Interfacciamento dei processori

Quarta Parte: Le periferiche di visualizzazione a 8 bit [2 di 3]

PREMESSA

Nella **prima parte** di questa serie ci siamo occupati delle **interfacce d'uscita per un dato a 8 bit**, e abbiamo concluso che qualunque dispositivo programmabile (*microcontrollore* o *personal computer*) dispone ***intrinsecamente*** di almeno **almeno una** porta in grado di assicurare questo servizio.

Nella **seconda parte** abbiamo studiato le **caratteristiche elettriche tipiche di una porta d'uscita** , analizzando le specifiche di quella **TTL** compatibili (per altro le più diffuse) e proponendo una tecnica d'*analisi elettronica* facilmente generalizzabile anche per eventuali altre tecnologie. Abbiamo affrontato il concetto di **carico**, e proposto i primi due schemi d'interfaccia, uno con l'utilizzo dei ***buffer di corrente non* invertenti 74LS244** e uno con l'impiego delle **memorie esterne a 8 bit 74LS374** (*batterie* di 8 flip-flop **D-Type** *in parallelo*).

Nella **terza puntata** abbiamo affrontato la necessità di disporre di **periferiche adatte ad esprimere l'infomazione**, presentando la **batteria di 8 led** e i **digit a sette segmenti** nelle sue due forme funzionali alternative (a *catodo* e a *anodo comune*); per entrambe è stata introdotta la **Tabella di associazione logica**, una struttura indispensabile per facilitare la creazione del codice di gestione.

PERIFERICHE DI VISUALIZZAZIONE A 2 DIGIT A 7 SEGMENTI

Abbiamo analizzato in dettaglio le *condizioni elettriche* a cui è sottoposta la porta d'uscita di un *microcontrollore* o di un *PC* quando ci colleghiamo ***direttamente*** un **digit**, senza *strato* di **interfaccia**; in sintesi questa soluzione gode di una ***esclusiva*** interessante:

- consentire il ***controllo assoluto*** di tutti gli **8 led** in esso contenuti, assicurando la visualizzazione di *caratteri originali* (particolari *combinazioni di segmenti* non *alfanumeriche*, non previsti dal codice Ascii) molto utili per fornire segnalazioni come *allarmi* o *segnali di attesa*
- consentire di *spegnere* il **digit** in modo diretto, senza bisogno di hardware
- permettere la gestione del *Decimal Point*, in presenza o in assenza di uno o più *segmenti*, al fine di creare ulteriori possibilità di visualizzazione

Per contro, per ogni **digit** è necessario *sprecare* una porta e, spesso, questo fatto non è conveniente, anche perché quello che si realizza in questo modo è un visualizzatore *atipico*: nell'uso comune i **visualizzatori a sette segmenti** servono da interfaccia numerica per tradurre un'informazione *binaria* (elaborata dal processore) in una simbolica, basata sul sistema di numerazione *decimale*, adatta alla sensibilità umana; tipiche applicazioni sono i *registratori di cassa* o le normali *calcolatrici*.

Ma vale la pena di pensare **anche** ad una possibilità molto ambita dai progettisti di sistemi programmabili: monitorare in tempo reale lo ***stato dell'hardware*** su cui si sta lavorando; il sogno di ogni appassionato è quello di crearsi un piccolo **sistema di sviluppo** in grado di mostrare *passo passo* l'effetto delle istruzioni, per esempio sullo stato dei *registri della CPU* o sul *contenuto* di una determinata *locazione della memoria* del sistema, magari mostrandone anche l'*indirizzo*; in questo caso il compito del **visualizzatore a sette segmenti** è quello di tradurre l'informazione *binaria* in una basata sul sistema di numerazione *esadecimale*, coinvolgendo oltre ai *caratteri numerici* anche quelli associati alle *prime sei lettere*.

Nota: non ti farebbe male *staccare un attimo* e passare alla veloce lettura della scheda **Misura dell'informazione**, per fissare le idee: sono certo che sono cose che già sai, ma una rinfrescatina non fa mai male..

GESTIONE DI 2 DIGIT CON I DECODER A SETTE SEGMENTI

Premessa

Per raggiungere entrambi gli obiettivi l'utilizzo di **digit** collegati ***direttamente*** alle porte d'uscita non è più conveniente!

Il metodo più efficace è quello di cercare di interpretare in modo ***naturale*** il *codice binario* fornito dalla logica programmabile (*micro* o *PC*) sulla porta d'uscita, cioè di affidare ad ogni sequenza di bit il suo *naturale valore* ***numerico***; certamente sei consapevole che una *sequenza di 8 bit* (cioè un *byte*) esprime un **numero** che, *in decimale*, può assumere un valore da **0** [=$(00000000)_2$] a **255** [=$(11111111)_2$] (.. se la cosa *non ti tornasse* vale il consiglio di poco fa: passa alla veloce lettura della scheda).

Per capirci: nel caso discusso in precedenza il *codice binario* fornito sulla porta d'uscita, doveva per forza essere una sequenza adatta ad accendere o spegnere i segmenti del **digit**, quindi tutt'altro che interpretabile in modo aritmetico, ma stabilita a partire dalle indicazioni della relativa **tabella di associazione logica**.

La volontà di fornire in uscita una *sequenza binaria* con significato ***numerico*** suggerisce però alcune argute considerazioni:

- in primo luogo *per esprimere un* ***numero*** (o meglio *uno dei* ***10*** *elementi* del sistema di numerazione decimale, vedi la scheda consigliata) sono sufficienti *4 bit* (cioè un *nibble*); resta inteso peraltro che le possibili combinazioni di *4 bit* portano alla definizione di ***16*** *diverse parole* appartenenti a quello che è bene chiamare subito ***Codice Binario Puro a 4 bit***: in buona sostanza, ***6 in più*** rispetto a quelle strettamente necessarie
- in ogni caso la *sequenza di 8 bit* proponibile sulla porta d'uscita potrà essere intesa come due gruppi di *4 bit* ciascuno, il *nibble alto* e *nibble basso* del *byte* disponibile; è dunque in grado di organizzare il servizio per **due** digit
- di certo, come è facilmente intuibile, nessuno dei due *nibble* potrà essere fornito direttamente ai due **digit**

Da tempo immemorabile questo problema è stato affrontato e risolto dagli studiosi di *logica digitale*: si tratta di progettare una ***macchina combinatoria*** in grado di accettare in ingresso un *codice binario a 4 bit* e

rendere disponibili in uscita *sette linee logiche* adatte a controllare lo stato (*acceso* o *spento*) di altrettanti led.

Questa macchina è un **decoder**, cioè un circuito logico in grado di interpretare un **codice** e tradurre (*decod*ifica*r*e) ogni sua **parola** in una azione specifica.

I Codici

Le considerazioni precedenti ci consentono ora di approfondire il concetto di **codice** e di aprire una piccola parentesi; i **Codici** sono particolari *convenzioni* tramite le quali è possibile associare determinate *sequenze di bit* ai simboli di uso comune; tra essi:

- i ***Codici Alfanumerici***, inventati per consentire alle macchine digitali (come il processore) di gestire oggetti umani come i *caratteri*, da leggere sulle tastiere o da mettere sui visualizzatori (non solo monitor..); il più diffuso tra questi è il ***Codice Ascii***, chiamato ad associare una codifica binaria ad ogni simbolo conosciuto
- altrettanto diffusi sono però anche i ***Codici Numerici***, di solito utilizzati per minimizzare gli errori nella *gestione dei dati* (come nel caso dei ***Codici* Aiken**, **Eccesso a 3** o **Gray**), ma anche per favorirne la visualizzazione (come nel caso del **Codice BCD**); di norma questi **Codici** sono formati da *10 parole* ciascuna delle quali utilizza *4 bit* (un *nibble*) per codificare ciascuno dei *10 simboli* del sistema di numerazione decimale.

Per finire, è *buona cosa* abituarsi a trattare **anche** qualunque altra sequenza di bit come appartenente ad un ***Codice Binario Puro***:

- le 4 combinazioni possibili con *2 bit* esprimono un **Codice Binario Puro a 2 bit**

Binario	Decimale
000	0
001	1
010	2
011	3

- le 8 combinazioni possibili con *3 bit* esprimono un **Codice Binario Puro a 3 bit**

Binario	Decimale
000	0
001	1
010	2
011	3
100	4
101	5
110	6
111	7

- le 16 combinazioni possibili con *4 bit* esprimono un **Codice Binario Puro a 4 bit**

Binario	Decimale
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	8
1001	9
1010	10
1011	11
1100	12
1101	13
1110	14
1111	15

- il gioco delle definizioni può continuare all'infinito; molto frequentemente i ***Codici Binari Puri*** si trovano associati a componenti digitali di uso comune come *contatori* o *decoder binari* o

altro..; va sottolineato che i **Codici** più noti sono sempre un ***sottoinsieme*** di quelli ***Binari Puri***

Per la sua grande importanza (e perché sarà oggetto delle nostre analisi future) conviene sviluppare la conoscenza del **Codice BCD** (**B**inary **C**oded **D**ecimal, *codifica binaria dei simboli decimali*) formato, come gli altri, da *10 parole* ciascuna delle quali utilizza *4 bit* (un *nibble*) per codificare ciascuno dei *10 simboli* del sistema di numerazione decimale:

BCD	Binario	Decimale
0000	0000	0
0001	0001	1
0010	0010	2
0011	0011	3
0100	0100	4
0101	0101	5
0110	0110	6
0111	0111	7
1000	1000	8
1001	1001	9

Il **Codice BCD** *sembra* apparentemente uguale al ***Codice Binario Puro* a 4 bit**, ma non è la stessa cosa! La differenza sta nel fatto che *il primo* utilizza **solo** le prime 10 quaterne di bit *del secondo*; si tratta per questa ragione di un codice ***ridondante***, che **non ammette** tra le sue parole le possibili rimanenti **6 combinazioni**. Inoltre è un codice ***pesato***, secondo la sequenza classica **8421** di pesi, da sinistra a destra:

- il ***peso*** è il valore decimale di ciascuna cifra del numero: quella *più significativa* vale **8**, quella *meno significativa* vale **1**, quelle *intermedie* valgono rispettivamente **4** e **2**
- moltiplicando il valore di ciascuna cifra per il proprio ***peso*** si ottiene il valore decimale del numero binario; per esempio, il numero $(1001)_2$ vale $(1x\mathbf{8}+0x\mathbf{4}+0x\mathbf{2}+1x\mathbf{1})_{10}=(\mathbf{9})_{10}$

Per questa caratteristica il **Codice BCD** è noto anche con il nome di **Codice 8421**.

Poiché le porte d'uscita di un *microcontrollore* o di un *PC* sono a *8 bit* è importante capire fin d'ora che gli 8 bit di un ***Codice Binario Puro* a 8 bit** non sono la stessa cosa degli *8 bit* espressi con le parole di un ***Codice Numerico***; questo vale a maggior ragione per il **Codice BCD**, data la sua apparente *somiglianza* con il ***Codice Binario Puro* a 4 bit**:

- con 8 bit di ***Codice Binario Puro* a 8 bit** possiamo rappresentare tutti i numeri decimali da **0** a **255**
- con 8 bit di **Codice BCD** possiamo rappresentare tutti i numeri decimali da **0** a **99**

Abbiamo imparato che l'informazione binaria, così com'è, è difficile *da intendere* in modo decimale; la scrittura di un'informazione **in BCD** tende a superare questa difficoltà, imponendo di interpretare un numero a 8 bit non come *sequenza di 8 bit* (*byte*) ma come *sequenza di due quaterne di bit* (due *nibbles*) ciascuna delle quali di valore mai superiore a 1001 cioè tipicamente associata alle possibili cifre decimali, come in questo esempio:

$$(10010011)_{BCD} = (93)_{10}$$

E' facile verificare che le *2 parole* **BCD** $(1001)_{BCD}$ e $(0011)_{BCD}$ corrispondono rispettivamente ai simboli decimali $(9)_{10}$ e $(3)_{10}$; questa possibilità ne giustifica la fama e la diffusa applicazione, specialmente nella codifica delle informazioni da porre sui visualizzatori, come vedremo tra poco.

A conferma di quanto detto possiamo osservare come lo stesso numero inteso come ***binario puro* *a 8 bit*** corrisponda a tutt'altro numero decimale:

$$(10010011)_{2} = (147)_{10}$$

Questo esempio mostra, nella *codifica* di un numero decimale, anche scarsa compattezza delle espressioni **BCD** rispetto a quelle ***binarie pure*** e, al tempo stesso, la necessità di un numero di bit decisamente superiore; per rimanere nell'esempio precedente notiamo che per esprimere **in BCD** il numero decimale **147** servono ben ***12* *bit*** (che, in realtà, *diventano* ***16*** se, come in questo caso, il numero decimale ha un numero dispari di cifre: per *formattare il dato* binario alla unità di misura minima (il byte) è necessario aggiungere davanti altri 4 bit a 0):

$$(0000000101000111)_{BCD\ packed} = (147)_{10}$$

Nota: anche in questa occasione ti consiglio, per fissare le idee, di *staccare un altro attimo* e passare alla veloce lettura della scheda **Numeri e Logica Binaria**..

I componenti

Riprendiamo il discorso sui **decoder**, necessari per pilotare l'accensione dei segmenti di un **digit**; è ora facile ipotizzare che il numero binario proposto sulle sue 4 linee d'ingresso possa essere interpretato (*decodificato*) *in parte* (solo le prime 10 combinazioni, **Codice BCD**) oppure *nella sua interezza* (tutte le 16 combinazioni, ***Codice Binario Puro a 4 bit***):

- nel primo caso si parla di ***BCD to 7-segment* Decoder**: gli integrati disponibili sul mercato sono (quasi) esclusivamente di questo tipo e non c'è da stupirsi, dato che saranno chiamati a formare **visualizzatori** destinati a mostrare informazioni (numeri..) decimali, come *registratori di cassa*, *bilance*, *strumenti* di ogni tipo,...
- alla seconda categoria appartiene solo il **9368** (nonostante abbia fatto numerose ricerche non ne ho trovati altri..) e, sebbene il suo *data sheet* non preveda per esso una definizione specifica, lo definirei ***Binary to 7-segment* Decoder**, in contrapposizione con quelli della precedente categoria; si tratta di un componente magico, indispensabile per portare a termine il nostro ambizioso progetto di creare un sistema in grado di visualizzare (per esempio..) il *contenuto* dei *registri della CPU* o di una *locazione della memoria*; l'unico cioè in grado di mostrare tutti i *16 simboli* del sistema di numerazione esadecimale, interpretando e decodificando tutte le possibili combinazioni di *4 bit* proponibili al suo ingresso

Curiosamente tutti i **Decoder** descritti in seguito sono (parzialmente) *pin-out compatibili* tra loro:

- condividono le linee d'ingresso [**D** (pin**6**), **C** (pin**2**), **B** (pin**1**), **A** (pin**7**), in ordine di peso decrescente] sulle quali accettano una *sequenza di 4 bit* (***binaria pura*** o **BCD**)
- condividono le linee d'uscita per i segmenti [identificati da: **a** (pin**13**), **b** (pin**12**), **c** (pin**11**), **d** (pin**10**), **e** (pin**9**), **f** (pin**15**) e **g** (pin**14**)] sulle quali assicurano i livelli logici adatti per accendere il **digit** da esso controllato

- utilizzano il pin**16** per la *tensione positiva* d'alimentazione e il pin**8** per la *massa* (con eccezione del 74LS49, per altro poco significativo)
- affidano ai ***tre*** *piedini* rimanenti (pin**3**, pin**4** e pin**5**) compiti *di* ***controllo*** importanti e utili, talvolta *funzionalmente identici* tra loro e comunque tali da differenziarne anche sostanzialmente le caratteristiche del servizio; per poter capire il loro compito è necessario attendere la descrizione del contesto in cui saranno operativi, per cui saranno esaminati con dettaglio solo al termine dello studio dei singoli componenti

Decoder da *BCD* a *sette segmenti*

Va subito evidenziato che queste *macchine combinatorie* sono predisposte con 4 linee d'ingresso, sulle quali è possibile imporre tutte le possibili ***16*** *combinazioni* a *4 bit*, ma non sono progettate per interpretare le ***6*** *combinazioni* più significative, da $(1010)_2$ a $(1111)_2 = (10)_{10}$ a $(15)_{10}$; sta al programmatore *evitare con cura* che questo evento accada, altrimenti il **digit** ad esse collegato fornirà simboli improbabili, di norma inaccettabili.

I valori binari accettabili sono dunque quelli da $(0000)_2$ a $(1001)_2$ che, come abbiamo visto, *non solo* sono le prime *dieci sequenze* ***Binarie Pure*** **a 4 bit** ma anche le *dieci parole* del **Codice BCD**; per ciascuna di esse il componente attiverà in uscita i segmenti necessari per creare i corrispondenti *10 simboli* del sistema di numerazione decimale, da $(0)_{10}$ a $(9)_{10}$, esattamente quelli che verranno proposti sul **digit** controllato dal **Decoder**.

Per questo servizio sono disponibili i componenti **TTL** **74LS47** e **74LS48** (più **7446** e **74LS49**, meno adatti per la gestione diretta di **digit**) tra loro *funzionalmente identici* e *pin-out compatibili*, ma governati da tecnologie d'uscita diverse, e il componente **CMOS** **4511**, esso pure in parte *pin-out compatibile* (escludendo tre piedini di controllo) con i precedenti ma *funzionalmente* molto più sofisticato.

Decoder da *Binario puro a 4* bit a *sette segmenti*

Per questo servizio è disponibile un solo componente: il **9368**, anch'esso in parte *pin-out compatibile* (sempre escludendo tre piedini di controllo) con i precedenti **Decoder** e *funzionalmente* simile al **CMOS 4511**.

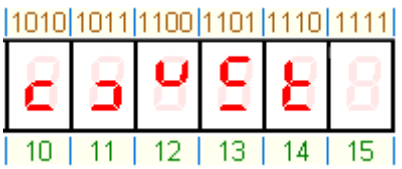
In ingresso accetta ora tutti i possibili valori binari, da $(0000)_2$ a $(1111)_2$, cioè *tutte* le *sequenze* ***Binarie Pure* a 4 bit**; per ciascuna di esse il componente attiverà in uscita i segmenti necessari per creare i corrispondenti *16 simboli* del sistema di numerazione esadecimale, da $(0)_{10}=(0)_H$ a $(15)_{10}=(F)_H$, proposti poi sul **digit** controllato dal **Decoder**.

Purtroppo è **l'unico** componente in grado di organizzare la visualizzazione dei *simboli* ***esadecimali***, indispensabili per tradurre in modo *umano* le informazioni *binarie pure* presenti *naturalmente* sulle porte d'uscita di un *micro* o di un *PC*, tipiche dell'ambiente del quale esse sono la parte terminale, come gli *indirizzi* (delle *locazioni di memoria*, visualizzabili su **4 digit** a partire dai loro *16 bit*) o *dati* (per esempio il *contenuto* di *registri* e/o *locazioni*, visualizzabili su **2 digit** a partire dai loro *8bit*)

DECODER DA BCD A SETTE SEGMENTI CON TECNOLOGIA TTL

Generalità

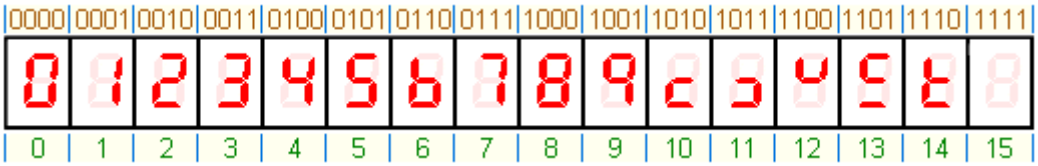
Tutti e quattro i componenti citati trattano *in modo particolare* le possibili ***6*** *combinazioni* a *4 bit* non appartenenti al **codice BCD**[da $(1010)_2=(10)_{10}$ a $(1111)_2=(15)_{10}$, eventualmente proposte sulle linee d'ingresso] *attivando* le linee d'uscita di segmento per formare i seguenti simboli:



La disponibilità di *questi simboli*, piuttosto di altri, ha una logica: chi si è cimentato nel progetto di una macchina combinatoria di questo tipo a partire dalla sua *tabella di verità* è a conoscenza delle cosiddette condizioni di indifferenza; in breve (sarebbe divertente poter andare più a fondo, ma non è questo l'ambito giusto...) partendo dal **presupposto** che le *ultime 6 combinazioni* non devono mai essere fornite (perché non appartenenti al **codice BCD**), il valore logico che la rispettiva uscita può assumere è ***indifferente***, cioè può essere assunto a piacere uguale a "**1**" o a "**0**", nel modo più conveniente ai fini del progetto stesso.
Se, **nonostante il divieto**, si fornisce (a progetto finito) una delle *ultime 6 combinazioni vietate*, l'aspetto delle uscite è dunque legato alle scelte imposte alle relative condizioni di indifferenza e si traduce (nel nostro caso) in quello mostrato dalla precedente figura.

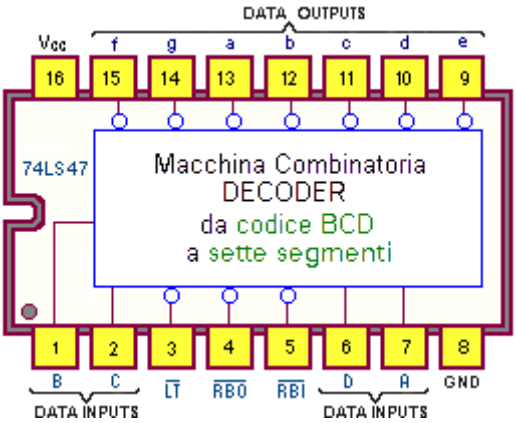
Il caso ha voluto che due dei simboli ottenuti siano *realistici* [una "**c**" per $(1010)_2$ e una "**t**" per $(1110)_2$], ma soprattutto l'ultimo [mostrato con ingressi a $(1111)_2$] si presta ad una interessante considerazione: se il programmatore impone in ingresso il codice binario (non **BCD**) 1111 otterrà come effetto quello di **spegnere** il **digit** controllato dal **Decoder**; vedremo tra un po' che questo effetto, decisamente utile, è gestibile anche da hardware ma la possibilità di ottenerlo direttamente da software ha il grande vantaggio di non costare nulla!

Prima di entrare nel merito dei singoli componenti mostriamo l'aspetto delle uscite per ognuna delle ***16*** *combinazioni* possibili; da notare le scelte imposte per la visualizzazione del "**6**" $(0110)_2$ e del "**9**" $(1001)_2$, entrambi privi del trattino esterno, 6 e 9, previsto invece da altri decoder:

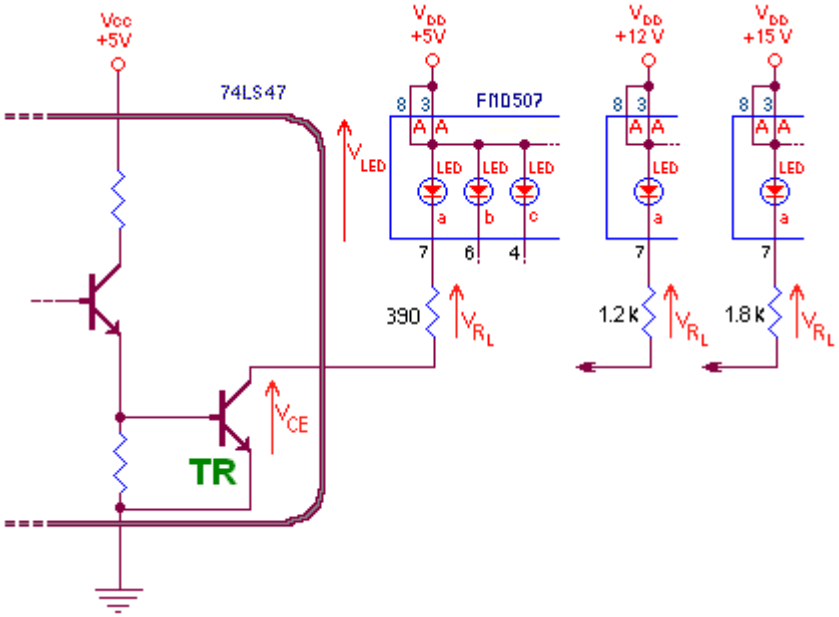


Decoder TTL: 74LS47

Il **Decoder TTL 74LS47** converte il **Codice BCD** proposto sulle sue linee d'ingresso generando sulle sue linee d'uscita delle sequenze *attive basse*: i segmenti del **digit** ad esso collegato si accenderanno dunque con uno "**0**" logico per cui questo componente è adatto a **digit** ad **anodo comune**; il suo **pin-out** è illustrato dal seguente schema:



Ciascuna delle sue uscite è dotata di un **driver** di tipo ***open-collector*** cioè si avvale di una tecnica *diversa* da quella *Totem-pole* trattata in precedenza; lo *stadio finale* del componente logico è ora un semplice transistor con *emettitore* collegato regolarmente a massa e *collettore* collegato direttamente all'uscita, senza nessun vincolo con la *sua* alimentazione interna.



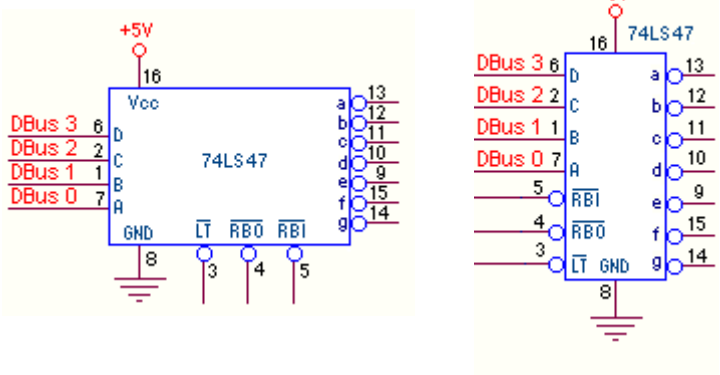
Il fatto che il **collettore** del transistor **TR** sia **aperto** (cioè non collegato a nulla, *fluttuante*) ci obbliga a collegare un **carico** esterno in grado di *chiudere il circuito d'uscita* verso una *tensione positiva*, al fine di consentire il funzionamento del transistor stesso; di solito si utilizza un **resistore**, detto di ***pull-up*** per significare (nel pittoresco modo di esprimersi degli anglosassoni) che il potenziale dell'uscita va "***tirato su***" verso l'alimentazione esterna.

Il vantaggio di disporre di uscite ***open-collector*** garantisce la possibilità di ***assorbire*** correnti piuttosto elevate e di alimentare il **carico** con tensioni *diverse* (di norma *maggiori*) dalla tipica alimentazione a +5V; questo consente di pilotare **direttamente** dispositivi esigenti (per esempio un *relè*), cosa assolutamente preclusa alle classiche uscite *Totem-pole*.

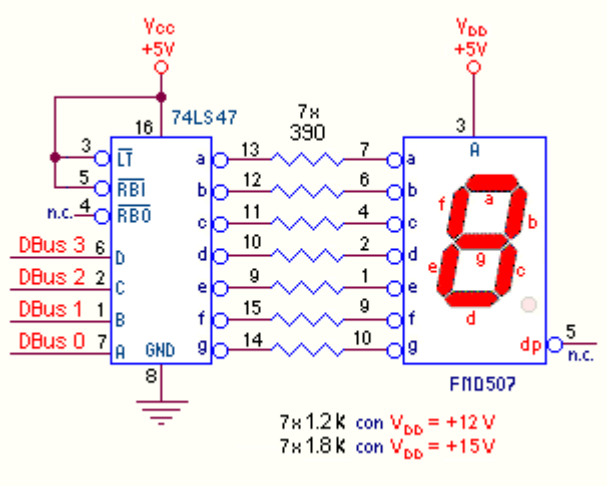
La figura mostra una *linea d'uscita* del **Decoder 74LS47** dotata di regolare **carico** (costituto da un **Led** e dal suo **resistore**) in grado di chiudere il circuito attraverso il suo transistor **TR**); sulla base di queste premesse, essa:

- *a livello basso* (**0** *logico*) garantisce una *corrente* ***assorbita*** di 24mA, attraverso il **carico** e in virtù del transistor **TR**, ora in condizioni di *forte conduzione*; la sua *tensione d'uscita* è comunque quasi nulla (coincide con la tensione V_{CE} di *saturazione* di **TR**, mai superiore a 0,3V)
- *a livello alto* (**1** *logico*) la sua *tensione d'uscita* può raggiungere i +15V, in sostanza il valore massimo ammesso per l'alimentazione esterna; è infatti facile pensare che, in queste condizioni, il transistor **TR** *non conduce* (*in interdizione*) per cui la *caduta di tensione* sul **carico** è nulla (non essendo possibile passaggio di corrente attraverso di esso) e quindi tutta la *tensione di alimentazione* si ritrova ai capi del transistor

Lo **schema funzionale** (proposto in due versioni) riassume le caratteristiche logiche di questo componente; le linee d'*ingresso* possono essere quelle del *Bus Dati* di un microprocessore o di *una porta* di un single-chip o della *porta parallela* di un PC, mentre le linee d'*uscita* rendono disponibili i sette valori logici *attivi bassi* necessari per controllare i *7 segmenti* di un **digit** ad **anodo comune**:



Nel normale funzionamento, i tre *piedini di controllo*, **LT** (pin**3**), **RBO** (pin**4**) e **RBI** (pin**5**), possono essere *inattivi*, lasciandoli *scollegati* o meglio collegandoli a +5V; per capirne il funzionamento dovremo pazientare ancora un po' ...



I 7 **resistori** presenti nel circuito hanno lo scopo di limitare la *corrente* che, dall'alimentatore esterno attraverso l'integrato, percorre ciascuno dei *segmenti-led* del **digit**, quando le uscite sono *attive* (*basse*), al fine di garantirne l'incolumità; il valore della loro *resistenza* è calcolato per una *corrente* di 8mA in presenza di tensioni d'alimentazione di +5V, +12V e +15V, e normalizzato ad un vicino valore standard, rispettivamente pari a 390, 1,2k e 1,8k ohm.

Può essere, infine, istruttivo consultare le pagine dei ***data sheet*** *originali*; una copia è disponibile qui: **74LS47** [Fairchild].

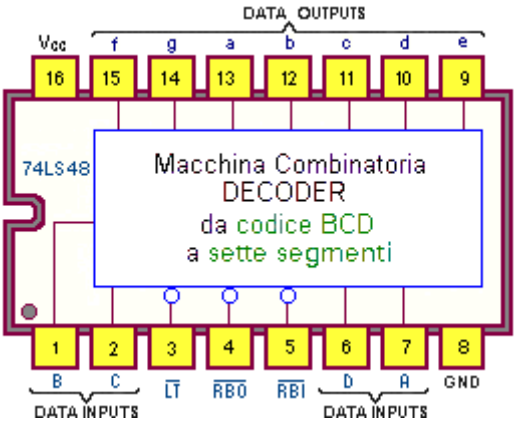
Decoder TTL: 7446

Il **Decoder TTL 7446** è una versione (probabilmente obsoleta) più potente del **74LS47** capace di ***assorbire*** *corrente* fino a 40mA, e di sopportare *tensioni di alimentazione esterna* fino a +30V; fatte salve queste differenze, per esso valgono tutte le considerazioni precedenti.

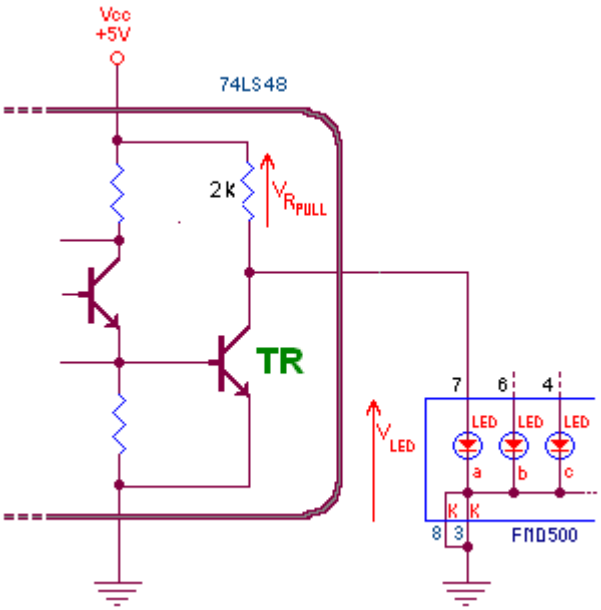
Una copia del suo ***data sheet*** *originale* è disponibile qui: **7446** [National].

Decoder TTL: 74LS48

Il **Decoder TTL 74LS48** converte il **Codice BCD** proposto sulle sue linee d'ingresso generando sulle sue linee d'*uscita* delle sequenze *attive alte*: i segmenti del **digit** ad esso collegato si accenderanno dunque con uno "**1**" logico per cui questo componente è adatto a **digit** a **catodo comune**; il suo **pin-out** è illustrato dal seguente schema:



Ciascuna delle sue uscite è caratterizzata, rispetto alla normale *Totem-pole*, dalla presenza di un **resistore** interno di ***pull-up***, il cui valore di *resistenza* è pari a **2**k; lo scopo di questa originale scelta funzionale è quello di assicurare *maggiore corrente* ***erogata***, in sintonia con le nostre precedenti osservazioni sul fatto che le *uscite* **TTL** non sono (nella norma) adatte a questo servizio.

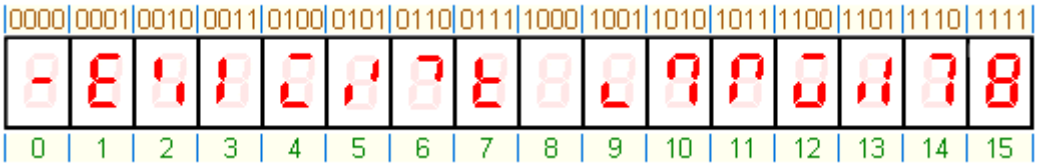


Tuttavia, pur in presenza di un ***pull-up*** interno di **2**k, il **Decoder 74LS48** non è in grado di pilotare direttamente un normale **digit**: questo componente è stato progettato per "*vedere*" una logica TTL o DTL, o per pilotare un *amplificatore* di corrente (**driver**) ad esso *successivo* (per esempio un **ULN2003**) per il controllo di **lampade**, piuttosto che per quello di **digit**; alcune copie del ***data sheet*** *originale* del **driver ULN2003** (contenente 7 coppie di transistor NPN in configurazione Darlington) sono disponibili qui: **ULN2003** [Allegro], **ULN2003** [Silan], **ULN2003** [Sprague], **ULN2003** [ST].

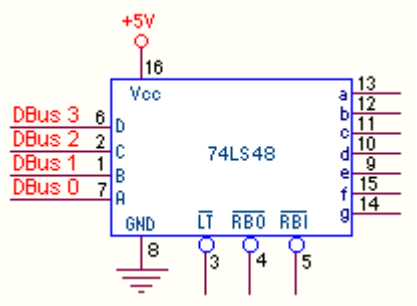
Con queste premesse possiamo puntualizzare che:

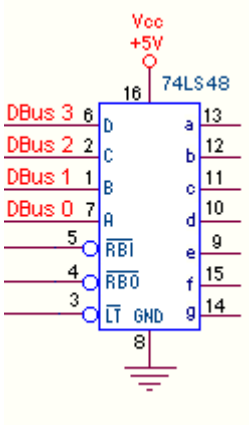
- *a livello alto* (**1** *logico*) può ***erogare*** (secondo il *data sheet*) al massimo 2mA; se il transistor **TR** *non conduce* (*interdetto*) la *caduta di tensione* sul ***pull-up*** interno di **2**k, pari a (5V-1,8V) = 3,2V, impone il passaggio di 1,6mA, una *corrente* decisamente insufficiente per i *segmenti-led* di un normale **digit** a **catodo comune**
- *a livello basso* (**0** *logico*) è garantita (sempre secondo il *data sheet*) una *corrente* ***assorbita*** di 6mA, attraverso il transistor **TR**, ora *in saturazione*: d'istinto *sembrerebbe più logico* l'impiego con **digit** ad **anodo comune** (naturalmente corredato da adeguati **resistori** in serie a ciascun *segmento* per limitare la *corrente* al valore di 6mA, di *resistenza* 470ohm con alimentazione esterna di +5V), ma va tenuto presente che le sequenze logiche d'uscita sono progettate per dare

effetto in *logica positiva*, per cui (essendo *attivo*, in questo caso, lo **0** *logico*) l'effetto sul digit sarebbe decisamente inaccettabile:

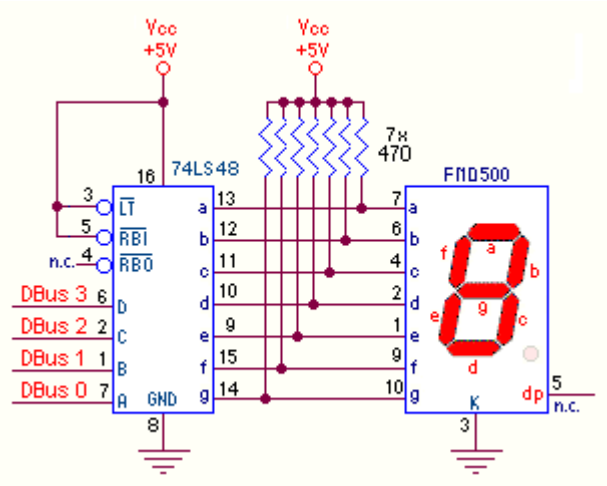


Lo **schema funzionale** (proposto in due versioni) riassume le caratteristiche logiche di questo componente; le linee d'*ingresso* possono essere quelle del *Bus Dati* di un microprocessore o di *una porta* di un single-chip o della *porta parallela* di un PC, mentre le linee d'*uscita* rendono disponibili i sette valori logici *attivi alti* necessari per controllare i *7 segmenti* di un **digit** a **catodo comune**:





L'impiego di questo **Decoder 74LS48** con un normale **digit** a **catodo comune** è dunque possibile **solo** con l'impiego di **resistori** addizionali esterni di ***pull-up*** su ciascuna delle 7 linee d'uscita; la scelta del valore di *resistenza* è legata all'intensità luminosa desiderata per i *segmenti-led*; va da se che che non è il caso di scendere a valori troppo piccoli per non rischiare di bruciare le uscite del componente.

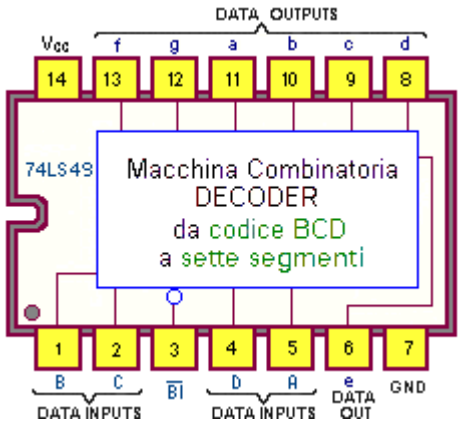


Nel normale funzionamento, i tre *piedini di controllo*, **LT** (pin**3**), **RBO** (pin**4**) e **RBI** (pin**5**), possono essere *inattivi*, lasciandoli *scollegati* o meglio collegandoli a +5V.

Consultare le pagine dei ***data sheet*** *originali* è sempre cosa istruttiva: un paio sono disponibili qui: **74LS48** [National], **74LS48** [Motorola].

Decoder TTL: 74LS49

Il **Decoder TTL 74LS49** è una versione più ***compatta*** e ***complementare*** del **74LS47**, essendo ora *attive alte* le sue linee d'uscita; il **Codice BCD** proposto in ingresso genera dunque delle sequenze adatte ad accendere con uno "**1**" logico i segmenti del **digit** ad esso collegato, per questo di tipo a **catodo comune**; il suo **pin-out** è illustrato dal seguente schema:



Le sue uscite sono di tipo ***open-collector***, cioè il suo *stadio finale* è caratterizzato dalla presenza di un transistor con *emettitore* collegato a massa e con ***collettore*** lasciato ***aperto*** in uscita (cioè non collegato a nulla); anche questo **Decoder** non è adatto alla gestione diretta di un normale **digit**, essendo stato progettato per pilotare una logica TTL o per essere anteposto ad un **driver** per il controllo di **lampade**.

Secondo il *data sheet* *a livello alto* (**1** *logico*) può ***erogare*** al massimo 0,250mA, una *corrente* decisamente insufficiente per i *segmenti-led* di un normale **digit** a **catodo comune**; *a livello basso* (**0** *logico*) è garantita una *corrente* ***assorbita*** di 8mA, ma il dato sembra irrilevante nei confronti del servizio atteso da questo componente.

Nel normale funzionamento, l'unico *piedino di controllo*, **BI** (pin**3**), può essere *inattivo*, lasciandolo *scollegato* o meglio collegandoli a +5V; anche per questo componente un paio di ***data sheet*** *originali*: **74LS49** [Hitachi], **74LS49** [Texas].

DECODER TTL: I SEGNALI DI CONTROLLO

Generalità

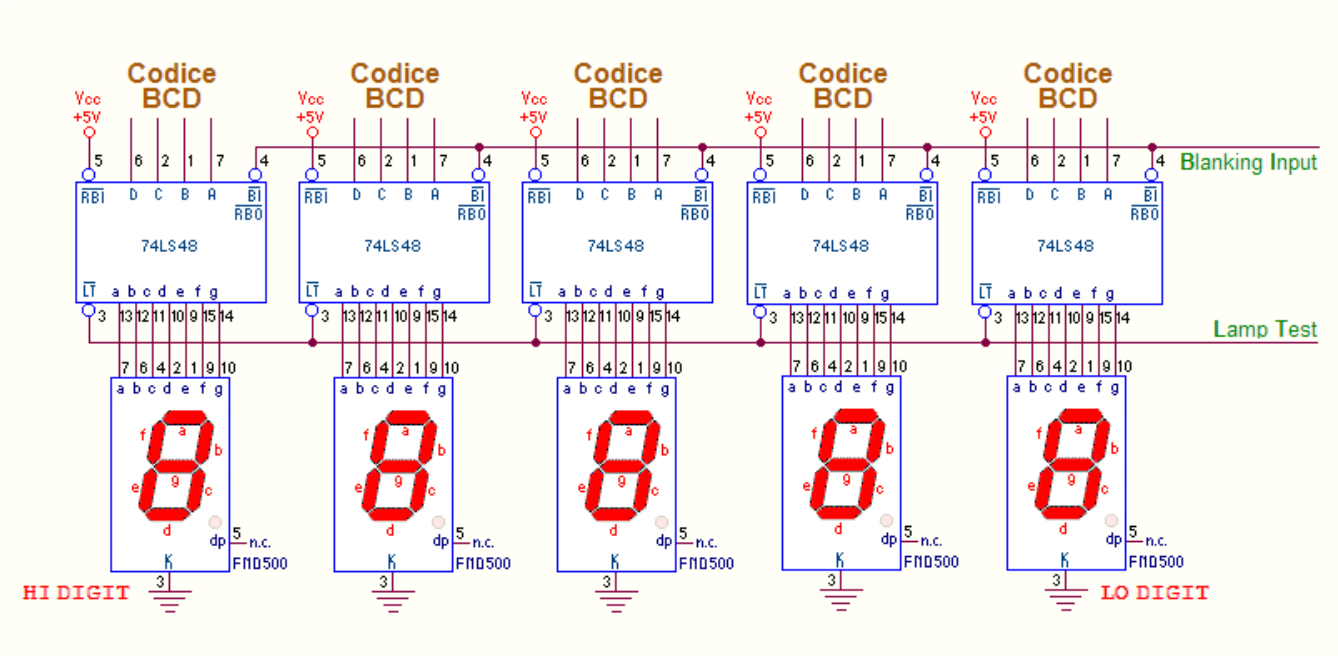
Tutti i **Decoder** analizzati finora utilizzano tre piedini per assicurare ***servizi*** piuttosto particolari, poco interessanti dal punto di vista della loro gestione mediante *porte d'uscita* dei dispositivi programmabili (*micro* o *PC*), ma molto utili nell'ambito dei visualizzatori a *più* **digit**, tipici dei comuni dispositivi presenti sul mercato.

Le linee *di controllo*, tutte e tre *attive basse*. sono note come **LT** (***L****amp* ***T****est*, pin**3**), **BI** / **RBO** (***B****lanking* ***I****nput* / ***R****ipple* ***B****lanking* ***O****utput*, pin**4**) e **RBI** (***R****ipple* ***B****lanking* ***I****nput*, pin**5**),

Le linee collegate al pin**3** e al pin**5** sono decisamente *d'ingresso* mentre quella collegata al pin**4** dispone di una circuiteria interna (una logica cablata di tipo ***wired-AND***) che permette sia di disporre di un *segnale d'uscita* **RBO** che di forzare su essa un *segnale d'ingresso* **BI**, senza produrre danni.

I segnali di controllo: ingresso BI

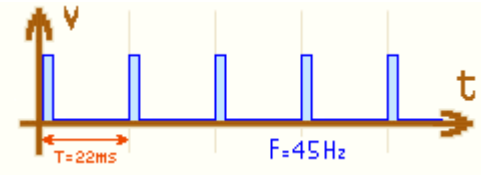
La linea collegata al pin**4**, utilizzata come ingresso (**BI**, ***B****lanking* ***I****nput*), se *attivata* (forzata *a* **0**), permette di **spegnere incondizionatamente** il **digit** collegato al **Decoder**, con qualunque codice presente sugli ingressi **DCBA**; nel funzionamento normale l'ingresso **BI** deve essere lasciato *scollegato* (o collegato al +5V), cioè *disattivo* (forzato *a* **1**); la figura mostra la predisposizione per il controllo di **BI** su un visualizzatore a 5 cifre, insieme a quello di **LT** (***L****amp* ***T****est*, pin**3**) che opera il servizio opposto (**accende** tutti i segmenti del **digit** collegato al **Decoder**); da notare che l'ingresso **BI** è *prioritario* rispetto a tutti gli altri, cioè quando esso è *attivo* contemporaneamente ad ogni altro ingresso la funzione degli altri viene ignorata.



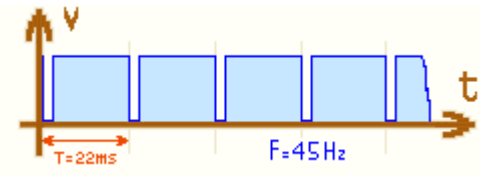
La tecnica di ***B****lanking* (*spegnimento*) può essere usata anche per controllare l'intensità luminosa del visualizzatore; è sufficiente applicare sul pin**4** un'*onda quadra* non simmetrica e modificare il suo *duty cycle*; come è noto questo parametro esprime (anche in percentuale) il rapporto tra la durata della *parte alta* e il periodo del segnale:

- se l'aspetto dell'*onda quadra* è quello di un treno d'impulsi di *breve durata*, molto distanziati tra loro (tipico di un *duty cycle* molto basso, per esempio del 5%) l'effetto sui **digit** è quello di *accenderli* (quasi) *al*

massimo



- se invece l'*onda quadra* assomiglia ad un treno d'impulsi *molto lunghi*, praticamente attaccati l'uno all'altro (tipico di un *duty cycle* molto alto, per esempio del 95%) l'effetto sui **digit** è quello di *tenerli* (quasi) *spenti*



Per garantire questo evento è per altro necessario che la *frequenza* del segnale così generato sia sufficientemente elevata, maggiore di 45 Hz per il rosso: solo così la veloce sequenza di spegnimenti (imposti da **BI** *attivo*, cioè a **0**) e accensioni (imposti da **BI** *disattivo*, cioè a **1**) può essere percepita come *variazione di luminosità* per effetto della *persistenza ottica* dell'immagine sulla retina dall'occhio umano.

I segnali di controllo: ingresso LT

Il segnale **LT** (***L**amp **T**est*), è disponibile sul pin**3** di tutti e cinque gli integrati: si tratta di un ingresso *attivo basso* che (posto a massa) *attiva* tutte le linee d'uscita di segmento, forzando l'accensione di tutti i segmenti del **digit**; il suo compito è quello di fornire uno strumento adatto a verificarne l'integrità.

La sua azione ha un livello di *priorità* secondo solo al segnale d'ingresso **BI** per cui potrà essere esercitata solo se **BI** non è contemporaneamente attivo.

I segnali di controllo: ingresso RBI e uscita RBO

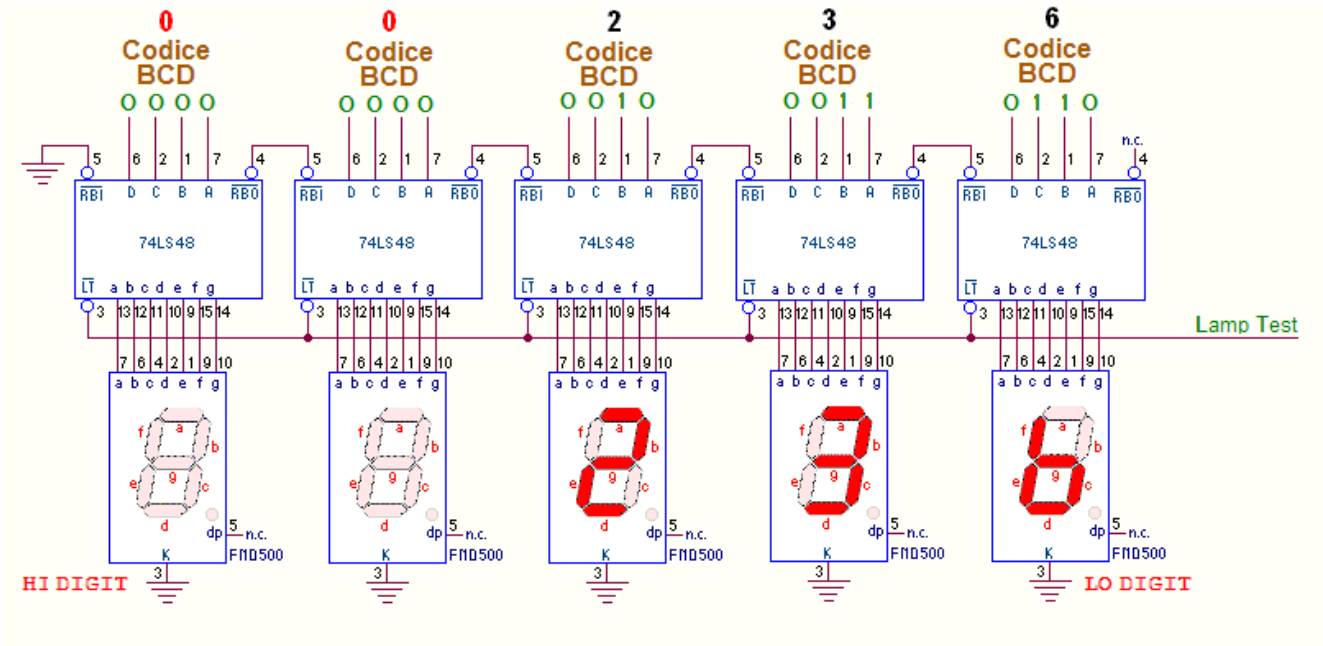
L'ingresso **RBI**, se *attivato* (forzato *a livello logico* **0**):

- consente di **spegnere** (*blanking*) il **digit** collegato al **Decoder** solo se contemporaneamente il codice presente sugli ingressi **DCBA** è $(0000)_2$; in queste condizioni forza a **0** anche l'uscita **RBO**

- se il codice presente sugli ingressi **DCBA** è diverso da $(0000)_2$ il **digit** mostra regolarmente i *simboli* corrispondenti e l'uscita **RBO** rimane *disattiva* (cioè *a livello logico* **1**)

La condizione per poter visualizzare comunque la cifra ***zero*** è di lasciare l'ingresso **RBI** *scollegato* (o meglio collegato al +5V), cioè *disattivo* (forzato *a livello logico* **1**).

L'uscita **RBO** è un utile meccanismo che consente di **evitare di accendere** tutti gli **zero a sinistra** di un numero, decisamente antiestetici e inutili; anche nella comune aritmetica sembra logico scrivere **236** piuttosto che **00236**; il metodo per raggiungere questo scopo è quello di collegare a massa l'ingresso **RBI** del **Decoder** collegato al **digit** *più significativo* e di collegare in cascata l'uscita **RBO** di ciascun digit (a cominciare da quello *più significativo*) con l'ingresso **RBI** del digit *successivo* (subito a *destra*):

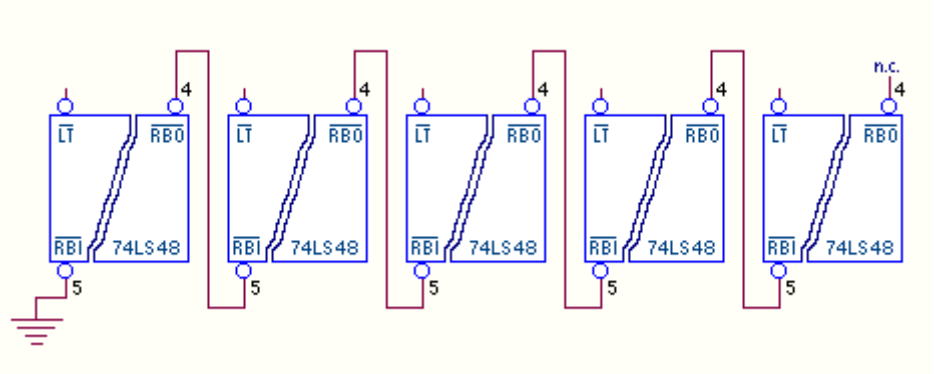


Nell'ipotesi che i **codici BCD** in ingresso ai cinque **Decoder** siano quelli corrispondenti alle cifre **00236**, vediamo come si realizza questa tecnica:

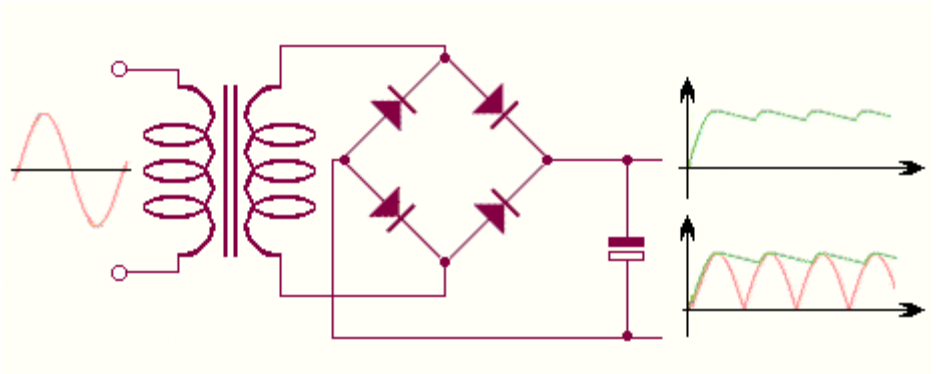
- fissando *a massa* l'ingresso **RBI** del **Decoder** collegato al **digit** *più significativo* se (come nel nostro esempio) sui suoi ingressi **DCBA** è presente del codice $(0000)_2$, si evita la visualizzazione dello 0 su di esso (***B**lanking* della cifra)

- questa situazione forza *a* **0** anche l'uscita **RBO**; essa a sua volta forza *a massa* l'ingresso **RBI** del **Decoder** collegato al **digit** *successivo* (subito a *destra*) evitando la visualizzazione dello 0 se sui suoi ingressi **DCBA** è presente del codice $(0000)_2$
- poiché il nostro esempio prevede *proprio questo* anche il secondo **digit** (subito a *destra* del primo) rimane spento e, in coerenza con la regola, il **Decoder** ad esso associato forza *a* **0** anche la sua uscita **RBO**; essa a sua volta forza *a massa* anche l'ingresso **RBI** del **Decoder** collegato al **digit** *successivo* (il terzo da sinistra)
- poiché il codice presente sugli ingressi **DCBA** di questo **Decoder** ***non*** *è* $(0000)_2$ il fatto che il suo ingresso **RBI** sia *a livello logico* **0** è ora del tutto irrilevante e, da questa posizione in poi ogni cifra verrà visualizzata, anche gli eventuali 0 interni

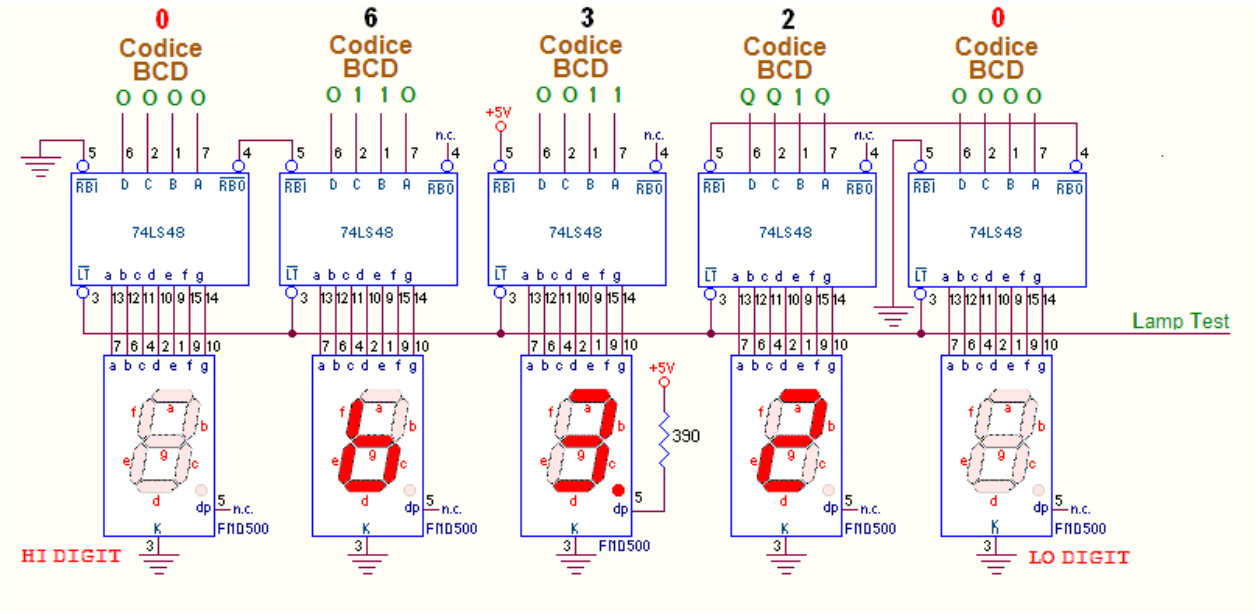
Una **curiosità**: osservando lo schema seguente si capisce la ragione per la quale gli ingressi (**RBI**, ***I**nput*) e le uscite (**RBO**, ***O**utput*) di ***B**lanking* vengano caratterizzate dall'attributo ***Ripple*** (il *gergo elettronico* anglosassone utilizza molto frequentemente queste definizioni onomatopeiche): il temine può essere tradotto con ***ondulazione***, ed è proprio il senso che danno i quattro collegamenti tra **RBI** e **RBO**:



Tra l'altro esso è utilizzato anche in un altro ambito, molto comune: ***Ripple*** è l'***ondulazione*** *residua* rilevabile in uscita ad un semplice alimentatore non stabilizzato (*trasformatore* + *ponte di greatz* + *condensatore elettrolitico*) sottoposto ad un piccolo assorbimento di corrente:



All'occorrenza questo artificio può essere applicato anche per **evitare di accendere** tutti gli **zero a destra** di un numero; la cosa ha senso solo se ci si riferisce alla *parte frazionaria* di un numero; per esempio sembra logico scrivere **63,2** piuttosto che **063,20**:



Ripetiamo l'analisi del visualizzatore precedente, ora organizzato per visualizzare la *parte intera* di un numero sui primi tre **digit** e la *parte fazionaria* sugli ultimi due; la situazione è evidenziata dalla presenza del *punto decimale* acceso permanentemente sul **digit** centrale. L'ipotesi di lavoro è che i **codici BCD** in ingresso ai cinque **Decoder** siano ora quelli corrispondenti alle cifre **06320**; la gestione della *parte intera* è identica a quella già descritta:

- il **Decoder** associato al **digit** *più significativo* ha l'ingresso **RBI** *a massa* perciò se (come nel nostro esempio) sui suoi ingressi **DCBA** è

presente del codice $(0000)_2$, la visualizzazione dello  viene evitata (***B**lanking* della cifra) e la sua uscita **RBO** viene forzata *a massa*
- poiché la citata uscita **RBO** è collegata all'ingresso **RBI** del **Decoder** associato al **digit** *successivo* (subito a *destra*), l'eventuale presenza sui suoi ingressi **DCBA** del codice $(0000)_2$ eviterà anche per esso la visualizzazione dello 
- il **Decoder** del **digit** *centrale* (quello con *punto decimale* acceso) non è soggetto a nessun controllo **RBI** per cui mostrerà qualunque cifra decimale, compreso lo

La gestione della *parte fazionaria* riflette la stessa logica ma, dovendo evitare l'accensione degli **zero a destra** del numero, il controllo dovrà essere esercitato a partire dal **digit** *meno significativo*:

- fissando *a massa* l'ingresso **RBI** del **Decoder** collegato al **digit** *meno significativo* se (come nel nostro esempio) sui suoi ingressi **DCBA** è presente del codice $(0000)_2$, la visualizzazione dello  su di esso viene evitata
- questa situazione forza *a* **0** anche l'uscita **RBO** che, a sua volta, forza *a massa* l'ingresso **RBI** del **Decoder** collegato al **digit** *precedente* (subito a *sinistra*) evitando la visualizzazione dello  se sui suoi ingressi **DCBA** è presente del codice $(0000)_2$

DECODER TTL: LE TABELLE DI VERITA'

Ogni componente logico è progettato a partire da una **Tabella** chiamata a stabilire il livello logico che tutte le sue *uscite* assumeranno in funzione delle possibili combinazioni dei propri *ingressi*; questa struttura, detta **Tabella di verità**, è sempre fornita dai *data sheet*, presentata spesso come ***Truth Table*** o ***Function Table***.

Poiché essa è grado di **sintetizzare** il suo funzionamento è molto importante *saperla consultare* e cercare di *capirne il contenuto*: ogni considerazione proposta in precedenza è frutto di una attenta lettura di queste Tabelle.

Di seguito ho composto quella del **Decoder TTL 74LS47** (e **7446**, entrambi con uscite ***open-collector*** *attive basse*, adatti a **digit** ad **anodo comune**):

7446 7447 74LS47 74246 74247 74LS247

	INGRESSI						USCITE (*attive **basse***)							
n°	**LT**	**RBI**	**D**	**C**	**B**	**A**	**BI/RBO**	**a**	**b**	**c**	**d**	**e**	**f**	**g**
0	1	**0**	0	0	0	0	0	1	1	1	1	1	1	1
		1					1	0	0	0	0	0	0	1
1	1	X	0	0	0	1	1	1	0	0	1	1	1	1
2	1	X	0	0	1	0	1	0	0	1	0	0	1	0
3	1	X	0	0	1	1	1	0	0	0	0	1	1	0
4	1	X	0	1	0	0	1	1	0	0	1	1	0	0
5	1	X	0	1	0	1	1	0	1	0	0	1	0	0
6	1	X	0	1	1	0	1	1	1	0	0	0	0	0
7	1	X	0	1	1	1	1	0	0	0	1	1	1	1
8	1	X	1	0	0	0	1	0	0	0	0	0	0	0
9	1	X	1	0	0	1	1	0	0	0	1	1	0	0
10	1	X	1	0	1	0	1	1	1	1	0	0	1	0
11	1	X	1	0	1	1	1	1	1	0	0	1	1	0
12	1	X	1	1	0	0	1	1	0	1	1	1	0	0
13	1	X	1	1	0	1	1	0	1	1	0	1	0	0
14	1	X	1	1	1	0	1	1	1	1	0	0	0	0
15	1	X	1	1	1	1	1	1	1	1	1	1	1	1
BI	X	X	X	X	X	X	**0**	1	1	1	1	1	1	1
LT	**0**	X	X	X	X	X	1	0	0	0	0	0	0	0

… e quella del **Decoder TTL 74LS48** (con uscite ***pull-up 2k*** *attive alte*, adatti a **digit** a **catodo comune**):

7448 74LS48 74248 74LS248														
	INGRESSI						USCITE (*attive* ***alte***)							
n°	**LT**	**RBI**	**D**	**C**	**B**	**A**	**BI/RBO**	**a**	**b**	**c**	**d**	**e**	**f**	**g**
0	1	**0**	0	0	0	0	0	0	0	0	0	0	0	0
		1					1	1	1	1	1	1	1	0
1	1	X	0	0	0	1	1	0	1	1	0	0	0	0
2	1	X	0	0	1	0	1	1	1	0	1	1	0	1
3	1	X	0	0	1	1	1	1	1	1	1	0	0	1
4	1	X	0	1	0	0	1	0	1	1	0	0	1	1
5	1	X	0	1	0	1	1	1	0	1	1	0	1	1
6	1	X	0	1	1	0	1	0	0	1	1	1	1	1
7	1	X	0	1	1	1	1	1	1	1	0	0	0	0
8	1	X	1	0	0	0	1	1	1	1	1	1	1	1
9	1	X	1	0	0	1	1	1	1	1	0	0	1	1
10	1	X	1	0	1	0	1	0	0	0	1	1	0	1
11	1	X	1	0	1	1	1	0	0	1	1	0	0	1
12	1	X	1	1	0	0	1	0	1	0	0	0	1	1
13	1	X	1	1	0	1	1	1	0	0	1	0	1	1
14	1	X	1	1	1	0	1	0	0	0	1	1	1	1
15	1	X	1	1	1	1	1	0	0	0	0	0	0	0
BI	X	X	X	X	X	X	**0**	0	0	0	0	0	0	0
LT	**0**	X	X	X	X	X	1	1	1	1	1	1	1	1

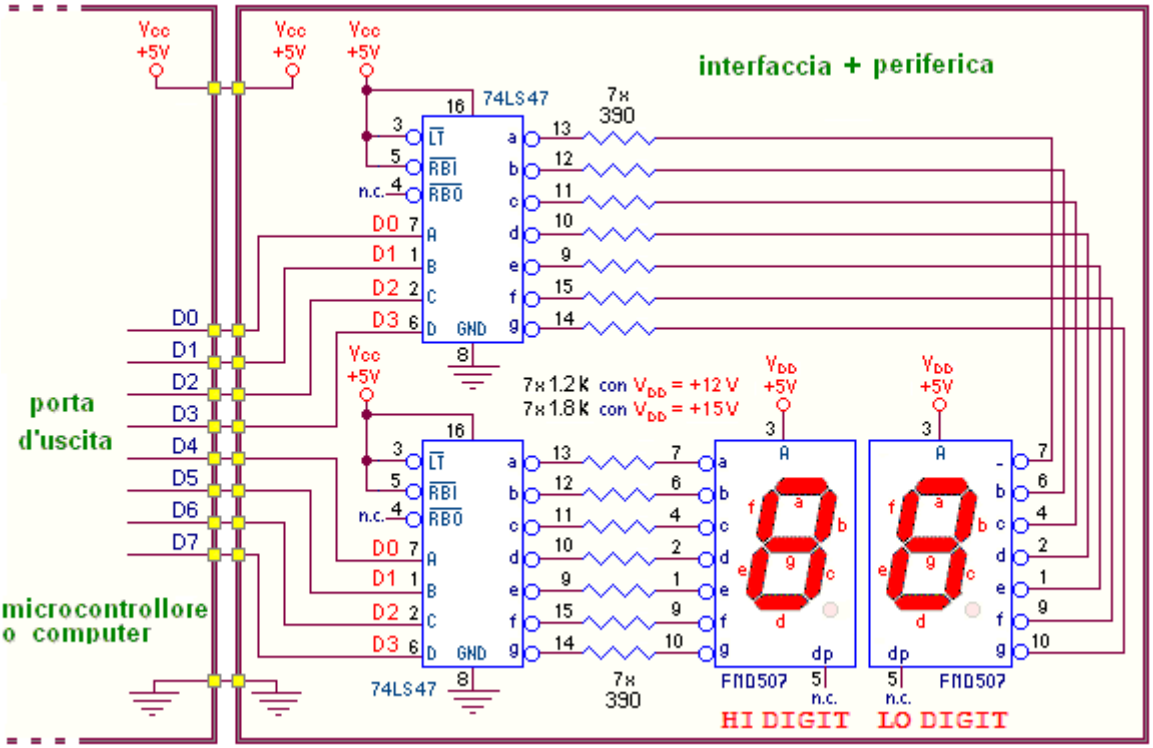
Sebbene il **Decoder TTL 74LS49** (con uscite ***open-collector*** *attive alte*, adatto a **digit** a **catodo comune**) abbia poca rilevanza, allego per completezza anche la sua **Tabella** funzionale:

74LS49						74249							74LS249
	INGRESSI						USCITE (*attive **alte***)						
n°	BI	D	C	B	A		a	b	c	d	e	f	g
0	0	X	X	X	X		0	0	0	0	0	0	0
	1	0	0	0	0		1	1	1	1	1	1	0
1	1	0	0	0	1		0	1	1	0	0	0	0
2	1	0	0	1	0		1	1	0	1	1	0	1
3	1	0	0	1	1		1	1	1	1	0	0	1
4	1	0	1	0	0		0	1	1	0	0	1	1
5	1	0	1	0	1		1	0	1	1	0	1	1
6	1	0	1	1	0		0	0	1	1	1	1	1
7	1	0	1	1	1		1	1	1	0	0	0	0
8	1	1	0	0	0		1	1	1	1	1	1	1
9	1	1	0	0	1		1	1	1	0	0	1	1
10	1	1	0	1	0		0	0	0	1	1	0	1
11	1	1	0	1	1		0	0	1	1	0	0	1
12	1	1	1	0	0		0	1	0	0	0	1	1
13	1	1	1	0	1		1	0	0	1	0	1	1
14	1	1	1	1	0		0	0	0	1	1	1	1
15	1	1	1	1	1		0	0	0	0	0	0	0

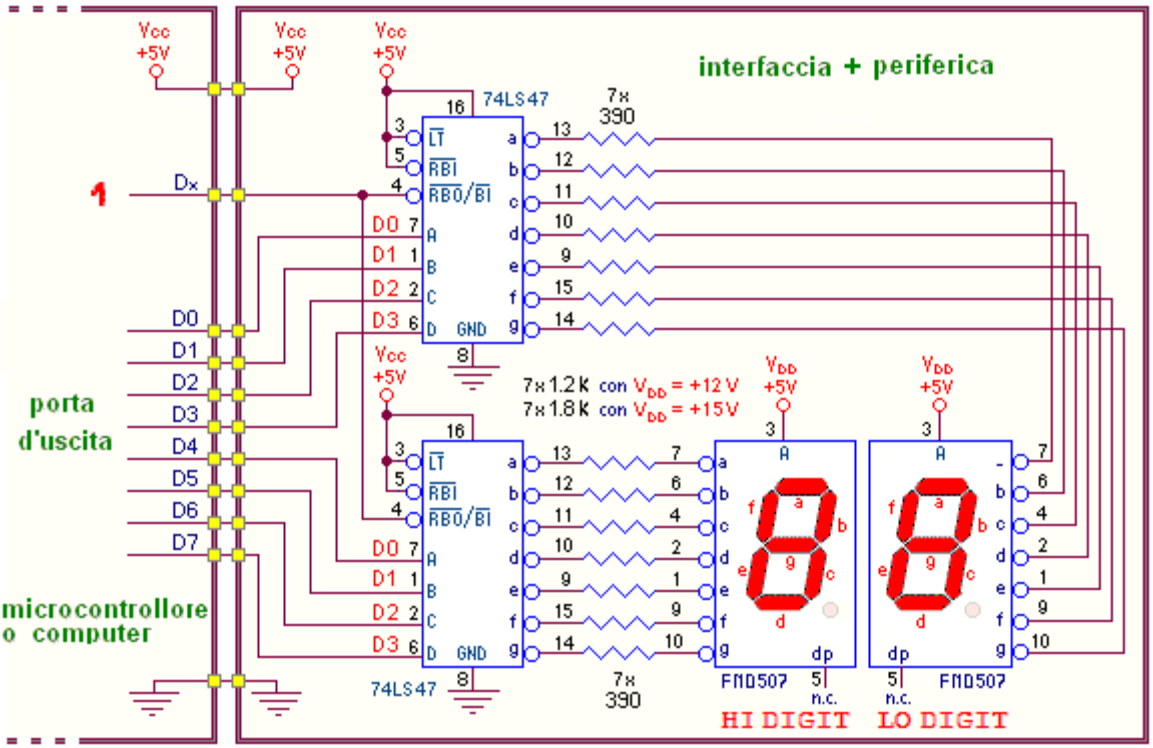
INTERFACCIA CON DECODER TTL PER VISUALIZZATORE A 2 DIGIT CON INGRESSO BCD

La conoscenza dei componenti (**Decoder da BCD a *sette segmenti***) in grado pilotare l'accensione dei segmenti di un **digit** ci permette finalmente di produrre gli schemi d'**interfaccia** adatti per la *porta d'uscita* di un *microcontrollore* o per la *porta parallela* di un *PC*.

I primi due progetti sono basati sui **Decoder TTL 74LS47** e sono caratterizzati dalla presenza di **resistori** (in serie o di *pull-up*) su ciascuna delle 14 linee d'uscita *attive basse*, adatte al controllo dei *7 segmenti* di **digit** ad **anodo comune**:

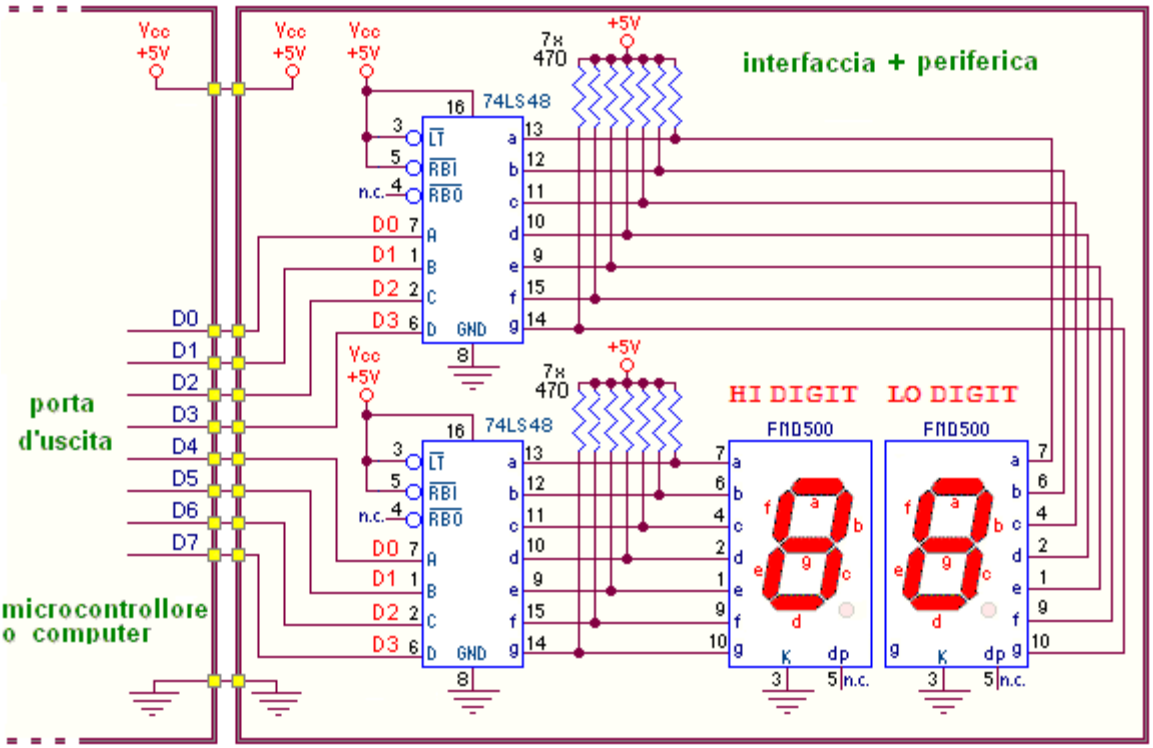


Le due linee *d'ingresso* (entrambe *attive basse*) **LT** (***L**amp **T**est*, che accende di tutti i segmenti) e di **RBI** (***R**ipple **B**lanking **I**nput*, che invece li spegne se contemporaneamente gli ingressi **DCBA** sono a $(0000)_2$) non sembrano particolarmente utili in un ambito programmabile, per cui sono rese *inattive*, collegandole a +5V (sebbene non sia consigliabile potevano anche essere lasciate *scollegate*); più interessante la linea **BI** / **RBO** (***B**lanking **I**nput* / ***R**ipple **B**lanking **O**utput*) la cui circuiteria interna permette di forzare su essa il segnale *d'ingresso* **BI** (*attivo basso*) in grado di spegnere tutti i segmenti incondizionatamente; la variante circuitale mostra come utilizzare una linea di una seconda *porta d'uscita* di *micro* o *PC* per esercitare questo controllo:

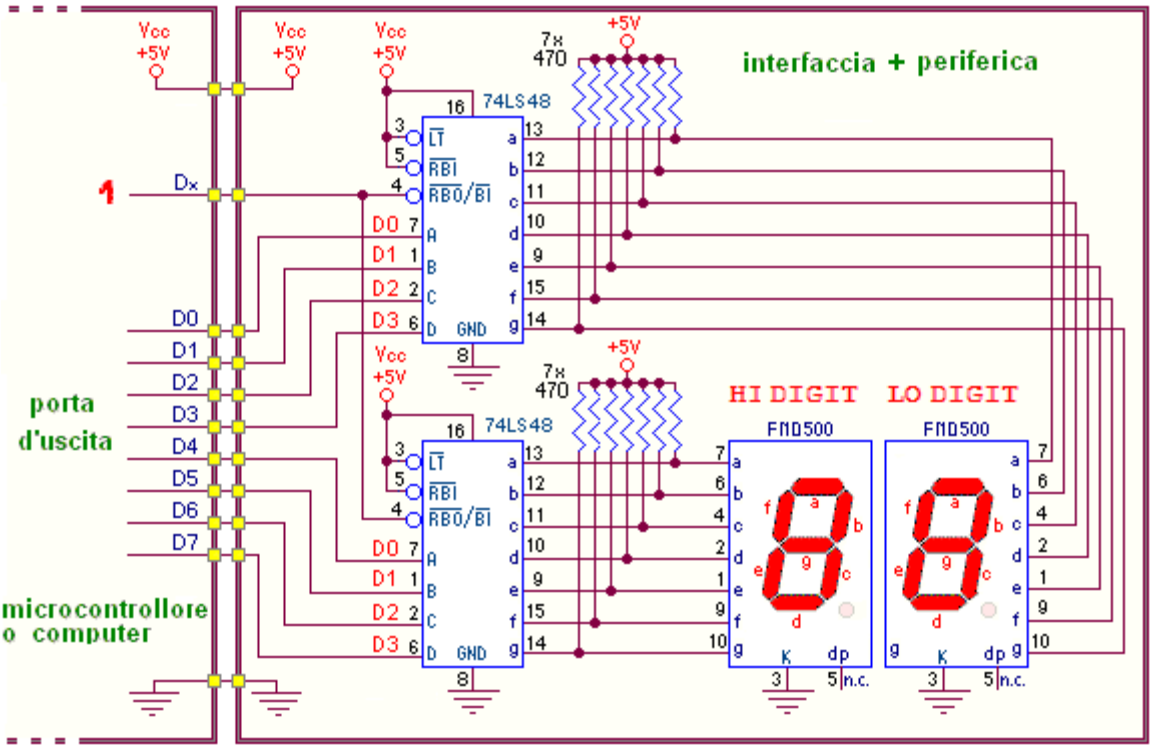


Il programma che controllerà questa **interfaccia** dovrà farsi carico di inizializzare a **1** la linea di controllo per **BI** e di formattare in **BCD** i dati binari da spedire sulla *porta principale*: ognuna delle ***6*** *combinazioni* non appartenente al **codice** [da $(1010)_2$ a $(1111)_2$] provocherebbe una inopportuna visualizzazione *non decimale*; tuttavia proprio questa contingenza offre una inattesa possibilità che rende inutile la variante hardware appena proposta: è possibile spegnere tutti i segmenti incondizionatamente **anche** senza dover intervenire su **BI**, se da software si spedisce la *sequenza* $(11111111)_2$.

La seconda serie di due progetti è basata sui **Decoder TTL 74LS48** ed è caratterizzata dalla presenza di **resistori** addizionali esterni di ***pull-up*** su ciascuna delle 14 linee d'uscita *attive alte*, solo così adatte al controllo dei *7 segmenti* di **digit** a **catodo comune**:



Sebbene il suo funzionamento sia complementare a quello del **74LS48** le considerazioni sulle tre linee *d'ingresso* sono le stesse: **LT** e **RBI**, non essendo utili per il nostro scopo, sono rese entrambe *inattive*, collegandole a +5V, mentre la linea **BI** / **RBO** può essere usata come *ingresso attivo basso* (in virtù della circuiteria interna che lo permette) per spegnere tutti i segmenti incondizionatamente, in accordo con la variante circuitale mostrata qui di seguito:



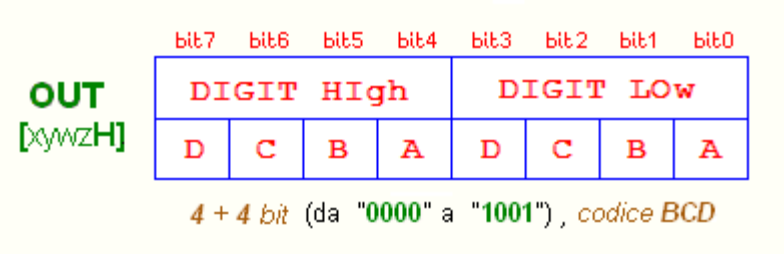
Anche per il programma che controllerà questa **interfaccia** valgono le stesse considerazioni:

- sulla *porta principale* dovranno essere forniti dati binari rigorosamente **BCD**, per evitare inopportune visualizzazioni *non decimali* e linea di controllo **BI** dovrà essere inizializzata a **1**
- la possibilità di spedire, da software, la *sequenza* $(11111111)_2$ sulla *porta principale* rende inutile la variante hardware appena proposta, essendo essa associata alla capacità di spegnere tutti i segmenti incondizionatamente **anche** senza dover intervenire su **BI**

Da notare che nessuno dei quattro progetti consente la **gestione diretta** dei *punti decimali* dei due **digit**; all'occorrenza è comunque possibile associarne il controllo a due linee di una seconda *porta d'uscita* (con *microcontrollore*) o due linee delle quattro linee del **Registro 037AH / 027AH** (con la *porta parallela* del *PC*, vedi http://www.giobe2000.it/HW/Parallela/Pag/RegistriSPP4.htm).

INTERFACCIA CON DECODER TTL : TABELLA DI ASSOCIAZIONE LOGICA

La seguente **tabella di associazione logica** è chiamata a *virtualizzare* tutte e quattro le soluzioni proposte :



Nella prossima puntata riprenderemo il discorso dell'**interfacciamento** di due **digit** , occupandoci dei due **Decoder** non ancora trattati, entrambi dotati della possibilità di **memorizzare il dato** che li attraversa e corredati da un importante **driver** d'uscita in grado di assicurare le condizioni ottimali per ciascuno dei **Led** ad essi collegati.

Puoi scaricare **qui** la **versione PDF** della **QUARTA PARTE**

Alcuni argomenti **citati** in questa parte sono **disponibili** sul sito

Giobe2000

una ricca raccolta di dispositivi e progetti
pensata per aiutarti a comprendere i segreti del tuo computer
[sia *Personal* che *microcontrollore*]