fino all'ultima, l'ottantesima (10100111 = **A7**H).
OpCode **C0**H **E7**H	*sulla seconda* linea, a partire dalla prima (11000000 = **C0**H) fino all'ultima, l'ottantesima (11100111 = **E7**H).

Controller/Driver HD44780 - Software - Programmazione: Fasi Iniziali

Inizializzazione del componente

- Le informazioni delle pagine precedenti sono sufficienti per la ***completa programmazione*** del **controller** (e del **modulo LCD** ad esso associato); la prima fase di ogni programma deve prowedere all'**inizializzazione del componente** che consiste nel realizzare questi obiettivi:
  - la ***flag di busy*** è mantenuta a **1** (controller occupato) fino alla fine della fase di inizializzazione.
  - la ***memoria DDRam*** viene riempita di spazi (come l'OpCode **01**H)
  - l'***interfaccia*** viene predisposta per 8 bit di dato (DL=1), per visualizzatore ad 1 linea (N=0) e per matrice del carattere con 5x7 pixel (F=0) (come l'OpCode **30**H)
  - il ***visualizzatore*** viene spento (D=0) e il cursore reso invisibile (C=0) e a forma di trattino (B=0) (come l'OpCode **08**H)
  - il ***modo d'accesso*** dei caratteri prevede l'autoincremento dell'indirizzo (I=0) con display bloccato (S=0) (come l'OpCode **04**H)

- Di solito queste condizioni iniziali sono imposte da hardware nel momento dell'accensione o in seguito all'attivazione del segnale ***reset***; va per altro sottolineato che il costruttore specifica chiaramente che ***per garantire l'inizializzazione da HW*** il tempo concesso all'alimentazione per raggiungere i 4,5V (power supply rise time) è al massimo di **10 ms**.

- Un buon programmatore non lascia mai niente al caso ed assicura **comunque** la sequenza delle istruzioni necessarie; l'esatta sequenza iniziale prevede (sempre con **R/$\overline{W}$=0** e **RS**=0):
  - un **ritardo iniziale** di almeno **15 ms** (con alimentazione a 5V) o di almeno **40ms** (con alimentazione a 2,7V).
  - la predisposizione di un'**interfaccia a 8 bit** senza curarsi, per ora, della forma del visualizzatore e della matrice del carattere; il codice operativo può essere dunque ambiguo su tutti i 4 bit meno significativi, cioè sarà del tipo 0011xxxx (va bene l'OpCode **30**H).
  - l'imposizione di un **ulteriore ritardo** di **4,1 ms**.
  - la ripetizione della predispone per **interfaccia a 8 bit**, ancora in modo ambiguo su tutti i 4 bit meno significativi (va bene ancora l'OpCode **30**H).
  - un **ulteriore ritardo** di **100 microsecondi**.
  - per la terza volta la medesima predisposizione ambigua per **interfaccia a 8 bit** (va bene ancora l'OpCode **30**H).

- Dopo queste 3 istruzioni di sincronizzazione (durante le quali **R/$\overline{W}$=0** e **RS**=0 e non è ammesso verificare la ***flag di busy***) sono fornite in sequenza quelle che impostano il controller per le nostre effettive esigenze; dopo ciascuna di esse è ammessa ora la verifica della flag, per altro non necessaria se il tempo di che trascorre tra una e la successiva (a livello microprocessore..) è sufficientemente elevato (almeno **1 ms**).

- Una prova pratica ha dimostrato, per altro, che le ***sequenze di ritardo*** consigliate dal costruttore e la necessità di ribadire ***3 volte*** la predisposizione ambigua di un'**interfaccia a 8 bit** con il codice operativo **30**H non è strettamente necessaria...
- Della *sequenza di inizializzazione* descritta poco fa, per sincronizzare il componente è stato sufficiente eseguire *un solo OUT* del valore **30**H .....
- Naturalmente, essendo operazioni *una-tantum* vale la pena comunque seguirle alla lettera... Ad esse faranno seguito i comandi per la gestione con interfaccia a **8 bit** o interfaccia a **4 bit**, che vedremo nelle pagine seguenti.

- **Riassumendo**:

Istruzione	Input		Codice Operativo & Data Bus								byte	Descrizione
	RS	R/$\overline{W}$	D7	D6	D5	D4	D3	D2	D1	D0		
												Ritardo di 15 ms
Function Set	0	0	0	0	1	1	x	x	x	x	30H	interfaccia a 8 bit
												Ritardo di 4,1 ms
Function Set	0	0	0	0	1	1	x	x	x	x	30H	interfaccia a 8 bit
												Ritardo di 0,1 ms
Function Set	0	0	0	0	1	1	x	x	x	x	30H	interfaccia a 8 bit

Controller/Driver HD44780 - Software - Programmazione: Scelte gestionali

Sequenza Finale per Interfaccia a 8 bit e per Interfaccia a 4 bit

- Nella fase iniziale si è comunque predisposta un'**interfaccia a 8 bit** senza curarsi della forma del visualizzatore e della matrice del carattere, erogando 3 codici operativi uguali a **30**H.
- Vediamo la sequenza finale per **interfaccia a 8 bit**; naturalmente in questo caso sono **coinvolti *tutti* e 8 i bit** del bus dati, da *DB7* a *DB0*, per cui *basta* scrivere **un solo byte** per ogni istruzione (sempre con R/$\overline{W}$=**0** e **RS**=**0**):
  - si specifica che l'effettiva **interfaccia** sarà a 8 bit: non è ancora necessario specificare la forma del visualizzatore e della matrice del carattere, per cui il codice operativo può essere ancora ambiguo su tutti i 4 bit meno significativi e sarà del tipo 0011 xxxx (va bene l'OpCode **30**H).
  - fissa i **dettagli** dell'interfaccia a **8 bit** (DL=1) (**Predisposizione funzionale**): per esempio se si desidera un visualizzatore a **1** linea (N=0) e una matrice del carattere con **5x7** pixel (F=0) il codice operativo necessario è 0011 00xx (=**30**H).
  - inizializza le caratteristiche del display (**Controllo Display**): per display acceso (D=1) cursore invisibile (C=0) e a forma di trattini (B=0) il codice operativo necessario è 0000 1100 (=**0C**H).
  - azzera la memoria DDRam (**Clear Display**): il codice operativo necessario è 0000 0001 (=**01**H).
  - fissa il modo di gestire il display (**Modo d'accesso dei caratteri**): con autoincremento dell'indirizzo (I=1) e display bloccato (S=0) il codice operativo necessario è 0000 0110 (=**06**H).

Riassumendo:

Istruzione	Input		Codice Operativo & Data Bus								byte	Descrizione
	RS	R/$\overline{W}$	D7	D6	D5	D4	D3	D2	D1	D0		
Function Set	0	0	0	0	1	1	×	×	×	×	30H	interfaccia a 8 bit
Effettivo Function Set	0	0	0	0	1	DL 1	N 0	F 0	×	×	30H	interfaccia a 8 bit una linea matrice 5x7
Display Control	0	0	0	0	0	0	1	D 1	C 0	B 0	0CH	display acceso cursore invisibile cursore a trattino
Display Clear	0	0	0	0	0	0	0	0	0	1	01H	azzera la DDRan
Enter Mode Set	0	0	0	0	0	0	0	1	I 1	S 0	06H	autoincremento cursore a destra display bloccato

- Vediamo la sequenza finale per **interfaccia a 4 bit**; in questo caso sono **disponibili solo i *4 bit* più significativi** del bus dati, *DB7*, *DB6*, *DB5* e *DB4*:
  - si specifica che l'effettiva **interfaccia** sarà a 4 bit: non è ancora necessario specificare la forma del visualizzatore e della matrice del carattere, per cui il codice operativo può essere ancora ambiguo su tutti i 4 bit meno significativi, cioè sarà del tipo 0010xxxx (va bene l'OpCode **20**H).

> ⚠ **Da notare** che l'istruzione precedente deve essere erogata con **una sola OUT**, come se fosse disponibile il bus al completo, a 8 bit: è proprio l'interpretazione di questa ulteriore *Function Set* che attiva il **multiplexer interno** per i prossimi comandi, così che ciascun codice (istruzione o dato a 8 bit) fornito da adesso in poi, sarà **scritto in 2 tempi**, prima il nibble alto e poi quello basso, sugli unici 4 bit disponibili.

  - fissa i **dettagli** dell'interfaccia a **4 bit** (DL=0) (**Predisposizione funzionale**), cominciando da ora ad usare la metà alta del bus dati (cioè dando per scontato che i bit da BD3 a DB0 non sono più disponibili); per esempio se si desidera un visualizzatore a **1** linea (N=0) e una matrice del carattere con **5x7** pixel (F=0) il codice operativo necessario è 0010 00xx (=**20**H), cioè bisogna scrivere in sequenza i 2 bytes **20**H (per 0010 xxxx) e **00**H (per 00xx xxxx).
  - inizializza le caratteristiche del display (**Controllo Display**): per display acceso (D=1) cursore invisibile (C=0) e a forma di trattini (B=0) il codice operativo necessario è 0000 1100 (=**0C**H), cioè bisogna scrivere in sequenza i 2 bytes **00**H (per 0000 xxxx) e **C0**H (per 1100 xxxx).
  - azzera la memoria DDRam (**Clear Display**): il codice operativo necessario è 0000 0001 (=**01**H), cioè bisogna scrivere in sequenza i 2 bytes **00**H (per 0000 xxxx) e **01**H (per 0001 xxxx).
  - fissa il modo di gestire il display (**Modo d'accesso dei caratteri**): con autoincremento dell'indirizzo (I=1) e display bloccato (S=0) il codice operativo necessario è 0000 0110 (=**06**H), cioè bisogna scrivere in sequenza i 2 bytes **00**H (per 0000 xxxx) e **60**H (per 0110 xxxx).

Riassumendo:

Istruzione	Input		Codice Operativo								Data Bus				byte	Descrizione
	RS	R/$\overline{W}$	7	6	5	4	3	2	1	0	D7	D6	D5	D4		
Function Set	0	0	0	0	1	0	×	×	×	×	0	0	1	0	20H	interfaccia a 4 bit
Effettivo Function Set	0	0	0	0	1	DL 0	N 0	F 0	×	×	0 0	0 0	1 ×	0 ×	20H 00H	interfaccia a 4 bit una linea matrice 5x7
Display Control	0	0	0	0	0	0	1	D 1	C 0	B 0	0 1	0 1	0 0	0 0	00H C0H	display acceso cursore invisibile cursore a trattino
Display Clear	0	0	0	0	0	0	0	0	0	1	0 0	0 0	0 0	0 1	00H 01H	azzera la DDRan
Enter Mode Set	0	0	0	0	0	0	0	1	I 1	S 0	0 0	0 0	1 ×	0 ×	00H 60H	autoincremento cursore a destra display bloccato

Controller/Driver HD44780 - Software - Programmazione: Erogazione dei caratteri

- Al termine delle 2 sequenze descritte nelle pagine precedenti (fase iniziale e scelta gestionale) il componente è pronto a ricevere le informazioni (**caratteri**) dal microprocessore e a **mostrarle sul display**; la sequenza è la seguente:
  - la posizione del carattere da visualizzare è comunque specificata dal valore corrente del contatore d'indirizzo **AC**, Address Counter.
  - se si danno per scontate le predisposizioni imposte dalla precedente inizializzazione **AC** vale **0**, per cui la scrittura comincerà a partire dal primo carattere in alto a sinistra.
  - se invece si desidera la scritta in una posizione diversa da quella iniziale è necessario ricorrere all'istruzione di **indirizzamento della DDRam** (**R/$\overline{W}$=0** e **RS=0**) associando il valore desiderato ai 7 bit meno significativi:
    # da **1**0000000 = **80**H a **1**1001111 = **CF**H, per display ad **1** linea.
    # da **1**0000000 = **80**H a **1**0100111 = **A7**H (*prima* linea) o da **1**1000000 = **C0**H a **1**1100111 = **E7**H (*seconda* linea) per display a **2** linee.
  - in questo secondo caso il controller memorizza il codice operativo in **IR**, **I**nstruction **R**egister, e da questo estrae l'indirizzo e lo copia automaticamente nel contatore d'indirizzo **AC**, Address Counter.
- Il processore prowede ora a **spedire il dato** (*codice del carattere*, **R/$\overline{W}$=0** e **RS=1**) e il controller lo copia nel registro di transito delle informazioni **DR**, **D**ata **R**egister; in questo istante il visualizzatore mostra la *matrice di punti corrispondente* nella posizione specificata:
  - la posizione di visualizzazione viene automaticamente incrementata (o decrementata) dopo ogni nuova entrata.
  - per questo ogni successiva **scrittura di dati** non ha bisogno di essere indirizzata.
  - prima della scrittura del dato è comunque necessario **leggere lo stato** del controller (**R/$\overline{W}$=1** e **RS=0**) verificando *in polling* il valore della ***flag di busy***, il bit7 del byte acquisito, aspettando che torni a **0**.

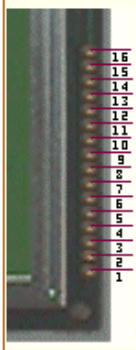
Progetti con Modulo Visualizzatore

Interfaccia con la porta parallela LPT1

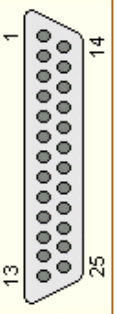
- La prima cosa da stabilire è il modo con cui controllare il **Modulo LCD**; naturalmente ogni tipo di computer o di microcomputer (*single-chip*) è in grado di assolvere egregiamente al compito.
- Questa Sezione offre alcune pregiate applicazioni software, tutte basate sul medesimo hardware, cioè sul collegamento alla *porta parallela* standard **SPP** (**S**tandard **P**arallel **P**ort) **LPT1**, programmata con l'Assembly del *processore 80x86*.
- Per la comunicazione con il **Modulo LCD** sono stati coinvolti solo i 2 **Registri** d'uscita della *porta parallela*:
  - 4 degli 8 bit del **Registro 0378H/0278H**, utilizzati come *bus dati*, disponibili sui pin da **6** (bit**4**) a **9** (bit7)
  - 3 dei 4 bit del **Registro 037AH/027AH**, utilizzato per i *segnali di controllo*, originalmente previsti dall'interfaccia *Centronics* per lo *strobe* (pin **1**), l'*init* (pin **16**) e il *select in* (pin **17**)
  - va sottolineato che il bit**0** ed il bit**3** sono sottoposti ad un'*inversione logica interna*, prima di essere disponibili sul connettore della porta; di questo dovremo quindi tener conto quando scriveremo il programma

⚠ Le *scelte circuitali* rispecchiano la filosofia del progetto e sono legate alla **conoscenza assoluta** del *controller* presente sul **Modulo LCD**; in particolare darò per scontata la presenza del controller **HD44780**, prodotto dalla Hitachi, descritto in dettaglio nelle pagine successive.

- Tutte le applicazioni proposte coinvolgono solo **4** degli **8** bit del *bus dati* del **Modulo**; sebbene sia possibile fare altrimenti la scelta di un'***interfaccia a 4 bit*** consente di ridurre di 4 unità i collegamenti con la parallela, a costo di un lieve appesantimento del software di gestione.
- Insieme al *bus dati*, sono coinvolte solo altre 3 linee del **Modulo**, già presentate nella pagina precedente: *Read/Write* (pin **5**, **R/$\overline{W}$**), *Register Select* (pin **4**, **RS**) e *Enable signal* (pin **6**, **E**).
- Lo schema finale è riproposto in ogni progetto (per esempio in questa pagina); la tabella mostra i ***collegamenti*** tra i *14* pin del **Modulo** (alimentato *esternamente* con un piccolo alimentatore da 5 volt stabilizzati) e *8* dei pin del connettore DB-25 della *porta parallela*:

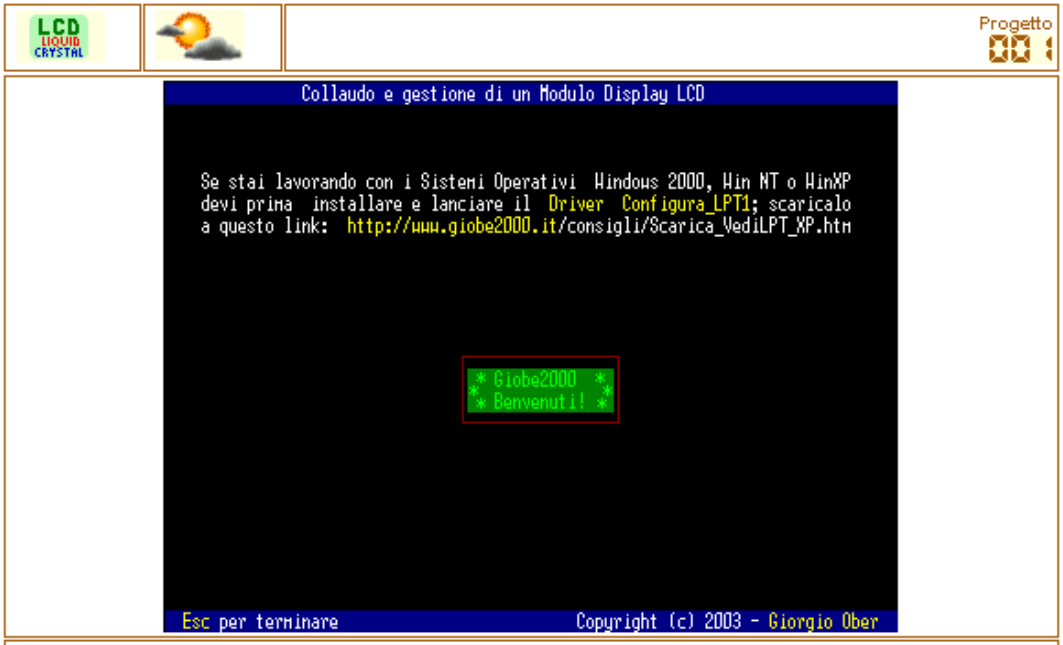


Modulo LCD			Porta Parallela LPT1		
pin	Funzione		pin	Funzione	Registro
1	Gnd	massa	18	Gnd	massa
2	Vcc	+5V	n.c.		
3	Gnd	massa	18	Gnd	massa
4	RS	Register Selector	16	bit2	037AHOut
5	R/$\overline{W}$	Read/Write	1	bit0 (invertito)	037AHOut
6	E	Enable signal	17	bit3 (invertito)	037AHOut
7	D0	Data Bus 0	n.c.		
8	D1	Data Bus 1	n.c.		
9	D2	Data Bus 2	n.c.		
10	D3	Data Bus 3	n.c.		
11	D4	Data Bus 4	6	bit4	0378HOut
12	D5	Data Bus 5	7	bit5	0378HOut
13	D6	Data Bus 6	8	bit6	0378HOut
14	D7	Data Bus 7	9	bit7	0378HOut



Progetto n°001 - Modulo Visualizzatore LCD su parallela LPT1

Gestione Modulo LCD: messaggio 2 x 16 (con controller HD44780)



Gestione di un modulo LCD a 16 caratteri: il progetto propone un *messaggio* ed un *gioco di caratteri* sul *display* (a *2 linee* di *16 caratteri* ciascuna) e sul video, in una particolare interfaccia grafica. La programmazione del *Controller HD44780* è ottenuta con l'aiuto dei 2 *registri d'uscita* (*0378H* e *037AH*) di una *porta parallela LPT1*.

Presentazione - Analisi del Problema

- Il progetto si occupa della gestione di un **Visualizzatore a Matrice di Punti** a cristalli liquidi, in grado di visualizzare *16 caratteri alfanumerici* su ciascuna delle sue *2 linee*; esso è generalmente integrato su un **Modulo LCD** e, spesso, si tende a confondere il primo con il secondo...
- In realtà il **display** (*visualizzatore*) **alfanumerico** è solo la parte più appariscente del **Modulo LCD**: il supporto in vetroresina (circuito stampato) su cui è saldato il *display* mette in evidenza anche numerosi integrati, saldati sul lato opposto.
- Uno di questo è il *controller* del **Modulo LCD**, l'interfaccia intelligente tra il visualizzatore e il processore o il microcontrollore (single-chip) chiamato a gestirlo
- Questo *chip* (nel nostro caso l'**integrato HD44780**) è un *componente programmabile* dotato dalla struttura necessaria per *gestire* fino a 50 (10*5) pixel per ciascuno dei possibili caratteri del *display*.
- La Sezione cui appartiene questo progetto è prodiga di informazioni sul suo funzionamento hardware e sulle sue necessità software...
- Per il controllo del **Modulo LCD** il progetto prevede la programmazione della *porta parallela* standard **SPP** (**S**tandard **P**arallel **P**ort) **LPT1**.

NB: Con i moderni Sistemi Operativi (**Windows NT**, **Windows 2000**, **Windows XP**) **non è più concesso l'*accesso diretto*** alle **porte di Input/Output** dall'ambiente ***Assembly*** o dai linguaggi di programmazione (***Pascal***, ***Delphi***, ***Visual Basic*** ...), come si poteva fare prima con **Windows 95/98/ME**.

- Quando si tenta, come fa il nostro progetto, un **Output** agli indirizzi Hardware viene generata una segnalazione d'errore di "**istruzione protetta**" o, semplicemente **non succede nulla...**
- Naturalmente un problema di questo tipo *non poteva rimanere irrisolto*: puoi accedere al **driver** che restituisce l'**accesso diretto** all'Hardware del computer in ambiente **Window2000/NT/XP** cliccando su questo link:
  - **Visibilità delle operazioni di I/O in ambiente Windows 2000/NT/XP**
- L'**obiettivo** del progetto è quello di programmare il *controller* del **Modulo LCD**, nel nostro caso l'**integrato HD44780** della Hitachi.
- Il risultato è facilmente ottenibile con l'aiuto delle linee d'uscita della *porta parallela*.
- Lo schema elettrico di tale interfaccia elimina ogni residuo dubbio ed è descritto al termine di questa trattazione.
- Il ***codice assembly*** del programma è visibile, **nella sua totalità**, scorrendo con la barra laterale il *testo* della seguente *casella*:

```
PAGE 66,132
TITLE** PROGRAMMA di GESTIONE di programmi ASSEMBLER  tipo COM    (marzo 2003)
SUBTTL ** TUTORIAL ASSEMBLY  --  www.giobe2000.it   --    by ing. Giorgio OBER
;┌──────────────────────────────────────────────────────────────────────────┐
;| NOME        : Gest_LCD                                                   |
;| AUTORE      : Giorgio OBER                                               |
;| VERSIONE    : marzo 2003                                                 |
;| DESCRIZIONE: Programma per il collaudo  dell'Interfaccia Software/Hardware|
;|               di un Visualizzatore LCD collegato  su porta parallela LPT1.|
;|               Il programma si limita ad erogare un semplice messaggio su un|
;|               Visualizzatore LCD a 16 caratteri per riga, con 2 righe.   |
;| ┌──────────┐  ------------------------------------------------------------|
;| |▒▒▒▒▒▒▒▒▒▒| Se si lavora con i Sistemi Operativi Win2000, Win NT o WinXP |
;| |▒▒ NB !! ▒| bisogna prima installare il Driver Configura_LPT1; scaricalo |
;| |▒▒▒▒▒▒▒▒▒▒| a:   http://www.giobe2000.it/consigli/Scarica_VediLPT_XP.htm |
;| ├──────────┤  ------------------------------------------------------------|
;| |▒▒▒▒▒▒▒▒▒▒| Se si lavora con i Sistemi Operativi Win2000, Win NT o WinXP |
;| |▒▒ NB !! ▒| la SHELL DOS deve essere a "Pieno Schermo" (l'esecuzione  in |
;| |▒▒▒▒▒▒▒▒▒▒| "finestra DOS" non produce effetto sul Display LCD)          |
;| ├──────────┤  ------------------------------------------------------------|
;| |▒▒▒▒▒▒▒▒▒▒| In ogni caso PRIMA DI OGNI SESSIONE DOS è necessario eseguire|
;| |▒▒ NB !! ▒| il programma Config_LPT1.EXE in ambiente Windows per attivare|
;| |▒▒▒▒▒▒▒▒▒▒| la visibilità dei driver installati con Configura_LPT1 !!!!! |
;| └──────────┘  ------------------------------------------------------------|
;└──────────────────────────────────────────────────────────────────────────┘
```

Analisi del Codice Sorgente - Le procedure importanti

- La **scrittura di caratteri** su un *visualizzatore LCD* non è operazione banale; comporta la conoscenza approfondita del controller che lo governa.
- E' necessario conoscere la sua *architettura*, i *codici operativi* (istruzioni) necessari per la sua programmazione, i *diagrammi temporali* che bisogna generare per simulare da SW i segnali HW necessari.

Per tutto questo è ***assolutamente necessario*** leggere con attenzione le *pagine precedenti*, dedicate all'integrato **HD44780**.

- Al solito mi occupo inizialmente della descrizione del **Main Loop**; il suo aspetto *innocuo* non deve far pensare ad una passeggiata in buona compagnia...
- Si tratta solo della dimostrazione, una volta di più, di come la parte principale di un programma si debba occupare **solo** di chiamare il lavoro delle sue procedure, per mantenere alta la leggibilità e non distogliere il pensiero dal vero obiettivo.

```
   MOV Byte Ptr CS:[Posiz],00H;Fissa l'indirizzo iniziale  in DDRam (il primo)
        CALL    DESKTOP       ;Inizializza  completamente il piano  di  lavoro
                              ;┐Inizializza il  controller HD 44780, simulando
                              ;│l'attivazione  della  linea di RESET;  vengono
        CALL    Pre_Ini       ;│erogati 3 bytes  a 30H separati  da  piccoli
                              ;│ritardi   "pesati";    per    default    viene
                              ;┘predisposta l'interfaccia a 8 bit
                              ;┐Predispone controller HD 44780 per  gestire il
                              ;│visualizzatore  nel modo operativo desiderato:
        CALL    Mia_Ini       ;│interfaccia a 4 bit, visualizzatore a 2 linee,
                              ;│matrice del carattere con  5x7 pixel,  display
                              ;│acceso,  cursore  invisibile  a  forma  fissa,
                              ;│azzeramento della memoria DDRam,autoincremento
                              ;┘dell'indirizzo, display bloccato
        MOV     DH,13         ;┐Fissa la Riga e la Colonna della posizione  di
        MOV     DL,33         ;┘stampa iniziale
        LEA     SI,TxtLCD     ;┐
XXX00:  MOV     AL,CS:[SI]    ;│Legge e mette a video la frase
        CMP     AL,00H        ;│    "      Giobe2000    "
        JE      XXX01         ;│    "     Benvenuti!    "
        CALL    Sta_Chr       ;│
        INC     SI            ;│
        JMP     SHORT XXX00   ;┘
XXX01:  CALL    Giochino
_OUT:   MOV     AH,4CH
        INT     21H
```

- La *casella di testo* mostra in dettaglio le ***fasi fondamentali*** del progetto:
  - come di consueto si predispone l'interfaccia grafica (*desktop*); la procedura locale (`CALL DESKTOP`) organizza la stampa dei messaggi di presentazione ed è assolutamente simile a quella coinvolta in quasi tutti i progetti
  - non è quindi necessario descriverla in dettaglio; ricordo che il servizio è ottenuto con l'intercessione delle **Procedure** e delle **Macro** appartenenti alle mie 2 librerie, **Giobe.MAC** e **Giobe.LIB**, disponibile in forma sorgente in **Giobe.ASM**.
  - inizializza (`CALL Pre_ini`) il *controller* **HD 44780**, simulando da SW l'attivazione della sua linea di ***reset***, vengono erogati 3 bytes a **30**H separati da piccoli ritardi "pesati", e viene predisposta (per default) l'***interfaccia a 8 bit***
  - predispone *controller* **HD 44780** per gestire il visualizzatore nel *modo operativo* desiderato: interfaccia a 4 bit, visualizzatore a 2 linee, matrice del carattere con 5x7 pixel, display acceso, cursore invisibile a forma fissa, azzeramento della *memoria DDRam*, autoincremento dell'indirizzo, display bloccato
  - legge e mette *a video* e sul *visualizzatore LCD* la frase "*Giobe2000 - Benvenuti!*":
    - associa il puntatore **SI** alla stringa da stampare (`LEA SI,TxtLCD`)
    - assume il carattere corrente (`MOV AL,CS:[SI]`) e controlla se si tratta del carattere terminatore (`CMP AL,00H`)
    - solo in questo caso pone termine alla visualizzazione (`JE XXX01`)
    - altrimenti prowede alla sua visualizzazione; di questo si occupa la **Procedura `Sta_chr`**, descritta più avanti
  - per mostrare l'efficienza del progetto offre un *messaggio animato* (`CALL Giochino`) che continua fino alla pressione di un tasto qualunque
  - le ultime istruzioni (`MOV AH,4CH,INT 21H`) *chiudono* il codice sorgente *restituendo il controllo al sistema operativo* che ci ospita.

- Un progetto come questo deve affrontare 2 tipi di problema:
  - la gestione hardware, cioè la creazione dei segnali richiesti
  - la gestione software, cioè la scelta dei codici operativi in grado di istruire il componente sul da farsi..
- La risposta al primo dei 2 quesiti è affrontata e risolta in questa pagina.

Ricostruzione temporale dei segnali con **Bus a 8 bit** [1 di 2]

- In una una pagina precedente abbiamo già descritto dettagliatamente il livello logico (*temporizzazioni*) che devono assumere i vari segnali da fornire al *controller* **HD44780** del **Modulo LCD** nella *fase di* ***scrittura*** (quelle *di* ***lettura*** non sono coinvolte dal progetto).
- Ti consiglio di aprire una seconda finestra, con lo *schema elettrico* dell'interfaccia, durante la lettura della descrizione.
- La **Procedura Out_By8** ha dunque il compito di ricostruire la *sequenza temporale* richiesta (nelle *varie fasi* della scrittura) sui segnali **E** (pin **6** del modulo), **RS** (pin **4**) e **R/$\overline{W}$** (pin **5**) del *controller* **HD44780**, assicurando le condizioni ideali per l'*output* verso il **Modulo LCD**, con l'aiuto del **Registro** d'uscita **037AH/027AH** della porta parallela.

⚠ Per poter sfruttare il visualizzatore in tutta la sua potenza è necessario anzitutto studiare con pazienza le ***istruzioni*** necessarie per programmarlo, descritte dettagliatamente nelle *pagine precedenti,* dedicate all'integrato **HD44780**.

- Entriamo nel merito, con riferimento al *testo* delle *caselle* seguenti:

```
;┌──────────────────────────────────────────────────────────────────────┐
;│PROCEDURA  :   Out_By8                                                │
;│DESCRIZIONE:   In ingresso AL deve essere predisposto con il  byte esadeci-│
;│               male da spedire al controller HD 44780,  attraverso il regi-│
;│               stro di dato (0378H) della porta parallela.            │
;│               Il valore presisposto in AH serve per predisporre il segnale│
;│               RS a 0 (AH=00H) se il byte da scrivere è un codice operativo│
;│               (istruzione) o RS a 1 (AH=01H) se invece deve essere scritto│
;│               un carattere (dato).                                   │
;│               La procedura ricostruisce la  sequenza  temporale  richiesta│
;│               sui segnali RS, R/W e ENABLE del  controller  HD  44780  del│
;│               modulo visualizzatore, assicurando le condizioni ideali  per│
;│               l'output verso il modulo LCD, con l'aiuto  del  registro  di│
;│               controllo (037AH) della porta parallela.               │
;│  ┌──────┐     I bit3 e bit0 della  porta  di  controllo  della  parallela │
;│  │ NB   │     (037AH) sono invertiti internamente e  vanno  riportati  al │
;│  └──────┘     loro valore naturale (di nuovo invertiti) da software !!  │
;│  ┌──────┐     Questa procedura organizza la gestione dei dati verso    il │
;│  │ NB   │     mudulo visualizzatore con INTERFACCIA A 8 BIT !!          │
;│  └──────┘                                                             │
;└──────────────────────────────────────────────────────────────────────┘
;                                        -\ /--------------------\ /----
;   SCRITTURA Carattere  (RS=1)    DATA    /   dato o istruzione /
;                  o                     -/ \--------------------/ \----
;   SCRITTURA Istruzione (RS=0)    ENABLE       /---\
;                                        ------/     \------------------
;                                  WRITE
;                                        ------------------------------
```

- Intanto ho pensato di passare alla procedura (in ingresso, in **AL**) il byte esadecimale da scrivere sul *controller* **HD44780** attraverso il **Registro** d'uscita **0378H/0278H** della *porta parallela*; esso è momentaneamente salvato in **BL** e sarà ripreso più avanti.

- Osservando lo schema sottolineo che, *per simulare i 3 segnali*, si usano rispettivamente i bit**3**, bit**2** e bit**0**, collegati rispettivamente ai pin **17** , **16** e **14** del del connettore **DB-25** della *porta parallela*.

Ricostruzione temporale dei segnali con **Bus a 8 bit** [segue, 2 di 2]

- Il primo compito della procedura è dunque quello di forzare **E** a **0**, **RS** a **0** o a **1** a seconda che si deva scrivere rispettivamente un *codice operativo* (istruzione) o un *carattere* (dato), e **R/$\overline{W}$** a **0**.

- Ciò significa forzare al medesimo valore logico i corrispondenti pin17, 16 e 1 del connettore della *porta parallela*.
- Per far questo basta scrivere sul registro la giusta combinazione, ma non bisogna dimenticare che le uscite del **Registro** d'uscita **037AH/027AH** della parallela, corrispondenti ai bit3 (associato ad **E**) e bit**0** (associato ad **R/$\overline{W}$**) sono *invertiti internamente*.

- In definitiva, per generale la **prima sequenza** di livelli sui segnali del modulo dovremo avere:
  - per scrivere un *codice operativo* (istruzione) la sequenza xxxx**00**x**0** sul connettore e la sequenza xxxx**10**x**1** sul **Registro** d'uscita **037AH/027AH**,
  - per scrivere un *carattere* (dato) la sequenza xxxx**01**x**0** sul connettore e la sequenza xxxx**11**x**1** sul **Registro** d'uscita **037AH/027AH**.

- Per rendere la procedura adatta sia alla scrittura di *dati* che a quella di *comandi* ho pensato di predisporre in **AH** un numero che la aiuti ad impostare automaticamente entrambe le situazioni.
  - se in ingresso **AH**=00H il byte da scrivere è un *codice operativo* (istruzione), il segnale **RS** deve essere posto a **0** e il byte da scrivere del **Registro** d'uscita **037AH/027AH** è xxxx**10**x**1**, per esempio **0B**H.
  - se in ingresso **AH**=01H il byte da scrivere è un *carattere* (dato), il segnale **RS** deve essere posto a **1** e il byte da scrivere del **Registro** d'uscita **037AH/027AH** è xxxx**11**x**1**, per esempio **0F**H.

```
Out_By8:PUSH    DX
        MOV     BL,AL           ; Salva il valore del byte  (dato o istruzione)
                                ;┌─[ segnali ]──┬─[bit del registro 037AH]─┬─byte┐
                                ;│              │b3/neg  b2/dir  bl  b0/neg│     │
        MOV     AL,00001011B    ;│ E RS  x R/W  │   EN_     RS   xx    R/W_│     │
                                ;│ 0  0  x  0   │   1       0    1       1  │ 0BH │
                                ;└─OUT ISTRUZ──┴──────────VALORE in AL──────┴─AL─┘
        CMP     AH,00H          ;┐AH=00H   RS 0 da scrivere codice operativo
        JE      Out_By0         ;┘AH=01H   RS 1 da scrivere carattere (dato)
                                ;┌─[ segnali ]──┬─[bit del registro 037AH]─┬─byte┐
                                ;│              │b3/neg  b2/dir  bl  b0/neg│     │
        MOV     AL,00001111B    ;│ E RS  x R/W  │   EN_     RS   xx    R/W_│     │
                                ;│ 0  1  x  0   │   1       1    1       1  │ 0FH │
                                ;└─OUT CARATT──┴──────────VALORE in AL──────┴─AL─┘
Out_By0:
```

- Finalmente tutto è pronto per iniziare la fase di *scrittura* sul modulo; la configurazione di bit che farà assumere ai segnali il giusto valore viene effettivamente attivata; da notare il *piccolo ritardo*, successivo all'attivazione, per consentire ai segnali **RS** e **R/W** di mantenere il loro valore logico per almeno **60 ns** (***Tas***, Address Setup Time) prima di portare alto il segnale di abilitazione **E**:

```
Out_By0:MOV     DX,037AH        ;┐Forza i segnali Enable, RS e R/W ai livelli
        OUT     DX,AL           ;│desiderati (0 , 1, 0) tramite il Registro di
                                ;┘Controllo della parallela, 037AH
        CALL    Delay           ;┐Ritardo Tas (Address Setup Time, almeno 60 ns
                                ;┘prima di riportare alto il segnale Enable)
```

- E' il momento di porre sul *bus dati* il byte da **scrivere**; per evitare di perderlo in precedenza (prima istruzione) era stato salvato in **BL** e da esso viene estratto per essere spedito al **Registro** d'uscita **0378H/0278H** della *porta parallela*.
- Da notare che questa operazione non altera (**PUSH**/**POP**) il valore corrente di **AL**, che contiene ancora in uscita il prezioso codice corrispondente allo stato attuale dei segnali (xxxx**10**x**1** per le istruzioni e xxxx**11**x**1** per i dati):

```
        PUSH    AX              ; Salva il valore di AH (dato o istruzione)
        MOV     AL,BL           ;┐Prepara per tempo il byte (dato o istruzione)
        MOV     DX,0378H        ;│sul bus dati del controller (via parallela)
        OUT     DX,AL           ;┘
        POP     AX              ; Recupera il valore di AH (dato o istruzione)
```

- Il segnale **E** è ancora a **0**; bisogna portarlo a **1** ed attendere almeno altri **450 ns** (***Tw***, Enable Pulse Width) e comunque almeno **195 ns** (***Tds***, Data Setup Time) prima di riportarlo di nuovo a **0**.
- Le istruzioni seguenti, con l'aiuto della **AND**, forzano a **0** solo il bit3 e quindi solo il segnale **E** a **1** (per via dell'*inversione interna*) lasciando inalterati **RS** e **R/$\overline{W}$**:

```
                                ;┌─[ segnali ]──┬─[registro 037AH]─┐Forza a "1"
                                ;│              │ b3  b2  bl  b0   │il segnale
        AND     AL,11110111B    ;│ E RS  x R/W  │ EN_ RS  xx  R/W_ │Enable (cioè
                                ;│ 1 inalterati │ 0    inalterati  │bit3 a "0")
                                ;└─OUT ISTRUZ──┴──VALORE in AL─────┘

        MOV     DX,037AH        ;┐Forza i segnali Enable, RS e R/W ai livelli
        OUT     DX,AL           ;│desiderati (1 , 1, 0) tramite il Registro di
                                ;┘Controllo della parallela, 037AH
        CALL    Delay           ;┐Ritardo Tw (Enable Pulse Width, al minimo 450
                                ;┘ns prima di riportare basso il segnale Enable)
```

- E' arrivato il momento di ***scrivere*** il byte presente sul *bus dati* nella *memoria del controller* del modulo: poichè l'evento si manifesta sul *fronte di discesa* di **E** è necessario riportarlo a **0**.
- Le istruzioni seguenti, con l'aiuto della **OR**, forzano a **1** solo il bit3 e quindi solo il segnale **E** a **0**, per via dell'*inversione interna*) lasciando inalterati **RS** e **R/$\overline{W}$**.
- La *procedura di ritardo* è chiamata perchè il dato deve essere mantenuto sul bus almeno altri **10 ns** (***Th***, Data Hold Time) dopo che il segnale **E** è tornato basso:

```
                                ;┌─[ segnali ]──┬─[registro 037AH]─┐Forza a "0"
                                ;|              | b3  b2  b1  b0   |il segnale
        OR      AL,00001000B    ;| E RS  x R/W  | EN_ RS  xx  R/W_ |Enable (cioè
                                ;| 0 inalterati| 1    inalterati  |bit3 a "1")
                                ;└─OUT ISTRUZ───┴──VALORE in AL────┘

                                ;┐Forza i segnali Enable, RS e R/W ai livelli
                                ;|desiderati (0 , 1, 0) tramite il Registro di
                                ;|Controllo della parallela, 037AH; il byte
        OUT     DX,AL           ;|presente sul bus dati viene scritto sul fronte
                                ;|di discesa di E [si da per scontato che esso
                                ;|sia presente sul bus almeno 195 ns (Tds, Data
                                ;┘Setup Time) prima che E torni a '0'
                                ;┐Ritardo Th/Tah: il dato deve essere mantenuto
                                ;|sul bus almeno altri 10 ns (Th, Data Hold Time)
                                ;|dopo che il segnale Enable è tornato basso.
        CALL    Delay           ;|Non appena il segnale di  abilitazione  Enable
                                ;|viene riportato a '0' devono passare almeno 20
                                ;|ns (Tah, Address Hold Time) prima di rilasciare
                                ;┘i segnali RS e R/W;
```

- La **Procedura Out_By8** è finalmente terminata; poichè nel caso di gestione con *bus a 4 bit* è necessario chiamarla 2 volte, il valore del byte da scrivere sul modulo è lasciato inalterato in uscita, prelevato da **BL**, in cui era stato salvato in precedenza (prima istruzione):

```
        MOV     AL,BL           ;┐Lascia in uscita (in AL) il valore del byte
                                ;┘(dato o istruzione) appena messo sull'LCD
        POP     DX
        RET
```

Ricostruzione temporale dei segnali con **Bus a 4 bit**

- La nostra applicazione **non** usa tutto il **bus dati** (interfaccia a 8 bit) bensì un **interfaccia a 4 bit**: la scelta consente di ridurre di 4 unità i collegamenti con la parallela, ma ci costringe a *multiplexare* la scrittura dei dati.
- Nella gestione del visualizzatore con *bus a 4 bit* l'informazione è gestita ponendo sui 4 bit più significativi del bus (DB7÷DB4) prima la **parte alta** del byte da scrivere e poi **quella bassa**; penserà un *multiplexer dedicato*, integrato nel chip, a ricostruire l'originario *dato ad 8 bit*.
- Dal punto di vista del programmatore ciò si traduce nella necessità di **eseguire 2 volte** la sequenza di *temporizzazioni*.
- La **Procedura Out_By4** provvedere alla scrittura di un *carattere* (dato, se in ingresso **AH**=01H, **RS** a **1**) o di un *codice operativo* (istruzione, se in ingresso **AH**=00H, **RS** a **0**):

```
;┌──────────────────────────────────────────────────────────────────────┐
;|PROCEDURA  :   Out_By4                                                |
;|DESCRIZIONE:   In ingresso AL deve essere predisposto con il  byte esadeci-|
;|               male da spedire al controller HD 44780,  attraverso il regi-|
;|               stro di dato (0378H) della porta parallela.           |
;|               RS a 0 (AH=00H) se il byte da scrivere è un codice operativo|
;|               (istruzione) o RS a 1 (AH=01H) se invece deve essere scritto|
;|               un carattere (dato).                                  |
;|               La procedura ricostruisce la  sequenza  temporale  richiesta|
;|               sui segnali RS, R/W e ENABLE del  controller  HD  44780  del|
;|               modulo visualizzatore, assicurando le condizioni ideali  per|
;|               l'output verso il modulo LCD, con l'aiuto  del  registro  di|
;|               controllo (037AH) della porta parallela.              |
;| ┌──────┐      I bit3 e bit0 della  porta  di  controllo  della  parallela |
;| | NB   |      (037AH) sono invertiti internamente e  vanno  riportati  al |
;| └──────┘      loro valore naturale (di nuovo invertiti) da software !!   |
;| ┌──────┐      Questa procedura organizza la gestione dei dati verso    il |
;| | NB   |      mudulo visualizzatore con INTERFACCIA A 4 BIT !! (di  fatto |
;| └──────┘      il byte da scrivere viene proposto sul bau in 2 tempi).   |
;└──────────────────────────────────────────────────────────────────────┘
```

- Vale solo la pena di ricordare che i 4 shift (**SHR**) tra una chiamata e l'altra servono proprio *portare verso l'alto* i 4 bit meno significativi del byte da scrivere, in modo da posizionarlo correttamente sulle uniche 4 linee di bus disponibili (quelle da DB7 a DB4).

```
Out_By4:CALL    Out_By8         ;Scrive il nibble alto (D7-D4) del dato o istruz.
        SHL     AL,1            ;┐Sposta i 4 bit meno significativi del
        SHL     AL,1            ;|byte da scrivere al posto di quelli più
        SHL     AL,1            ;|significativi di AL
        SHL     AL,1            ;┘
        CALL    Out_By8         ;Scrive il nibble basso (D3-D0) del dato o istruz
        RET
```

- Risolte le questioni Hardware rimangono quelle Software: l'interfaccia con la parallela è ora operativa, ma bisogna istruire il *controller* **HD44780** del **Modulo LCD** per assicurare il compito che ci siamo prefissi.
- Per il corretto funzionamento di questo *chip* è necessario sottoporlo *una-tantum* ad una ***programmazione preliminare***, all'accensione; per visualizzare qualsiasi frase sul display sarà poi sufficiente fornirgli solo ***semplici istruzioni*** contenenti i codici ASCII dei caratteri desiderati.

⚠ Insisto ancora sulla necessità di conoscere a fondo le ***istruzioni*** del componente, descritte nelle *pagine precedenti*, dedicate all'integrato **HD44780**.

- Le *parti preliminari* di codice sono 2:
  - una standard, per sincronizzare e inizializzare il *controller* **HD44780**, simulando l'attivazione della linea di ***Reset***
  - una personalizzata, per gestire il visualizzatore nel modo operativo desiderato

Inizializzazione del *controller*: simulazione dell'azione del segnale di RESET

- La fase di ***inizializzazione*** è sempre la prima da mettere in atto, quando si gestisce un *dispositivo*; di solito esiste un segnale fisico (***Reset***) che prowede alle stesse funzioni, ma un buon componente prevede sempre la possibilità di *re-inizializzarsi* anche da software.
- Le operazioni da compiere sono raccolte in forma di procedura, sebbene di norma essa sia chiamata una sola volta; questa scelta dà comunque dignità all'importante compito che svolge e si offre come eventuale risorsa, nel caso si dovesse prowedere a ri-sincronizzare il modulo...

```
;┌──────────────────────────────────────────────────────────────────────┐
;|PROCEDURA  :    Pre_Ini                                               |
;|DESCRIZIONE:    Inizializza il controller HD 44780, simulando l'attivazione |
;|                della linea di RESET;  vengono  erogati  3  bytes  a  30H |
;|                separati da  piccoli  ritardi  pesati;  per  default  viene |
;|                predisposta l'interfaccia a 8 bit, senza curarsi, per ora,  |
;|                della forma del visualizzatore e della matrice del carattere|
;└──────────────────────────────────────────────────────────────────────┘
```

- La **Procedura Pre_Ini** si occupa di fornire la sequenza di istruzioni standard, sempre necessaria all'inizio di ogni sessione.
- Si comincia con un *ritardo iniziale* di una **decina** di **ms** al massimo (power supply rise time), per assicurare l'*assestamento hardware* del *controller* dal momento dell'accensione, con alimentazione di 4,5V:

```
Pre_Ini:MOV     CX,15           ;┐
Pre_0:  CALL    Delay           ;|Ritardo iniziale di 15 ms (con Vcc=5V)
        LOOP    Pre_0           ;┘
```

- Inizialmente si predispone comunque l'**interfaccia** come se fosse **a 8 bit**, senza curarsi, per ora, della forma del visualizzatore e della matrice del carattere; dopo il comando si impone un ulteriore **ritardo** di **4,1 ms**:

```
                                ;┐Predispone le successive chiamate di Out_By8
        MOV     AH,00H          ;|per provvedere alla scrittura di un comando
                                ;┘cioè a portare il segnale RS=0, istruzione.
                                ;┐Predispone l'interfaccia a 8 bit senza curarsi
                                ;|della forma del visualizzatore e della matrice
        MOV     AL,30H          ;|del carattere; il codice operativo può essere
        CALL    Out_By8         ;|dunque ambiguo su tutti i 4 bit meno si-
                                ;|gnificativi, cioè sarà del tipo 0011 xxxx
                                ;┘30H=0011 00xxB  interfaccia a 8 bit
        MOV     CX,5            ;┐
Pre_1:  CALL    Delay           ;|Ulteriore ritardo di 4,1 ms
        LOOP    Pre_1           ;┘
```

- Si ripete altre 2 volte il comando che predispone per un'**interfaccia a 8 bit**:

```
                                ;┐Ripete la predisposizione dell'interfaccia a 8
        MOV     AL,30H          ;|bit, ancora in modo ambiguo su tutti i 4
        CALL    Out_By8         ;|bit meno  significativi:  va  bene  ancora  il
                                ;┘codice operativo 30H=0011 00xxB
                                ;┐Ritardo  finale di 0,1 ms, incluso nell'esecu-
                                ;┘zione della procedura appena eseguita
        MOV     AL,30H          ;┐Ripete  per  la terza volta la predisposizione
        CALL    Out_By8         ;|"ambigua"  dell'interfaccia a 8 bit; il codice
        RET                     ;┘operativo sarà ancora 30H=0011 00xxB
```

- Analizziamo ora la *parte preliminare personalizzata*, necessaria per inizializzare il visualizzatore nel modo operativo

Inizializzazione del *controller*: predisposizione delle modalità di visualizzione

- Non rimane che inizializzare il **Modulo LCD** (o meglio il suo *controller* **HD44780**) in funzione delle esigenze di visualizzazione desiderate; dopo la *fase standard* si deve prowedere alle scelte operative vere e proprie.
- Le possibilità di gestione sono 2: quella con interfaccia a 8 bit o quella con interfaccia a 4 bit.

- Il nostro progetto prevede l'*interfaccia a **4 bit***; in questo caso sono **disponibili solo i *4 bit* più significativi** del *bus dati*, *DB7*, *DB6*, *DB5* e *DB4*:
- Anche in questo caso le operazioni da compiere sono raccolte in forma di procedura, sebbene di norma siano eseguite una sola volta, per dare comunque dignità all'importante compito che svolge.
- La **Procedura `Mia_Ini`** si occupa di fornire la sequenza di istruzioni personalizzata, in accordo con i nostri specifici desideri.

```
;┌──────────────────────────────────────────────────────────────────────┐
;|PROCEDURA  :   Mia_Ini                                                |
;|DESCRIZIONE:   Predispone il controller HD 44780 per gestire il visualizza-|
;|               tore nel modo operativo desiderato, comunque con interfaccia|
;|               a 4 bit:                                               |
;| Cancella Display  00000001 01H  Azzera la memoria DDRam (Clear Display)   |
;| Accesso Caratteri 000001IS 06H  #  I=1 (bit1) > autoincrem.indirizzo DDRam|
;|                                 #  S=0 (bit0) > display bloccato        |
;| Modo  del Display 00001DCB 0CH  #  D=1 (bit2) > display acceso           |
;|                                 #  C=0 (bit1) > cursore invisibile      |
;|                                 #  B=0 (bit0) > cursore a forma fissa   |
;| Scorrimento Displ 0001SRxx 14H  # S/C=0(bit3) > si muove il cursore      |
;|                                 # R/L=1(bit2) > verso destra           |
;| Parametri         001DNFxx 28H  #  DL=0(bit4) > interfaccia a 4 bit      |
;|                                 #  N=1 (bit3) > visualizzatore a 2 linee |
;|                                 #  F=0 (bit2) > matrice carattere 5x7 pix|
;└──────────────────────────────────────────────────────────────────────┘
```

- Per prima cosa si specifica che l'effettiva **interfaccia** sarà **a 4 bit**, ancora senza entrare nel merito (forma del visualizzatore e della matrice del carattere):
  - in ogni caso il controller ritiene di essere ancora nella configurazione di default (come se potesse disporre del bus al completo a 8 bit), per cui il comando viene ancora erogato con una sola scrittura (OUT) sul bus, cioè con la **Procedura `Out_By8`**.
  - Il compito di questo comando è dunque quello di attivare il *multiplexer interno* per i prossimi comandi, così che ciascun codice (*istruzione* o *dato* a 8 bit) fornito da adesso in poi, sarà **scritto in 2 tempi**, prima il *nibble alto* e poi *quello basso*, sugli unici 4 bit disponibili.
  - La presenza dell'istruzione `MOV AH,00H` conferma alla **`Out_By8`** la necessità di trattare il byte da scrivere come comando (**RS=0**):

```
Mia_Ini:                        ;┐Predispone le successive chiamate di Out_By8
        MOV     AH,00H          ;|per provvedere alla scrittura di un comando
                                ;┘cioè a portare il segnale RS=0, istruzione.
                                ;┐Predispone l'interfaccia a 4 bit senza curarsi
                                ;|della forma del visualizzatore e della matrice
        MOV     AL,20H          ;|del carattere; il codice operativo può essere
        CALL    Out_By8         ;|dunque ambiguo su tutti i 4 bit meno si-
                                ;|gnificativi, cioè sarà del tipo 0010 xxxx
                                ;┘20H=0010 00xxB  interfaccia a 4 bit
```

- Conferma la scelta per l'interfaccia a **4 bit** cominciando da ora ad usare la ***metà alta*** del *bus dati* (cioè dando per scontato che i bit da BD3 a DB0 non sono più disponibili); il codice operativo scelto per il nostro progetto (28H) predispone la visualizzazione a **2** linee e una matrice del carattere con **5x7** pixel (se vuoi saperne di più: **Predisposizione funzionale**):

```
                                ;┐Fissa i dettagli dell'interfaccia a 4 bit (DL=
                                ;|bit4=0) cominciando da ora ad  usare  la  metà
        MOV     AL,28H          ;|alta del bus dati (cioè dando per scontato che
        CALL    Out_By4         ;|i bit da BD3 a DB0 non sono  più disponibili);
                                ;|si impone: visualizzatore a 2 linee (N=bit3=1)
                                ;|e matrice carattere con 5x7 pixel (F=bit2= 0);
                                ;┘il codice operativo è dunque 0010 10xx (=28H)
```

- Azzera la *memoria DDRam*; il codice operativo necessario è 01H (se vuoi saperne di più: **Clear Display**):

```
        MOV     AL,01H          ;┐Azzera la memoria DDRam (Clear Display ): il
        CALL    Out_By4         ;┘codice operativo necessario è 0000 0001 (=01H)
```

- Fissa il modo di gestire il display; il codice operativo scelto per il nostro progetto (06H) predispone con *autoincremento dell'indirizzo* e per *display bloccato* e per *movimento del cursore* (se vuoi saperne di più: **Modo d'accesso dei caratteri**):

```
                                ;┐Fissa il modo di gestire il display: con auto-
        MOV     AL,06H          ;|incremento dell'indirizzo DDRam e  del cursore
        CALL    Out_By4         ;|(I=bit1=1) e  con display bloccato (S=bit0=0);
                                ;┘il codice operativo è  dunque 0000 0110 (=06H)
```

- Inizializza le caratteristiche del display; il codice operativo scelto per il nostro progetto (0CH) predispone per *display acceso*, *cursore invisibile* e *a forma fissa* (se vuoi saperne di più: Controllo Display):

```
                        ;┐Inizializza le caratteristiche del display per
        MOV     AL,0CH  ;|display  acceso (D=bit2=1), cursore invisibile
        CALL    Out_By4 ;|(C=bit1=0) e a forma fissa (B=bit0=0);
                        ;┘il codice operativo è  dunque 0000 1100 (=0CH)
```

- Fissa il modo di gestire il movimento del cursore; il codice operativo scelto per il nostro progetto (14H) predispone *spostamento del solo cursore verso destra* (se vuoi saperne di più: Scorrimento Cursore/Testo):

```
                        ;┐Inizializza lo scorrimento sul display:  fissa
        MOV     AL,14H  ;|il movimento del cursore (S/C=bit3=0) e  verso
        CALL    Out_By4 ;|destra  (R/L=bit2=1);  il  codice  operativo è
                        ;┘dunque 0001 01xx (=14H)
```

- In uscita fissa il cursore in modo da puntare la prima locazione della *memoria DDRam*, tenendo memoria anche nella *variabile locale* [Posiz], a beneficio della **Procedura Addr_LCD** che si occupa della effettiva stampa del carattere sul visualizzatore:

```
        MOV     AL,00H          ;┐Fissa il  cursore  nella  prima  posizione, in
        MOV     CS:[Posiz],AL   ;|alto a sinistra, e istruisce il controller LCD
        OR      AL,80H          ;|puntando anche  la prima  locazione  della sua
        CALL    Out_By4         ;|memoria DDRam, di  indirizzo  xxxxxxx=0000000;
        RET                     ;┘l'istruzione è del tipo 1xxxxxxx=10000000=80H
```

Scrittura del Dato nella memoria del controller

- L'ultima procedura importante del progetto si occupa di ***scrivere il carattere*** sul *visualizzatore* del **Modulo LCD** e di ***farne eco*** sul *monitor*.
- Per questo si awale della **Procedura Sta_chr** che:
  - pone a video il *carattere* nella posizione di stampa corrente
  - aggiorna la posizione di stampa successiva, mantenendola nei limiti del riquadro previsto
  - Con l'istruzione `MOV AH,01H` conferma alla **Out_By4** la necessità di trattare il byte da scrivere come dato (**RS=1**)
  - trasmette il carattere (`CALL Out_By4`) anche al *controller* **HD44780** del **Modulo LCD**
  - aggiorna la posizione (*indirizzo*) di scrittura in *memoria DDram*, con l'aiuto della **SottoProcedura Addr_LCD** (che vediamo sotto):

```
;┌──────────────────────────────────────────────────────────────────────┐
;|SERVIZIO    : Sta_Chr                                                 |
;|DESCRIZIONE: Servizio per visualizzazione del carattere passato in ingresso|
;|             in AL; esso è posto a video nella posizione di stampa corrente|
;|             sul Visualizzatore e trasmesso al controller LCD; la posizione|
;|             di stampa viene aggiornata per puntare a quella del  carattere|
;|             successivo.                                              |
;└──────────────────────────────────────────────────────────────────────┘
Sta_Chr:PUSH    AX
        M_Chr   AL,DH,DL,2AH  ; Stampa il carattere sul Display a Video
        INC     DL            ; Incrementa la colonna
        CMP     DL,49         ;┐se NON si supera l'ultima  colonna della  riga
        JNE     Serv_C1       ;|video corrente  si  prosegue  stampando  sulla
                              ;┘stessa riga corrente
        MOV     DL,33         ;┐Altrimenti si forza il cursore  a  capo  della
        INC     DH            ;|linea opposta a quella che si sta lasciando..
        CMP     DH,15         ;|(rileva  automaticamente se  quella  superiore
        JNZ     Serv_C1       ;|o se quella inferiore)
        MOV     DH,13         ;┘

Serv_C1:POP     AX            ;┐Predispone la  successiva  chiamata di Out_By4
        MOV     AH,01H        ;|per provvedere alla scrittura  di un carattere
                              ;┘cioè a portare il segnale RS=1, carattere
        CALL    Out_By4       ;┐Trasmette al controller LCD il carattere in AL
        CALL    Addr_LCD      ;|(RS=1) e aggiorna la posizione del cursore per
        RET                   ;┘puntare il carattere successivo
```

- La **SottoProcedura Addr_LCD** ha un compito semplice ma fondamentale; sulla base del valore corrente della *variabile locale* [Posiz], calcola il nuovo indirizzo *DDram* (*memoria Dati del Display*) e ricostruisce da esso l'istruzione per forzare la scrittura proprio in quella locazione del controller.
- La logica che governa il calcolo deriva dal fatto che (come descritto in questa pagina) l'indirizzo di un carattere in *DDRam* varia in funzione della dimensione del visualizzatore presente sul **Modulo**, o più precisamente in funzione del loro *numero di linee* e del *numero di caratteri per linea*:

Linee & Caratteri	Posizione carattere	Indirizzo in DDRam	
		sulla linea 0	sulla linea 1
2 x 16	00 a 15	da 00H a 0FH	da 40H a 4FH
2 x 20	00 a 19	da 00H a 13H	da 40H a 53H
2 x 24	00 a 23	da 00H a 17H	da 40H a 57H
2 x 32	00 a 31	da 00H a 1FH	da 40H a 5FH
2 x 40	00 a 39	da 00H a 27H	da 40H a 67H

- Per assicurare la *circolarità della scrittura*, cioè per passare a capo linea al termine della riga di stampa corrente è necessario tener presente i valori finali (funzione del *numero di caratteri per linea*), sapendo che l'indirizzo iniziale è sempre 00H (sulla *prima linea*) o 40H(sulla *seconda linea*).
- Inoltre, in ogni caso, il *codice operativo* dell"istruzione necessaria per forzare la *posizione corrente* del *cursore* nel **Contatore di indirizzo** (**AC**, Address Counter): è sempre uguale a quella dell'indirizzo con bit7 a 1, cioè per esempio con *40 caratteri per linea*, da **80**H a **A7**H sulla *prima linea* e da **C0**H a **E7**H sulla *seconda linea*.

```
;┌──────────────────────────────────────────────────────────────────────┐
;|PROCEDURA  : Addr_LCD                                                 |
;|DESCRIZIONE: Predispone l'indirizzo del carattere da visualizzare sull'LCD;|
;|             in pratica ciò significa  indirizzare  locazione  della  DDRam|
;|             (memoria Dati del Display) che corrispondente  alla  posizione|
;|             desiderata; poichè le locazioni sono 80, per indirizzarle sono|
;|             sufficienti 7 bit (2^7=128),  visto  che 6 (2^6=64)  non  sono|
;|             sufficienti.                                             |
;|             Dunque non tutte le 128 combinazioni sono necessarie     |
;|             E' comunque chiaro che il controller HD44780  è  in  grado  di|
;|             immagazzinare più caratteri di  quanti il visualizzatore possa|
;|             mostrare; per esempio, quelli ad 1 linea gestiscono fino  a 40|
;|             caratteri, per cui solo una metà [da 0 (00H) a 39 (27H)] delle|
;|             locazioni è copiata sul display; la rimanente  metà  può  però|
;|             essere fatta scorrere nelle medesime 40 posizioni del Display,|
;|             con le istruzioni descritte in precedenza...            |
;|             In conclusione:                                          |
;|             1) con visualizzatori ad 1 linea (N=0) gli indirizzi possibili|
;|                vanno da 00H (=0, comando 80H)  a  4FH (=79, comando CFH)  |
;|             2) con visualizzatori ad 2 linee (N=1) gli indirizzi:    |
;|                della 1.ma vanno da 00H (=0,  add 80H) a 27H(= 39, add A7H)|
;|                della 2.da vanno da 40H (=64, add C0H) a 67H(=103, add E7H)|
;└──────────────────────────────────────────────────────────────────────┘
Addr_LCD:MOV    AL,CS:[Posiz] ; Assume il valore corrente dell'indirizzo DDRam
         INC    AL            ; Incrementa l'indirizzo della locazione DDRam
         CMP    AL,10H        ;┐Se la posizione è in fondo alla prima linea si
         JE     Addr_L0       ;┘forza il cursore a capo della linea successiva
         CMP    AL,50H        ;┐
         JNE    Addr_L1       ;|Se la posizione è in fondo alla seconda  linea
         MOV    AL,00H        ;|si forza il cursore a capo  della  prima linea
         JMP    SHORT Addr_L1 ;|
                              ;┘
                              ;┐Fissa l'indirizzo interno della memoria  DDRam
Addr_L0:MOV     AL,40H        ;|del controller LCD per puntare l'indirizzo 40H
                              ;|corrispondente alla posizione del primo carat-
                              ;┘tere sulla seconda linea
                              ;┐Predispone la successiva chiamata di Out_By8
Addr_L1: MOV    AH,00H        ;|per provvedere alla scrittura di un comando
                              ;┘cioè a portare il segnale RS=0, istruzione.
                              ;┐Annota la posizione corrente del cursore nella
Addr_L2:MOV     CS:[Posiz],AL ;|memoria DDRam e ricostruisce l'istruzione  per
         OR     AL,80H        ;|per il controller LCD, forzando (con la OR) il
         CALL   Out_By4       ;|bit7 a 1,  come  prevede  l'istruzione  a  ciò
         RET                  ;|delegata; l'istruzione sarà del  tipo 1xxxxxxx
                              ;|da 80H (0,0) a A7H (0,39) sulla  prima  linea
                              ;┘da C0H (1,0) a E7H (1,39) sulla seconda linea
```

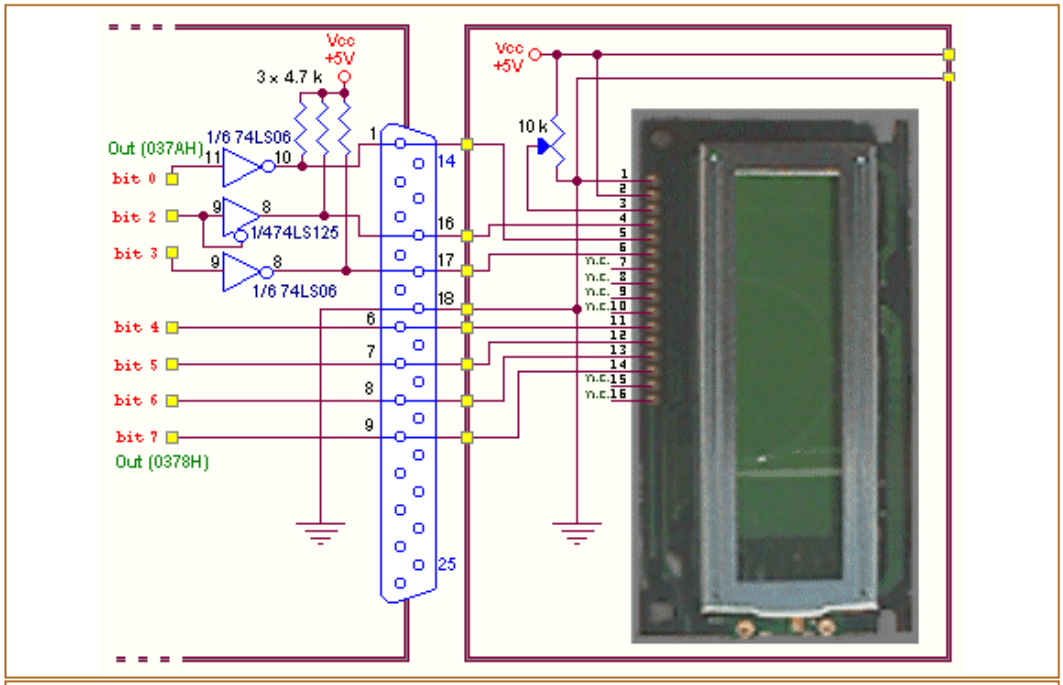
Effetto visivo con asterischi rotanti

- Terminiamo la trattazione con la descrizione di una procedura a margine, attivata al termine del programma, in attesa di tornare al DOS.
- Si tratta di una sequenza di istruzioni *ripetuta ad oltranza*, compresa tra quella identificata dall'etichetta `Giochino` e la `JZ Giochino`, che può essere interrotta *non appena viene premuto* un *tasto qualunque*.
- Data l'irrilevanza del suo compito lascio al lettore il piacere di scoprire come funziona....

```
;┌──────────────────────────────────────────────────────────────────────┐
;|PROCEDURA  :   Giochino                                               |
;|DESCRIZIONE:   Esegue un effetto visivo, in attesa di terminare       |
;└──────────────────────────────────────────────────────────────────────┘
TabGioc DB  33,13, 80H,48,13, 8FH ;┐
        DB  34,13, 81H,47,13, 8EH ;|
        DB  35,13, 82H,46,13, 8DH ;|Tabella interna di "Giochino"
        DB  35,14,0C2H,46,14,0CDH ;|
        DB  34,14,0C1H,47,14,0CEH ;|
        DB  33,14,0C0H,48,14,0CFH ;┘
        ;-------------------------------------------------------------------
GiroChr:PUSH    AX              ;┐    33 35        46 48
        MOV     DL,CS:[SI]      ;|    ┌─ ──────────── ─┐
        MOV     DH,CS:[SI+1]    ;|    |▒▒Giobe2000 ▒▒|  riga 13
        M_Chr   AL,DH,DL,2AH    ;|    |▒▒Benvenuti!▒▒|  riga 14
        PUSH    AX              ;|    └──────────────┘
        MOV     AL,CS:[SI+2]    ;|Procedura interna di "Giochino"
        MOV     AH,00H          ;|
        CALL    Out_By4         ;|  80H 83H      8DH 8FH  istruzione necessaria
        POP     AX              ;|   ┌─ ──────────── ─┐
        MOV     AH,01H          ;|   |▒▒Giobe2000 ▒▒|  riga 13
        CALL    Out_By4         ;|   |▒▒Benvenuti!▒▒|  riga 14
        POP     AX              ;|   └─ ──────────── ─┘
        RET                     ;┘  C0H C3H      CDH CFH  istruzione necessaria
        ;-------------------------------------------------------------------
Giochino:MOV    CX,6            ;┐   Giochino.... un attesa della pressione
        LEA     SI,TabGioc      ;|   del  tasto  per  terminare.............
Giocol: MOV     AL,'*'          ;|┐
        CALL    GiroChr         ;|||
        ADD     SI,3            ;|||
        CALL    GiroChr         ;|||
        SUB     SI,3            ;|┘
        PUSH    CX              ;|┐
        MOV     CX,40           ;|||
Gioco2: CALL    Delay           ;|||
        LOOP    Gioco2          ;|||
        POP     CX              ;|┘
        MOV     AL,' '          ;|┐
        CALL    GiroChr         ;|||
        ADD     SI,3            ;|||
        CALL    GiroChr         ;|||
        ADD     SI,3            ;|┘
        PUSH    DX              ;|┐
        PUSH    CX              ;|||
        PUSH    SI              ;|||
        MOV     AH,01H          ;|||Controlla se è premuto un tasto....
        INT     16H             ;|||... se è così termina il giochino
        POP     SI              ;|||
        POP     CX              ;|||
        POP     DX              ;|┘
        JNZ     Gioco3          ;|
        LOOP    Giocol          ;|
        JMP     Giochino        ;|
Gioco3: CALL    KEYwait         ;|┐Se non è ESC non si esce !!
        CMP     AL,ESC_         ;|||
        JNE     Giochino        ;|┘
        RET                     ;┘
```

- La possibilità di porre fine all'esecuzione ciclica di queste istruzioni è affidata alla Funzione **01**H dell'**INT 16H**, che provvede appunto alla *lettura della tastiera*:
  - se nessun tasto viene premuto la Funzione lascia la flag di Zero a **0** e il loop viene automaticamente ripetuto
  - se si *preme* un *tasto qualunque* la Funzione forza la flag di Zero a **1**, il loop ha termine e viene eseguita l'istruzione successiva a `JZ Giochino`, cioè il programma ha termine

Schema dell'Interfaccia



I *moduli LCD* sono frequentemente gestiti dal *controller HD44780*, l'interfaccia tra il *visualizzatore a cristalli liquidi* (la parte più evidente del modulo) e il *computer* chiamato alla gestione; per questo il progetto utilizza la *porta parallela SPP*, *LPT1*, generando da software le *linee di controllo* necessarie.

Descrizione dello schema

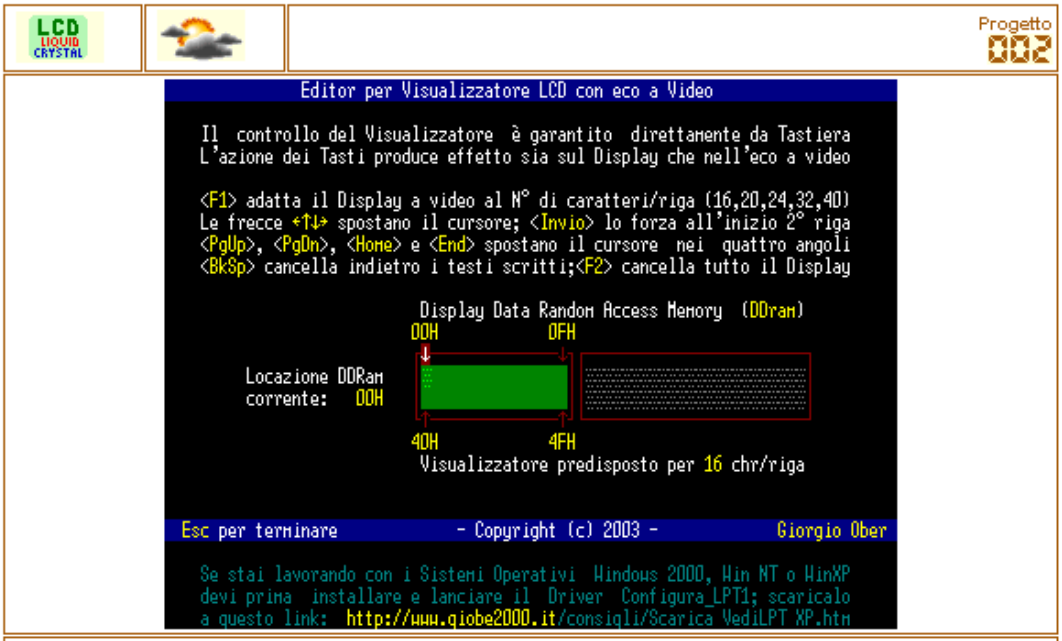
- Per seguire il commento sui **dettagli tecnici** può essere conveniente aprire una seconda finestra, con lo **schema** dell'interfaccia.
- Il progetto utilizza un **Modulo LCD**, completamente programmabile attraverso il suo *controller*, nel nostro caso l'**integrato HD44780** della Hitachi, alimentato *esternamente* con un piccolo alimentatore da 5 volt stabilizzati.
- Il progetto prevede, per il controllo del **Modulo LCD**, la programmazione della *porta parallela* standard **SPP** (**S**tandard **P**arallel **P**ort) **LPT1**; poichè non è prevista alcuna lettura sono stati coinvolti solo i 2 **Registri** d'uscita:
  - il **Registro** d'uscita **0378H/0278H**: coinvolge i suoi 4 bit più significativi, dal bit**7** al bit**4** (disponibili rispettivamente dal pin **9** al pin **6** del connettore **DB-25** della *porta parallela*) per organizzare il *bus dati a* **4** *bit* del **Modulo**
  - il **Registro** d'uscita **037AH/027AH**: coinvolge 3 dei suoi 4 bit per creare da software i 3 *segnali* necessari:
    - il bit**0**, disponibile sul pin **1** della parallela (*strobe* per l'interfaccia *Centronics*), usato per generare il segnale *Read/Write* (**R**/$\overline{\mathbf{W}}$, pin **5** del **Modulo**); deve essere posto a **0** se il dato è *scritto*, e a **1** se è *letto*
    - il bit**2**, disponibile sul pin **16** della parallela (*init* per l'interfaccia *Centronics*), usato per generare il segnale *Register Select* (**RS**, pin **4** del **Modulo**); serve per indicare la natura dell'informazione presente sul *bus dati*: deve essere posto a **0** se il dato rappresenta un *comando da eseguire* (*istruzione*), oppure a **1** se si tratta di *dati* (*caratteri*) da scrivere sul visualizzatore
    - il bit**3**, disponibile sul pin **17** della parallela (*select in* per l'interfaccia *Centronics*), usato per generare il segnale *Enable signal* (**E**, pin **6** del **Modulo**); viene forzato a **1** se il dispositivo è *pronto a gestire* un *dato* o un *comando* predisposto sul *bus dati*
    - va sottolineato che il bit**0** ed il bit**3** sono sottoposti ad un'*inversione logica interna*, prima di essere disponibili sul connettore della porta
- L'osservazione dello schema mette in evidenza anche che:
  - è necessaria una fonte di **alimentazione esterna** (di **+5V**).
  - la **massa** dell'alimentatore deve essere unita con quella della porta parallela: la conoscenza del suo connettore suggerisce la disponibilità di ben 8 pin (dal pin **18** al pin **25**); per questa realizzazione si è scelto il pin **18**.

La descrizione del programma ASM chiamato a gestire il progetto ha già evidenziato la necessità di **conoscere a fondo** le caratteristiche del *controller* **HD44780**; sebbene la cosa ti sia ormai ampiamente nota, ti ricordo i link che descrivono il suo funzionamento hardware e le sue necessità software.

- La scheda dovrà dunque prevedere un **connettore da 8 contatti**, tutti verso la porta parallela, **più 2** per l'alimentatore.
- Puoi disporre di dettaglio migliore consultando Tabella dei collegamenti tra i *14* pin del **Modulo** e *8* dei pin del connettore DB25 femmina della *porta parallela*.

Progetto n°002 - Modulo Visualizzatore LCD su parallela LPT1

Gestione Modulo LCD: editor avanzato 2x16,20,24,32,40 (con controller HD44780)



Gestione di un modulo LCD: il progetto è in grado di collaudare **qualunque** *Display LCD* a *2 linee*, offrendo la possibilità di preselezionare **16**, **20**, **24**, **32** o **40** caratteri per linea; realizza un **Editor AVANZATO**, grado di accettare qualunque carattere Ascii (prodotto della pressione di *tasti*) ed interpretando alcuni *tasti di controllo* (*Invio*, *BkSp*) e *di movimento* (le *frecce*, *PgUp*, *PgDn*, *Home* e *End*) per spostare il cursore senza scrivere. L'interfaccia grafica è particolarmente curata e coinvolge accorgimenti e tecniche di grande prestigio, adattando dinamicamente la dimensione del visualizzatore sia su *LCD* che sul *monitor*.

Presentazione - Analisi del Problema

- Il progetto raccoglie le esperienze dei 3 precedenti per fornire un prodotto di eccellente qualità; il suo compito è quello di collaudare un **qualunque** **Visualizzatore a Matrice di Punti** a cristalli liquidi, organizzato su *2 linee*, **indipendentemente dal numero** di *caratteri per linea*.
- Per il collaudo è ripresa la realizzazione di un **Editor AVANZATO per LCD**, ora *curato in ogni dettaglio*, anche nella *proposta a video*; la pressione del tasto *F1* consente ora di predisporre il numero di *caratteri* previsti (***16***, ***20***, ***24***, ***32*** o ***40***) su ognuna delle 2 righe del **Modulo LCD**.
- I *caratteri*, scritti contemporaneamente sul *display LCD* e sul *visualizzatore a video* (*eco a* ***video***), e i *comandi per l'Editing* sono assicurati dalla pressione di tasti, in virtù dell' interfacciamento con la ***tastiera*** del PC.
- La realizzazione dei controlli di Editing è legata alla conoscenza del modo con cui il *controller* **HD4478** gestisce i *caratteri* sul suo *display*.
- Ciascuna *posizione LCD* corrisponde (come è noto) ad una delle **80** locazioni di *RAM per i dati* (*DDRam*, **D**isplay **D**ata **R**andom **a**ccess **m**emory), indirizzate con *numeri esadecimali* organizzati in gruppi di 40, da **00**H a **27**H sulla *prima riga* e da **40**H a **67**H sulla *seconda*.
- Risulta evidente che, se la dimensione della linea del *display LCD* è minore di ***40 caratteri per riga***, per assicurare la *circolarità della scrittura*, cioè per passare a capo linea al termine di quella corrente, bisogna intervenire in modo dinamico sul *byte d'indirizzo* della *DDRam*, stabilendo il valore massimo accettabile su ciascuna di esse:

Caratteri per riga	Indirizzo in DDRam	
	sulla linea 0	sulla linea 1
16	da 00H a 0FH	da 40H a 4FH
20	da 00H a 13H	da 40H a 53H
24	da 00H a 17H	da 40H a 57H
32	da 00H a 1FH	da 40H a 5FH
40	da 00H a 27H	da 40H a 67H

- Oltre alla circolarità della scrittura dei caratteri assunti da tastiera, il controllo dell'**Editor per LCD** garantisce comunque:
  - l'intervento sulla posizione di stampa, *spostando il cursore* senza scrivere, con *Invio* (*manda a capo*) e con ***tutti*** i *tasti di movimento* (le *4 frecce*, *PgUp*, *PgDn*, *Home* e *End*)
  - la correzione dei testi già scritti con *BackSpace* (*cancella indietro*), intervenendo con *spazi* sia nelle locazioni *DDRam* che nell'eco a video.
  - la cancellazione (con *F2*) di entrambi i *display* (riempiendo di spazi sia tutte le locazioni *DDRam* che il campo di acquisizione dell'eco a video)
- Per il controllo del **Modulo LCD** il progetto prevede la programmazione della *porta parallela* standard **SPP** (**S**tandard **P**arallel **P**ort) **LPT1**.

NB: Con i moderni Sistemi Operativi (**Windows NT**, **Windows 2000**, **Windows XP**) **non è più concesso** l'***accesso diretto*** alle **porte di Input/Output** dall'ambiente ***Assembly*** o dai linguaggi di programmazione (***Pascal***, ***Delphi***, ***Visual Basic*** ...), come si poteva fare prima con **Windows 95/98/ME**.

- Quando si tenta, come fa il nostro progetto, un **Output** agli indirizzi Hardware viene generata una segnalazione d'errore di "**istruzione protetta**" o, semplicemente **non succede nulla...**

Analisi del Codice Sorgente

- L'**obiettivo** del progetto è quello di realizzare un **Editor AVANZATO** per **display LCD** a *2 linee* con qualunque numero di *caratteri* (***16***, ***20***, ***24***, ***32*** o ***40***) *per linea*.
- In sintonia con i progetti precedenti, mantiene inalterate le prerogative hardware (schema elettrico e descrizione dello schema) e quelle software legate alla programmazione (tramite le linee d'uscita della *porta parallela LPT1*) del *controller* del **Modulo LCD**, sempre l'**integrato HD44780** della Hitachi.
- Il ***codice assembly*** del programma è visibile, **nella sua totalità**, scorrendo con la barra laterale il *testo* della seguente *casella*:

```
PAGE 66,132
TITLE** PROGRAMMA di GESTIONE di programmi ASSEMBLER  tipo COM    (marzo 2003)
SUBTTL ** TUTORIAL ASSEMBLY  --  www.giobe2000.it   --    by ing. Giorgio OBER
;
;| NOME        : EditLCD2                                                      |
;| AUTORE      : Giorgio OBER                                                  |
;| VERSIONE 2  : marzo 2003                                                    |
;|  ┌──────────┐  ┌──────────────────────────────────────────────────────────┐ |
;|  |▒▒▒▒▒▒▒▒▒▒|  |Questa versione di "Edit LCD"  è  adatta ad ogni modello di| |
;|  |▒VARIANTE▒|  |Display a   2 linee,  da 16, 20, 24, 32 o 40  caratteri per| |
;|  |▒▒▒▒▒▒▒▒▒▒|  |linea; inoltre gestisce il Visualizzatore a  Video  in modo| |
;|  |▒▒▒▒▒▒▒▒▒▒|  |dinamico, ricostruendone la posizione  a  video  a  partire| |
;|  |▒▒▒▒▒▒▒▒▒▒|  |dalle coordinate  rigaIni,colnIni, riferite  all'angolo  in| |
;|  |▒▒▒▒▒▒▒▒▒▒|  |alto a sinistra dentro l'area del Visualizzatore  destinata| |
;|  |▒▒▒▒▒▒▒▒▒▒|  |all'acquisizione dei caratteri,  e  prefissate  nella  zona| |
;|  |▒▒▒▒▒▒▒▒▒▒|  |delle COSTANTI DEFINITE PER IL PROGRAMMA (qui sotto).      | |
;|  └──────────┘  └──────────────────────────────────────────────────────────┘ |
;| DESCRIZIONE: Programma per il collaudo  dell'Interfaccia Software/Hardware|
;|              di un Modulo LCD collegato su porta parallela LPT1.          |
;|              Si tratta di un vero e proprio "Editor" per  Display  LCD, in|
;|              grado di accettare qualunque carattere  Ascii  e  di  gestire|
;|              i tasti di movimento (le frecce     , PgUp, PgDn, Home e End)|
;|              per spostare il cursore senza scrivere, e quelli di controllo|
;|              come Invio (va a capo linea) e BackSpace (cancella indietro).|
;|              <F1> consente di  commutare il controllo da 16 a 40 caratteri|
;|              per riga, e <F2> cancella tutto il Display.                  |
;|              Nella visualizzazione a Video sono presenti accorgimenti   in|
;|              grado di modificare la dimensione standard del  Cursore,  per|
;|              favorire la migliore resa grafica.                           |
```

- La **scrittura di caratteri** su un *visualizzatore LCD* comporta la conoscenza approfondita del controller che lo governa.
- E' necessario conoscere la sua *architettura*, i *codici operativi* (istruzioni) necessari per la sua programmazione, i *diagrammi temporali* che bisogna generare per simulare da SW i segnali HW necessari.

⚠ Per tutto questo è ***assolutamente necessario*** leggere con attenzione le *pagine* dedicate all'integrato **HD44780**.

- In aggiunta, la necessità di garantire l'***editing dei testi*** e il ***controllo del cursore*** rende il progetto particolarmente impegnativo.
- Di tutto questo risente la descrizione del **Main Loop**, la parte principale di un programma, chiamata ora non solo ad *inizializzare il controller* ma anche a *gestire i servizi* associati alla pressione dei tasti autorizzati.

```
        MOV     AL,16           ; Per default è previsto un Visualizzatore con
        MOV     CS:[Limite],AL; 16 caratteri per riga (e 24 nascosti, in LCD)
MOV Byte Ptr CS:[Address],00H; Valore iniziale dell'indirizzo DDRam
        CALL    DESKTOP         ;Inizializza completamente il piano  di  lavoro
                                ;┐Inizializza il controller HD 44780, simulando
                                ;|l'attivazione della linea di RESET;  vengono
        CALL    Pre_Ini         ;|erogati 3 bytes a 30H separati da piccoli
                                ;|ritardi "pesati"; per default viene predisposta
                                ;┘l'interfaccia a 8 bit
                                ;┐Predispone il controller HD 44780 per gestire
                                ;|il visualizzatore nel modo operativo desiderato
                                ;|************** Parametri >>> 001DNFxx >>> 28H
                                ;|    #  DL=0(bit4) > interfaccia a 4 bit
                                ;|    #  N=1 (bit3) > visualizzatore a 2 linee
                                ;|    #  F=0 (bit2) > matrice carattere 5x7 pix
                                ;|****** Cancella Display >>> 00000001 >>> 01H
                                ;|    #  Azzera la memoria DDRam (Clear Display
                                ;|***** Accesso Caratteri >>> 000001IS >>> 06H
        CALL    Mia_Ini         ;|    #  I=1 (bit1) > autoincrem.indirizzo DDRam
                                ;|    #  S=0 (bit0) > display bloccato
                                ;|****** Modo del Display >>> 00001DCB >>> 0FH
                                ;|    #  D=1 (bit2) > display acceso
                                ;|    #  C=1 (bit1) > cursore visibile
                                ;|    #  B=1 (bit0) > a forma lampeggiante
                                ;|*** Scorrimento Display >>> 0001SRxx >>> 14H
                                ;|    #  S/C=0(bit3) > si muove il cursore
                                ;┘    #  R/L=1(bit2) > verso destra
;-------------------------------------------------------------------------------
                                ;┐Fissa la variabile  [Address]  con  il  valore
                                ;|della posizione corrente del  cursore e  copia
                                ;|l'informazione  nel  registro  puntatore della
Rientra:CALL    FissaAdd        ;|memoria DDRam del controller LCD, con  valori:
        CALL    FissaCur        ;|■da 00H (0,0) a 27H (0,39) sulla  prima  linea
                                ;|■da 40H (1,0) a 67H (1,39) sulla seconda linea
                                ;|Inoltre mostra il cursore sul Display a  Video
                                ;┘e stampa il valore della Locaz. DDRam corrente
Rien_0: CALL    KEYwait         ;Aspetta la pressione di tasti autorizzati e
                                ;attiva il servizio richiesto
                                ;
                                ; .... vedi pagina successiva ................
                                ;
                                ;┐Predispone le  successive  chiamate di Out_By8
_OUT:   MOV     AH,00H          ;|per provvedere  alla scrittura  di  un comando
                                ;┘cioè  a  portare  il segnale RS=0, istruzione.
        MOV     AL,01H          ;┐Azzera la  memoria  DDRam (Clear Display ): il
        CALL    Out_By4         ;┘codice operativo necessario è 0000 0001 (=01H)
                                ;┐Inizializza le caratteristiche del display per
        MOV     AL,08H          ;|display  spento (D=bit2=0), cursore invisibile
        CALL    Out_By4         ;|(C=bit1=0) e  a forma a trattino  (B=bit0=0);
                                ;┘il codice operativo è  dunque 0000 1000 (=08H)
        MOV     AH,4CH          ;Torna al DOS
        INT     21H
```

- La *casella di testo* mostra in dettaglio le ***fasi fondamentali*** del progetto:
  - la procedura locale (CALL DESKTOP) organizza la stampa dei messaggi di presentazione (***desktop***) dell'interfaccia grafica, simile a quella coinvolta negli altri progetti
  - non è quindi necessario descriverla in dettaglio; ricordo che il servizio è ottenuto con l'intercessione delle **Procedure** e delle **Macro** appartenenti alle mie 2 librerie, **Giobe.MAC** e **Giobe.LIB**, disponibile in forma sorgente in **Giobe.ASM**.

- inizializza (CALL Pre_ini) il *controller* **HD 44780**, simulando da SW l'attivazione della sua linea di ***reset***; vengono erogati 3 bytes a 30H separati da piccoli ritardi "pesati", e viene predisposta (per default) l'***interfaccia a 8 bit***
- predispone *controller* **HD 44780** per gestire il visualizzatore nel *modo operativo* desiderato: interfaccia a 4 bit, visualizzatore a 2 linee, matrice del carattere con 5x7 pixel, display acceso, cursore visibile lampeggiante, azzeramento della *memoria DDRam*, autoincremento dell'indirizzo, display bloccato
- ad ogni rientro:
  - aggiorna il **Registro** d'indirizzo della *DDRam*, fissando la posizione di stampa corrente, evidenziata anche dal cursore del visualizzatore a video
  - mostra a video (novità) il valore corrente del **Registro** d'indirizzo della *DDRam*
  - aspetta la pressione di tasti, ne filtra il valore e, se ammessi, mette *a video* e sul *visualizatore LCD* i caratteri Ascii stampabili e interpreta quelli di controllo e/o di movimento, richiesti dalla ***fase di editing*** (per la sua particolare importanza questa parte verrà analizzata a fondo nella pagina seguente)
- prima di chiudere lascia pulita la memoria del **display LCD**, rendendone spento anche l'aspetto esteriore (compreso il cursore)
- le ultime istruzioni (MOV AH,4CH,INT 21H) *chiudono* il codice sorgente *restituendo il controllo al sistema operativo* che ci ospita.

- Il ***loop di attesa tasti*** è certamente la più importante *parte di codice assembly* del programma principale.
- La sua efficienza regola e qualifica tutto il progetto, per cui è bene conoscere le tecniche e i trucchi per gestire al meglio la ***tastiera***; se non l'hai ancora fatto ti consiglio di leggere le 5 pagine dedicate alla **Procedura KEYwait**, una delle prestigiose *procedura* della libreria **Giobe.ASM**.
- La ***struttura d'attesa*** è pressocchè identica a quella del progetto precedente ed è raccolta nella seguente *casella di testo*:

```
Rien_0: CALL    KEYwait       ; Aspetta decisioni
                              ;┐Recupera  il  colore "rosso  su  nero"  della
                              ;│cornice  del  campo  di  acquisizione,  per
        CALL    Old_Col       ;│annullare  l'effetto  del  puntatore  grafico
                              ;│lampeggiante imposto dal movimento del cursore
                              ;┘sul Display e sul Visualizzatore a Video
        CMP     AL,00H        ;┐Si tratta di tasto Funzionale...
        JE      KeyFunz       ;┘Va servito a parte
        CMP     AL,Invio      ;┐
        JNE     Rien_1        ;│Servizio per la pressione di <Invio>
        JMP     Serv_Invio    ;┘
Rien_1: CMP     AL,BackSp     ;┐
        JNE     Rien_2        ;│Servizio per la pressione di <BackSpace>
        JMP     Serv_BkSp     ;┘
       ;▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒;┐NB !! per  il  Servizio  di  <BackSpace>
        JMP     Serv_BkSp_    ;│      esiste  una  versione  alternativa
       ;▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒;┘----------------------------------------
Rien_2: CMP     AL,ESC_       ;┐Servizio per la pressione di <ESC>
        JE      _OUT          ;┘
                              ;┐Servizio per la pressione di un  comune  tasto
        JMP     Serv_Chr      ;│Ascii: il carattere corrispondente viene posto
                              ;┘a video e trasmesso al visualizzatore
KeyFunz:CMP     AH,F1_        ;┐
        JNE     KeyFun0       ;│Servizio per la pressione di <F1>
        JMP     Serv_F1       ;┘
KeyFun0:CMP     AH,F2_        ;┐
        JNE     KeyFun1       ;│Servizio per la pressione di <F2>
        JMP     Serv_F2       ;┘
KeyFun1:CMP     AH,Frec_Su    ;┐
        JNE     KeyFun2       ;│Servizio Scorrimento cursore: Freccia
        JMP     Serv_Up       ;┘
KeyFun2:CMP     AH,Frec_Giu   ;┐
        JNE     KeyFuntre       ;│Servizio Scorrimento cursore: Freccia
        JMP     Serv_Dn       ;┘
KeyFuntre:CMP     AH,Frec_Dx    ;┐
        JNE     KeyFun4       ;│Servizio Scorrimento cursore: Freccia
        JMP     Serv_Dx       ;┘
KeyFun4:CMP     AH,Frec_Sn    ;┐
        JNE     KeyFun5       ;│Servizio Scorrimento cursore: Freccia
        JMP     Serv_Sn       ;┘
KeyFun5:CMP     AH,End_Fine   ;┐
        JNE     KeyFun6       ;│Servizio Scorrimento cursore: End (=Fine)
        JMP     Serv_End      ;┘
```

```
KeyFun6:CMP     AH,Home         ;┐
        JNE     KeyFun7         ;│Servizio Scorrimento cursore: Home
        JMP     Serv_Hom        ;┘
KeyFun7:CMP     AH,PgDn         ;┐
        JNE     KeyFun8         ;│Servizio Scorrimento cursore: PgDn
        JMP     Serv_PD         ;┘
KeyFun8:CMP     AH,PgUp         ;┐
        JNE     Rien_0          ;│Servizio Scorrimento cursore: PgUp
        JMP     Serv_PU         ;┘
```

- La chiamata `CALL KEYwait` lascia in **AL** con il *codice Ascii* e in **AH** il *codice di scansione* del tasto premuto, informazioni più che sufficienti per stabilire con certezza la richiesta di un preciso compito da eseguire.

- Il progetto prevede 3 ***categorie*** di tasti e la tecnica di indagine proposta nelle pagine dedicate alla **Procedura KEYwait** stabilisce delle ***priorità*** insindacabili:
  - per primi vanno indagati i **tasti Funzione** (nel nostro caso *F1* e *F2*) e i **tasti Funzionali** (nel nostro caso *End*, *PgUp*, *PgDn*, *Home* e le *4 frecce*), perchè sono caratterizzati dall'avere **codice Ascii** uguale a **00**H; il codice assembly verifica l'appartenenza a queste categorie (`CMP AL,00H`) e si accinge a dedicare loro un supplemento d'indagine (`JE KeyFunz`)
  - i tasti rimanenti sono tutti chiamati a rappresentare un codice **Ascii standard**, per lo più lettere, numeri, interpunzione,...; ma tra essi una piccola categoria va indagata prima degli altri: quelli che portano simboli o parole riconducibili ai caratteri **Ascii di controllo** (nel nostro caso *Invio*, *Backspace* e *Esc*)
  - tutti gli altri sono caratteri Ascii stampabili e subiranno il medesimo trattamento (`JMP Serv_Chr`): finiranno sui 2 *visualizzatori*, *LCD* e *a video*.

- Sebbene il ***dettaglio*** delle istruzioni utilizzate sia comprensibile solo dopo aver studiato con pazienza le più volte citate 5 pagine di **KEYwait** posso farti notare alcune particolarità:
  - la discriminazione tra i **tasti Funzione** e i **tasti Funzionali** avviene con riferimento ad **AH**, `CMP AH,....`; la cosa è chiara se si ricorda che il loro *codice Ascii* è nullo (cioè per essi **AL** è uguale a **00**H) per cui non rimane che affidarci al *codice di scansione* (comunque disponibile in **AH**)
  - per il riconoscimento dei *tasti di servizio* ho fatto ricorso a ***etichette***, piccole ***parole*** onomatopeiche, definite all'inizio del programma con l'aiuto della pseudooperazione **EQU**, chiamate a sostituire i ***numeri*** dei rispettivi *codici* (*ascii* o *di scansione*), difficili da ricordare (l'elenco completo è scaricabile qui)
  - quando un tasto viene riconosciuto si prowede al suo servizio ***saltando*** ad una zona di codice ad esso dedicata (per esempio `JMP Serv_F1` o `JMP Serv_Up`...); è assolutamente **importante** ricordare che questi *codici di Servizio* (documentati tra qualche pagina) **non sono procedure**, cioè non devono terminare con `RET` ma, a loro volta, devono **rientrare *con un salto* all'inizio del loop d'attesa**, cioè `JMP Rientra`
  - una cosa altrettanto **importante** (pena errori del tipo `error A2053: Jump out of range by nnn byte(s)`) è quella di imparare a **rovesciare la logica**; si tratta di un concetto semplice ed efficace per gestire i *salti condizionati* (come `JE`): se il punto da raggiungere è troppo lontano il compilatore si infastidisce e segnala `Jump out of range` (*salto fuori range*... non ce la faccio), aggiungendo di quanti bytes siamo oltre il possibile (`..by 27 bytes`). Se per esempio `JE Serv_F1` non ce la fa, basta *rovesciare la logica*, cioè saltare *subito sotto* se `NE` (`JNE KeyFun0`) saltando a Serv_F1 *in modo incondizionato*, `JMP Serv_F1` (poichè i servizi di tutti i tasti sono molto distanti dal punto che ne rileva la necessità, questa tecnica è applicata per ciascun di essi)

Analisi delle procedure importanti

- Come osservato nel commento del progetto precedente (del quale questo è un'evoluzione) la realizzazione di un **Editor AVANZATO per Visualizzatori LCD** si sviluppa su 3 livelli:
  - la *gestione hardware*, cioè la creazione dei segnali richiesti per pilotare il **Modulo LCD** con i **Registri** d'uscita della *porta parallela* standard **SPP** (**S**tandard **P**arallel **P**ort) **LPT1**
  - la *gestione software* del *controller* **HD44780** del **Modulo LCD**, cioè l'*inizializzazione* e la *personalizzazione* del suo funzionamento, in accordo con le nostre specifiche esigenze.
  - il controllo dele operazioni di Editing, legato alla lettura della ***tastiera*** del PC e all'interpretazione e al Servizio dei tasti premuti
- Naturalmente solo il terzo punto porterà differenze sostanziali, rispetto alle **procedure**, richieste dai primi 2, praticamente **le stesse** documentate nel primo progetto, qui riassunte per sommi capi (se vuoi conoscere i dettagli accedi ai vari link):

Ricostruzione temporale dei segnali con **Bus a 8 bit**

- dettaglio sulla **Procedura Out_By8**, necessaria per ricostruire la *sequenza temporale* richiesta (nelle *varie fasi* della scrittura) sui segnali **E** (pin 6 del modulo), **RS** (pin 4) e **R/W** (pin 5) del *controller* **HD44780**, assicurando le condizioni ideali per l'*output* verso il **Modulo LCD**, con l'aiuto del **Registro** d'uscita **037AH**/**027AH** della porta parallela.

```
;┌──────────────────────────────────────────────────────────────────────┐
;|PROCEDURA  :   Out_By8                                                |
;|DESCRIZIONE:   In ingresso AL deve essere predisposto con il  byte esadeci-|
;|               male da spedire al controller HD 44780,  attraverso il regi-|
;|               stro di dato (0378H) della porta parallela.           |
;|               Il valore presisposto in AH serve per predisporre il segnale|
;|               RS a 0 (AH=00H) se il byte da scrivere è un codice operativo|
;|               (istruzione) o RS a 1 (AH=01H) se invece deve essere scritto|
;|               un carattere (dato).                                  |
;|               La procedura ricostruisce la  sequenza  temporale  richiesta|
;|               sui segnali RS, R/W e ENABLE del  controller  HD  44780  del|
;|               modulo visualizzatore, assicurando le condizioni ideali  per|
;|               l'output verso il modulo LCD, con l'aiuto  del  registro  di|
;|               controllo (037AH) della porta parallela.              |
;└──────────────────────────────────────────────────────────────────────┘
```

Ricostruzione temporale dei segnali con **Bus a 4 bit**

- dettaglio sulla **Procedura Out_By4**, chiamata a gestire la scrittura di un *carattere* (dato, se in ingresso **AH**=01H, **RS** a **1**) o di un *codice operativo* (istruzione, se in ingresso **AH**=00H, **RS** a **0**) ponendo l'originario *dato ad 8 bit* sui *4 bit* più significativi del *bus* (DB7÷DB4, *interfaccia a 4 bit*) prima la **parte alta** del byte da scrivere e poi **quella bassa**, con l'aiuto di un *multiplexer dedicato*, integrato nel chip.

```
;┌──────────────────────────────────────────────────────────────────────┐
;|PROCEDURA  :   Out_By4                                                |
;|DESCRIZIONE:   Questa procedura organizza la gestione dei dati verso    il |
;|               mudulo visualizzatore con INTERFACCIA A 4 BIT !! (di  fatto |
;|               il byte da scrivere viene proposto sul bus in 2 tempi).   |
;|DESCRIZIONE:   In ingresso AL deve essere predisposto con il  byte esadeci-|
;|               male da spedire al controller HD 44780,  attraverso il regi-|
;|               stro di dato (0378H) della porta parallela.           |
;└──────────────────────────────────────────────────────────────────────┘
```

Inizializzazione del *controller*: simulazione dell'azione del segnale di RESET

- dettaglio sulla **Procedura Pre_Ini**, chiamata ad *inizializzare* il *controller* **HD44780**, simulando l'azione di un segnale di ***Reset***, con la *sequenza di istruzioni standard*, sempre necessaria all'inizio di ogni sessione.

```
;┌──────────────────────────────────────────────────────────────────────┐
;|PROCEDURA  :   Pre_Ini                                                |
;|DESCRIZIONE:   Inizializza il controller HD 44780, simulando l'attivazione |
;|               della linea di RESET;  vengono  erogati  3  bytes  a  30H |
;|               separati da  piccoli  ritardi  pesati;  per  default  viene |
;|               predisposta l'interfaccia a 8 bit, senza curarsi, per ora,  |
;|               della forma del visualizzatore e della matrice del carattere|
;└──────────────────────────────────────────────────────────────────────┘
```

Inizializzazione del *controller*: predisposizione delle modalità di visualizzione

- dettaglio sulla **Procedura Mia_Ini**, chiamata ad *inizializzare* il *controller* **HD44780** in funzione delle nostre specifiche esigenze di visualizzazione, con la *sequenza di istruzioni personalizzata*; in questo progetto:
  - conferma la scelta per l'interfaccia a **4 bit**, cioè l'intenzione di usare la ***metà alta*** del *bus dati* (**20**H)
  - predispone la visualizzazione a **2** linee e una matrice del carattere con **5x7** pixel (**28**H)
  - azzera la *memoria DDRam* (**01**H)
  - fissa il modo di gestire il display con *autoincremento dell'indirizzo* e per *display bloccato* e per *movimento del cursore* (**06**H)
  - inizializza le caratteristiche del display per *display acceso*, *cursore visibile* e *a forma lampeggiante* (**0F**H); il primo progetto usava, per altro, *cursore invisibile* e *a forma fissa* (**0C**H)
  - fissa il modo di gestire il movimento del cursore predisponendo *spostamento del solo cursore verso destra* (**14**H)
  - fissa in uscita il cursore in modo da puntare la prima locazione della *memoria DDRam*, tenendo memoria anche nella *variabile locale* [*Address*]

```
;┌──────────────────────────────────────────────────────────────────────┐
;|PROCEDURA  :   Mia_Ini                                                |
;|DESCRIZIONE:   Predispone il controller HD 44780 per gestire il visualizza-|
;|               tore nel modo operativo desiderato, comunque con interfaccia|
;|               a 4 bit:                                               |
;| Cancella Display  00000001 01H  Azzera la memoria DDRam (Clear Display)   |
;| Accesso Caratteri 000001IS 06H  #  I=1 (bit1) > autoincrem.indirizzo DDRam|
;|                                 #  S=0 (bit0) > display bloccato        |
;| Modo  del Display 00001DCB 0FH  #  D=1 (bit2) > display acceso           |
;|                                 #  C=1 (bit1) > cursore visibile        |
;|                                 #  B=1 (bit0) > cursore lampeggiante    |
;| Scorrimento Displ 0001SRxx 14H  # S/C=0(bit3) > si muove il cursore      |
;|                                 # R/L=1(bit2) > verso destra            |
;| Parametri         001DNFxx 28H  #  DL=0(bit4) > interfaccia a 4 bit      |
;|                                 #  N=1 (bit3) > visualizzatore a 2 linee |
;|                                 #  F=0 (bit2) > matrice carattere 5x7 pix |
;└──────────────────────────────────────────────────────────────────────┘
```

Scrittura del Dato nella memoria del controller [1 di 2]

- La **Procedura Sta_chr**, che si occupa di ***scrivere il carattere*** sul *visualizzatore* del **Modulo LCD** e di ***farne eco*** sul *monitor*, è sostanzialmente simile a quella descritta nel progetto precedente; data l'alta specializzazione del suo compito nella versione attuale i compiti di stampa sono ben distinti:
  - all'inizio si occupa della gestione dei *caratteri* del *display LCD*, provvedendo a:
    - spedire (`CALL Out_By4`) il *carattere* al *controller* **HD44780** del **Modulo LCD**, dopo averlo avvisato (`MOV AH,01H`) che si tratta di un dato (**RS=1**)
    - ricostruire l'*indirizzo* in *memoria DDram* relativo alla posizione associata al carattere successivo, al fine di garantire la *circolarità* della scrittura in presenza dei limiti imposti dalle possibili *diverse dimensioni* (***16***, ***20***, ***24***, ***32*** o ***40*** *caratteri per riga*) del *display LCD*
    - passare il nuovo indirizzo *controller* **HD44780** del **Modulo LCD**, con l'aiuto della `CALL FissaAdd` (descritta nella pagina seguente)
  - in uscita si occupa della stampa sul *visualizzatore a video*:
    - il *carattere* è posto nella posizione di stampa corrente, con l'aiuto della macro locale **`M_chr`**
    - le *coordinate* del cursore *video* (espresse in *riga*, *colonna*) sono rielaborate (e lasciate rispettivamente in **DH**, **DL**) per puntare il carattere successivo
    - il corsore viene mostrato nella nuova posizione, con l'aiuto della `CALL FissaCur` (descritta nella pagina seguente)

```
;┌──────────────────────────────────────────────────────────────────────┐
;|SERVIZIO   : Serv_Chr                                                 |
;|DESCRIZIONE: Servizio per la pressione di un qualunque tasto Ascii; il|
;|             corrispondente carattere, trovato in ingresso in AL, viene|
;|             posto a video nel campo del Visualizzatore e trasmesso al|
;|             controller LCD; la posizione di stampa viene aggiornata per|
;|             puntare la posizione del carattere successivo, in funzione del|
;|             tipo di Visualizzatore, attualmemente in uso.            |
;└──────────────────────────────────────────────────────────────────────┘
Serv_Chr:PUSH   AX              ;Salva il codice Ascii del carattere da stampare
        ;***********************;******* Servizio di Visualizzazione su LCD ****
                                ;┐Predispone la  successiva  chiamata di Out_By8
         MOV    AH,01H          ;|per provvedere alla scrittura  di un carattere
                                ;┘cioè a portare il segnale RS=1, carattere
         CALL   Out_By4         ;┐Trasmette al controller LCD il carattere in AL
                                ;|(RS=1) e aggiorna la posizione del cursore per
                                ;┘puntare il carattere successivo
                                ;|corrispondente alla posizione del primo carat-
                                ;┘tere sulla seconda linea
         MOV    AH,CS:[Limite]; Dimensione del visualizzatore LCD in uso
         MOV   AL,CS:[Address];┐Assume  il  valore  corrente  dell'indirizzo
         INC    AL             ;┘interno DDRam e ne incrementa il valore
         CMP    AL,AH          ;┐Se la posizione è in fondo alla prima linea si
         JE     Srv_C0         ;┘forza il cursore a capo della linea successiva
         ADD    AH,40H         ;┐
         CMP    AL,AH          ;|Se la posizione è in fondo alla seconda  linea
         JNE    Srv_C1         ;|si forza il cursore a capo  della  prima linea
         MOV    AL,00H         ;|
         JMP    SHORT Srv_C1   ;┘
                                ;┐Fissa l'indirizzo interno della memoria  DDRam
Srv_C0: MOV     AL,40H         ;┘del controller LCD per puntare l'indirizzo 40H
```

```
                                ;┐Annota la posizione corrente del cursore nella
                                ;│memoria DDRam e ricostruisce l'istruzione  per
                                ;│per il controller LCD, forzando (con la OR) il
Srv_C1: CALL    FissaAdd        ;│bit7 a 1,  come  prevede  l'istruzione  a  ciò
                                ;│delegata; l'istruzione sarà del  tipo 1xxxxxxx
                                ;│da 80H (0,0) a A7H (0,39) sulla   prima  linea
                                ;┘da C0H (1,0) a E7H (1,39) sulla  seconda linea
       ;**********************;****** Servizio di Visualizzazione a Video ****
        POP     AX              ;Recupera codice Ascii del carattere da stampare
        M_Chr   AL,DH,DL,2AH    ; Stampa il carattere sul Display a Video

        INC     DL              ; Incrementa la colonna
        MOV     AL,colnIni      ;┐AL= colonna iniziale in entrambe le linee
        MOV     AH,CS:[Limite];│Dimensione del visualizzatore LCD in uso
        ADD     AH,AL           ;┘AH= colonna successiva all'ultima possibile

        CMP     DL,AH           ;┐Se la posizione è in fondo alla prima o alla
        JNE     Srv_C3          ;│seconda linea si forza il cursore a capo della
                                ;┘linea opposta a quella corrente
Srv_C2: MOV     DL,AL           ;┐colnIni
        INC     DH              ;│Fissa  il  cursore  a  capo  dell'altra  linea
        MOV     AH,rigaIni      ;│
        ADD     AH,2            ;│
        CMP     DH,AH           ;│(rileva  automaticamente se  quella  superiore
        JNZ     Srv_C3          ;│o se quella inferiore)
        MOV     DH,rigaIni      ;┘
Srv_C3: CALL    FissaCur        ;┐Mostra il cursore sul Display a Video e stampa
        JMP     Rien_0          ;┘il valore della Locazione DDRam corrente
```

Scrittura del Dato nella memoria del controller [segue, 2 di 2]

- La **SottoProcedura** **FissAdd** ha la dignità e lo spazio di una solo perchè è citata più volte nell'ambito del programma; in realtà essa ha il compito di *fissare* l'indirizzo in *DDram*:
  - con `MOV AH,00H` conferma alla **Out_By4** la necessità di trattare il byte da scrivere come istruzione (**RS=0**)
  - ricostruisce automaticamente il *codice operativo* dell"istruzione necessaria per forzare la *posizione corrente* del *cursore* nel **Contatore di indirizzo** (AC, Address Counter), sempre uguale al valore dell'indirizzo con bit7 a **1**, cioè:

Caratteri per riga	codice operativo Address	
	sulla linea 0	sulla linea 1
16	da 80H a 8FH	da C0H a CFH
20	da 80H a 93H	da C0H a D3H
24	da 80H a 97H	da C0H a D7H
32	da 80H a 9FH	da C0H a DFH
40	da 80H a A7H	da C0H a E7H

  - con `CALL Out_By4` trasmette il comando (*istruzione*) al *controller* **HD44780** del **Modulo LCD**

```
;┌──────────────────────────────────────────────────────────────────────┐
;│PROCEDURA   : FissaAdd                                                │
;│DESCRIZIONE: Indirizza la locazione della DDRam (memoria Dati del Display) │
;│             che corrispondente alla posizione di stampa corrente,  quella │
;│             nella quale sarà visibile il cursore, sia  sul Visualizzatore │
;│             LCD che in quello simulato a video; aggiorna anche il  valore │
;│             contenuto nella variabile locale che memorizza la posizione.  │
;└──────────────────────────────────────────────────────────────────────┘
FissaAdd:                       ;┐Predispone la successiva chiamata di Out_By8
        MOV     AH,00H          ;│per provvedere alla scrittura di un comando
                                ;┘cioè a portare il segnale RS=0, istruzione.
                                ;┐Annota la posizione corrente del cursore nella
        MOV     CS:[Address],AL;│memoria DDRam e ricostruisce l'istruzione  per
        OR      AL,80H          ;│per il controller LCD, forzando (con la OR) il
        CALL    Out_By4         ;│bit7 a 1,  come  prevede  l'istruzione  a  ciò
                                ;│delegata; l'istruzione sarà del  tipo 1xxxxxxx
                                ;│da 80H (0,0) a A7H (0,39) sulla  prima  linea
        RET                     ;┘da C0H (1,0) a E7H (1,39) sulla seconda linea
```

- Data la *sofisticata* tecnica di gestione prevista dal programma, anche il *cursore video* dispone di una **SottoProcedura** specifica, la **FissCur**:
  - mostra il valore numerico dell'indirizzo in *DDram*, stampandolo alle coordinate [*Riga_xy*],[*Coln_xy*] con l'aiuto della `CALL Add_byt`
  - il servizio è sostanzialmente assicurato dalla potente **Procedura Byt2Asc**, in grado di tradurre il *numero esadecimale* dell'*indirizzo* nei 2 caratteri Ascii corrispondenti, ed è connotato da un tocco di classe, con l'aggiunta della H finale, gestita dalla **Procedura BIOchr1**, come la precedente appartenente alla libreria **Giobe.ASM**
  - gestisce la visualizzazione del cursore (`CALL Segna_Cur`, documentata solo nel sorgente scaricabile).

```
;┌──────────────────────────────────────────────────────────────────────┐
;|PROCEDURA  : FissaCur                                                 |
;|DESCRIZIONE: Mette a Video il Cursore "▒", nella posizione indicata da DH e|
;|             DL (riga e colonna) provvedendo prima a salvare  il  carattere|
;|             che sarà coperto dalla sua forma,  e  indicando  in  posizione|
;|             relativamente costante il valore  dell'indirizzo  corrente  in|
;|             DDRam (memoria Dati del Display)                         |
;└──────────────────────────────────────────────────────────────────────┘
FissaCur:PUSH   DX              ;┐
         MOV    DH,CS:[Riga_xy];|
         MOV    DL,CS:[Coln_xy];|Stampa dinamica del testo di commento alla
         M_COLOR 0EH            ;|Locazione DDRam corrente
         MOV    AL,CS:[Address];|
         CALL    Add_Byt        ;|
         POP     DX             ;┘
         PUSH    AX
         CALL    RAMcur         ;┐Assume   il   carattere   che   sarà   coperto
         MOV     AL,ES:[DI]     ;|direttamente  dalla  RamVideo,  e  ne fa copia
         MOV     CS:[OldChr],AL;┘nella variabile locale [OldChr]
                                ;┐Mostra un puntatore sulla cornice del campo di
         CALL    Segna_Cur      ;|acquisizione in "bianco lampeggiante su rosso"
                                ;|per evidenziare  il  movimento del cursore sul
                                ;┘Visualizzatore a Video.
         M_CURSOR DH,DL         ; Mostra il cursore sul Display a Video
       ; M_Chr    '▒',DH,DL,2AH ; <<<<< alternativa per cursore fisso e coprente
         POP     AX
         RET

;┌──────────────────────────────────────────────────────────────────────┐
;|PROCEDURA  : Add_Byt                                                  |
;|DESCRIZIONE: Stampa a video  l'indirizzo  della  DDRAM  corrispondente agli|
;|             attuali 4 angoli del Visualizzatore LCD                  |
;└──────────────────────────────────────────────────────────────────────┘
Add_Byt:PUSH    AX
         M_CURSOR DH,DL
         CALL    Byt2Asc
         MOV     AL,'H'
         CALL    BIOchr1
         POP     AX
         RET
```

- La descrizione del progetto di un **Editor per Visualizzatori LCD** non può aver termine senza aver fatto cenno alle *parti di codice* dedicate al ***Servizio dei tasti di Editing***.

- Il progetto prevede la possibilità di intervenire:
  - sull'organizzazione del programma, con *F1*, per predisporre entrambi i *visualizzatori* (*LCD* e *a video*) per la desiderata risoluzione in *caratteri per riga* (**16**, **20**, **24**, **32** o **40**), e *Esc* per terminare
  - sulla cancellare i testi, con *F2*, per cancellare tutti i caratteri nel *display LCD* (e nella *DDRam* del *controller*) e sul *visualizzatore a video*, e con *Backspace*, per cancellare indietro il carattere corrente, arretrando di un posto a sinistra anche la posizione del cursore
  - sulla posizione del cursore senza scrittura, con *End* (in basso a sinistra), *PgUp* (in alto a destra), *PgDn* (in basso a destra), *Home* (in alto a sinistra), *FrecciaDx* e *FrecciaSn* (in avanti o indietro di una posizione sulla stessa riga), *FrecciaSu* e *FrecciaSu* (in su o in giù sulla stessa colonna nella riga opposta) e con *Invio* (all'inizio della riga opposta a quella corrente)

- Ognuna di queste strutture risulta molto impegnativa, in misura maggiore anche nei confronti del progetto precedente, per la necessità di aggiornare la posizione di stampa corrente in avanti e indietro su 2 righe alternative.

- La più complessa di tutte è quella che si occupa del servizio per il tasto *Backspace*, che sposta la posizione di stampa indietro di una posizione, cancellando il carattere verso sinistra, anche nella *memoria DDRam* del *controller* LCD; il *sorgente scaricabile* offre 2 possibili soluzioni:
  - quella in uso che, alla pressione del tasto, sposta indietro di un carattere la posizione di stampa, *autonomamente* su ciascuna linea, bloccandosi sul primo carattere della rispettiva linea
  - quella alternativa (particolarmente complessa..), che "*cancella indietro*" ***tutto*** il campo visivo, proseguendo sull'altra linea, non appena arriva al primo carattere della linea sulla quale si è iniziato il servizio cui è entrato

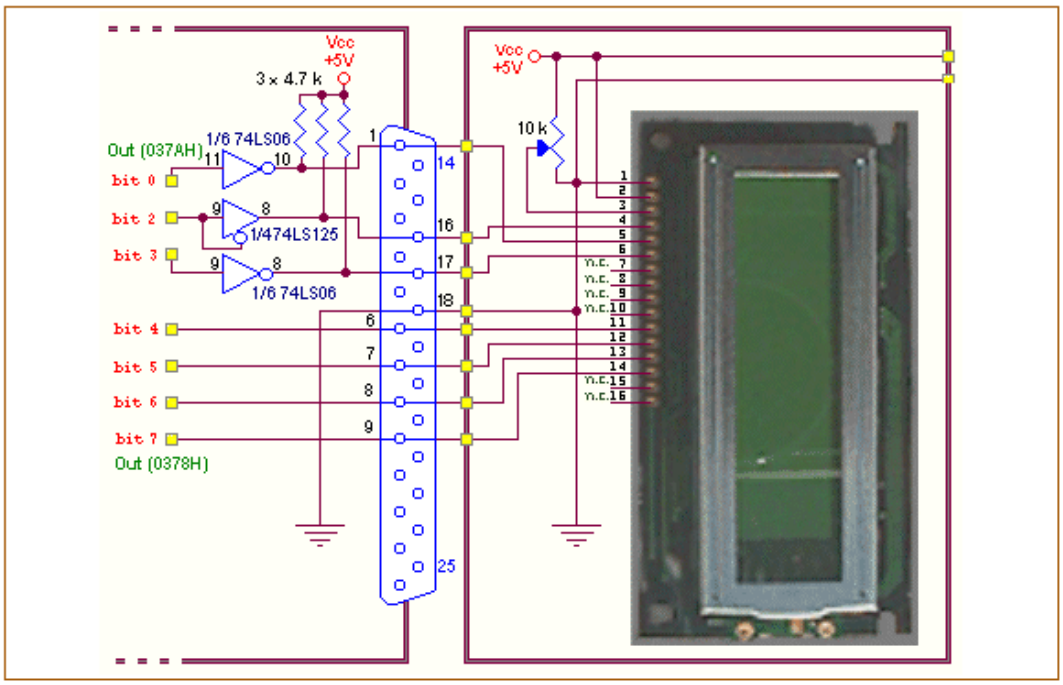
- A titolo d'esempio riporto la sequenza d'istruzioni relativa al servizio, forse, più immediato.., quella per il tasto *Invio*:

```
;┌──────────────────────────────────────────────────────────────────────┐
;|SERVIZIO   : Serv_Invio                                               |
;|DESCRIZIONE: La pressione del tasto "Invio" sposta la posizione di  stampa |
;|             per puntare il carattere all'inizio della seconda riga.  |
;└──────────────────────────────────────────────────────────────────────┘
Serv_Invio:
        CALL    Old_Chr         ;┐Annulla  la  visualizzazione  del  cursore nel
                                ;┘punto previsito per la stampa corrente
        MOV     DL,colnIni      ; Prepara il puntatore Video a capo linea
        MOV    AL,CS:[Address]; Assume il valore corrente dell'indirizzo DDRam
        MOV     DH,rigaIni      ;┐Prepara il puntatore Video sulla seconda linea
        INC     DH              ;┘
        MOV     AH,40H          ;┐Prepara il puntatore DDRam a inizio 2.da linea
        CMP     AL,27H          ;|Se la posizione è sulla prima linea  si  forza
        JBE     Serv_I          ;┘il cursore a capo della seconda linea
                                ;┐Altrimenti (posizione sulla seconda linea)  si
                                ;|forza il cursore a capo della prima linea....
        MOV     DH,rigaIni      ;|Prepara il puntatore Video sulla prima linea
        MOV     AH,00H          ;┘Prepara il puntatore DDRam a inizio 1.ma linea
Serv_I: MOV     AL,AH           ;
                                ;┐Fissa la variabile  [Address]  con  il  valore
                                ;|della posizione corrente del  cursore e  copia
                                ;|l'informazione  nel  registro  puntatore della
                                ;|memoria DDRam del controller LCD, con  valori:
        JMP     Rientra         ;|■da 00H (0,0) a 27H (0,39) sulla  prima  linea
                                ;|■da 40H (1,0) a 67H (1,39) sulla seconda linea
                                ;|Inoltre mostra il cursore sul Display a  Video
                                ;┘e stampa il valore della Locaz. DDRam corrente
```

- L'analisi del codice mette in evidenza:
  - il *recupero del carattere* presente nella posizione che si sta per lasciare (*a capo* della linea di partenza), coperto dal cursore; l'operazione è svolta dalla **SottoProcedura** **Old_Chr**, specializzata nel rimettere le cose a posto a livello RamVideo
  - predispone i *puntatori DDRam* (**00**H o **40**H) e le *coordinate del cursore a video* (**DH** riga e **DL** colonna) esattamente a capo della linea opposta a quella corrente
  - rientra quindi subito nel loop di attesa tasti, con un ***salto*** (`JMP Rientra`, non si tratta di una procedura, quindi ***non si torna*** con **RET**)
  - è importante sottolineare che, prima dell'effettiva attesa di tasti, a livello `Rientra`, viene ribadita la posizione di stampa, a capo linea, sia per il *display LCD*, (fissando l'indirizzo in *DDram* con **FissaAdd**) sia sul *visualizzatore a video* (con **FissaCur**)

```
;┌──────────────────────────────────────────────────────────────────────┐
;|SERVIZIO   : Old_Chr                                                  |
;|DESCRIZIONE: Ripropone il carattere coperto di volta in volta dal Cursore, |
;|             nei casi in cui è necessario, come dopo la  pressione di Invio|
;|             o di BackSpace.                                          |
;└──────────────────────────────────────────────────────────────────────┘
Old_Chr:PUSH    AX
        MOV     AL,CS:[OldChr];┐Assume  dalla  variabile  locale  [OldChr]  il
        CALL    RAMcur         ;|carattere coperto in precedenza dal Cursore
        MOV     ES:[DI],AL     ;┘e lo rimette al suo posto, in RamVideo
        POP     AX
        RET
```

Schema dell'Interfaccia



Descrizione dello schema

- Per seguire il commento sui **dettagli tecnici** può essere conveniente aprire una seconda finestra, con lo **schema** dell'interfaccia.
- Il progetto utilizza un **Modulo LCD**, completamente programmabile attraverso il suo *controller*, nel nostro caso l'**integrato HD44780** della Hitachi, alimentato *esternamente* con un piccolo alimentatore da 5 volt stabilizzati.
- Il progetto prevede, per il controllo del **Modulo LCD**, la programmazione della *porta parallela* standard **SPP** (**S**tandard **P**arallel **P**ort) **LPT1**; poichè non è prevista alcuna lettura sono stati coinvolti solo i 2 **Registri** d'uscita:
  - il **Registro** d'uscita **0378H/0278H**: coinvolge i suoi 4 bit più significativi, dal bit7 al bit4 (disponibili rispettivamente dal pin **9** al pin **6** del connettore **DB-25** della *porta parallela*) per organizzare il *bus dati a* **4** *bit* del **Modulo**
  - il **Registro** d'uscita **037AH/027AH**: coinvolge 3 dei suoi 4 bit per creare da software i 3 *segnali* necessari:
    - il bit**0**, disponibile sul pin **1** della parallela (*strobe* per l'interfaccia *Centronics*), usato per generare il segnale *Read/Write* (**R**/$\overline{\mathbf{W}}$, pin **5** del **Modulo**); deve essere a **0** se il dato è *scritto* e a **1** se è *letto*
    - il bit**2**, disponibile sul pin **16** della parallela (*init* per l'interfaccia *Centronics*), usato per generare il segnale *Register Select* (**RS**, pin **4** del **Modulo**); serve per indicare la natura dell'informazione presente sul *bus dati*: deve essere posto a **0** se il dato rappresenta un *comando da eseguire* (*istruzione*), oppure a **1** se si tratta di *dati* (*caratteri*) da scrivere sul visualizzatore
    - il bit**3**, disponibile sul pin **17** della parallela (*select in* per l'interfaccia *Centronics*), usato per generare il segnale *Enable signal* (**E**, pin **6** del **Modulo**); viene forzato a **1** se il dispositivo è *pronto a gestire* un *dato* o un *comando* predisposto sul *bus dati*
    - va sottolineato che il bit**0** ed il bit**3** sono sottoposti ad un'*inversione logica interna*, prima di essere disponibili sul connettore della porta
- L'osservazione dello schema mette in evidenza anche che:
  - è necessaria una fonte di **alimentazione esterna** (di **+5V**).
  - la **massa** dell'alimentatore deve essere unita con quella della porta parallela: la conoscenza del suo connettore suggerisce la disponibilità di ben 8 pin (dal pin **18** al pin **25**); per questa realizzazione si è scelto il pin **18**.

⚠ La descrizione del programma ASM chiamato a gestire il progetto ha già evidenziato la necessità di **conoscere a fondo** le caratteristiche del *controller* **HD44780**; sebbene la cosa ti sia ormai ampiamente nota, ti ricordo i link che descrivono il suo funzionamento hardware e le sue necessità software.

- La scheda dovrà dunque prevedere un **connettore da 8 contatti**, tutti verso la porta parallela, **più 2** per l'alimentatore.
- Puoi disporre di dettaglio migliore consultando Tabella dei collegamenti tra i *14* pin del **Modulo** e *8* dei pin del connettore DB25 femmina della *porta parallela*.

Display LCD

- **Modulo Visualizzatore - Connettore**
  - Connettore del Modulo - Segnali sul connettore
- **Controller/Driver HD44780 - Hardware**
  - Risorse Interne (1 di 2)
    - ROM del generatore di Caratteri
    - RAM del generatore di Caratteri
  - Risorse Interne (2 di 2)
    - RAM per i dati del Display
    - Registri a 8 bit
    - Flag di Busy
    - Contatore di indirizzo (AC, Address Counter)
  - Diagrammi temporali
    - Fase di scrittura
    - Fase di lettura
  - Conclusioni e Links - Data Sheet
- **Modulo Visualizzatore - Interfacciamento**
  - Interfaccia con la porta parallela LPT1
- **Controller/Driver HD44780 - Software**
  - Descrizione delle Istruzioni (1 di 2)
    - Nop
    - Clear Display
    - Cursore a capo
    - Modo d'accesso dei caratteri
  - Descrizione delle Istruzioni (2 di 2)
    - Controllo Display
    - Scorrimento Cursore/Testo
    - Predisposizione dell'indirizzamento della CGRam
    - Predisposizione dell'indirizzamento della DDRam
  - Programmazione - Fasi iniziali
    - Inizializzazione del componente
  - Programmazione - Scelte gestionali
    - sequenza finale per interfaccia a 8 bit
    - sequenza finale per interfaccia a 4 bit
  - Programmazione - Erogazione Caratteri
- **Progetti Hardware**
  - Gestione Modulo LCD: messaggio 2 x 16 (con controller HD44780)
  - Gestione Modulo LCD: editor 2x16 (con HD44780)
  - Gestione Modulo LCD: editor 2x16 (variante) (con HD44780)
  - Gestione Modulo LCD: editor avanzato 2x16,20,24,32,40 (con HD44780)