

Progetto di Interfacce.....	1
Introduzione.....	2
Librerie per la gestione in Delpi dell' I/O sulla porta parallela.....	2
Operazioni OR, AND, XOR e NOT, Operazioni SHR e SHL .....	3
Come "forzare" alcuni bit ad 1 con una operazione OR.....	4
Come "forzare" alcuni bit ad 0 con una operazione AND.....	4
Come "negare selettivamente" alcuni bit con una operazione XOR.....	4
Alcune semplici proprietà .....	4
Impulso in logica negativa, sull'uscita DB7 .....	4
Impulso in logica positiva, sull'uscita DB2.....	5
Interfaccia con Display a 7 segmenti non decodificato.....	5
Driver per la gestione del Display a 7 segmenti (non decodificato).....	7
Applicativo demo per Display a 7 segmenti.....	8
Interfaccia con 2 Display a 7 segmenti decodificati.....	8
Driver per la gestione dei 2 Display con decoder .....	9
Applicativo demo per 2 Display decodificati .....	10
Interfaccia con 4 Display a 7 segmenti decodificati.....	10
Driver per la gestione dei 4 Display con decoder .....	11
Applicativo demo per 4 Display decodificati .....	12
Interfaccia per la lettura di 3 Selettori digitali.....	13
Driver per la gestione dei 3 Selettori digitali.....	14
UTILIZZO DELLA PORTA PARALLELA COME INPUT .....	15
Indirizzi dei Registri interni della Porta Parallela della Stampante: .....	15
Appendice B: Connettore standard DB25F .....	16
Indirizzi dei Registri interni della Porta Parallela della Stampante: .....	16

## Progetto di Interfacce

prof. Cleto Azzani  
 IPSIA Moretto Brescia  
 Febbraio 2005  
 (PRELIMINARY)

## **Introduzione**

In queste pagine si cercherà di illustrare quali siano i passi da seguire quando si voglia interfacciare un dispositivo digitale (solitamente) alla porta parallela di un PC.

Il progetto di una interfaccia ha come presupposto fondamentale la conoscenza sufficientemente approfondita delle caratteristiche hardware della porta parallela del PC e naturalmente una conoscenza altrettanto accurata del dispositivo da interfacciare.

Il primo passo è rappresentato dalla definizione delle interconnessioni Hardware fra porta parallela e “dispositivo da interfacciare”; si dovrà pertanto predisporre un disegno, uno schema elettrico in cui siano chiare ed inequivocabili tutte le interconnessioni che si intende effettuare.

Il secondo passo è quello di scrivere, adottando un linguaggio di programmazione opportuno (per noi Borland Delphi), le “primitive software” necessarie a pilotare il “dispositivo da interfacciare”. Queste primitive software vengono genericamente denominate “driver del dispositivo” e sono il ponte di collegamento fra il software applicativo che deve utilizzare un determinato dispositivo e l’hardware di quel dispositivo.

Il terzo passo sarà quello di scrivere il “software applicativo”.

Per dare concretezza alle nostre intenzioni svilupperemo nel seguito una serie di “progetti” attraverso i quali sia possibile esaminare prima gli aspetti delle interconnessioni hardware poi gli aspetti della stesura del “driver” ed infine la predisposizione di un “software applicativo” in Delphi, che ci consenta per lo meno di effettuare il test del dispositivo interfacciato.

## **Librerie per la gestione in Delphi dell’ I/O sulla porta parallela**

Se sul nostro PC è installato il sistema operativo Windows 95 o 98 possiamo tranquillamente utilizzare la libreria PortW95 (disponibile in formato sorgente sul nostro sito) che interagisce direttamente con l’hardware della porta parallela. Per gli esempi che intendiamo sviluppare è sufficiente appoggiarsi alla funzione PortReadByte per le operazioni di lettura (input dalla porta):

```
function PortReadByte(Addr:Word) : Byte;
```

alla procedura PortWriteByte per le operazioni di scrittura (output verso la porta):

```
procedure PortWriteByte(Addr:Word; Value:Byte);
```

Esempio1: Desidero leggere lo stato della porta LPT1 (indirizzo \$378) e collocarlo in una zona temporanea X (variabile locale di tipo byte). Basterà scrivere nel programma l’istruzione:

```
X := PortReadByte($378);
```

Esempio2: Desidero scrivere sulla porta LPT1 (indirizzo \$378) la quantità esadecimale \$A5 (%10100101). Basterà scrivere nel programma l’istruzione:

```
PortWriteByte($378,$A5);
```

Se sul nostro PC è installato il sistema operativo Windows 2000 o XP non è consentito (dal sistema operativo) interagire direttamente con i singoli bit della parallela; l’operazione può essere fatta solamente attraverso l’uso di librerie apposite che utilizzano le apposite API (Application Program Interface) di Windows. Nel caso di Delphi può essere utilizzata la libreria “freeware” ZPORTIO che può essere scaricata dal sito <http://www.specosoft.com>. Tale libreria è costituita da tre file: zlportio.sys, zlportio.pas e ddkint.pas che vanno collocati nella stessa directory dove si trova l’applicazione che stiamo sviluppando. L’eseguibile (il file EXE) della nostra applicazione dovrà

sempre essere abbinato a `zlportio.sys` per potere funzionare correttamente (i due files dovranno risiedere nella medesima cartella).

Per gli esempi che intendiamo sviluppare è sufficiente appoggiarsi alla funzione `PortReadB` per le operazioni di lettura (input dalla porta):

```
function PortReadB(Addr:Word)      : Byte;
```

alla procedura `PortWriteB` per le operazioni di scrittura (output verso la porta):

```
procedure PortWriteB(Addr:Word; Value:Byte);
```

Gli esempi di utilizzo della function `PortReadB` e della procedura `PortWriteB` sono identici a quelli proposti a pagina precedente.

N.B.: Nello startup del programma Delphi (ossia nell'evento di Creazione della Form principale) si inserisca la chiamata alla procedura

```
zliosetiopm(True); // con parametro True si consente alla applicazione di utilizzare le
                  // istruzioni assembler IN e OUT direttamente nel codice macchina
                  // ciò fornisce la migliore performance per le operazioni di I/O
```

### **Operazioni OR, AND, XOR e NOT, Operazioni SHR e SHL**

Gli operatori OR, AND, XOR, NOT, SHR e SHL sono operatori logici estremamente importanti in tutti quei contesti in cui si richiede di “manipolare un dato” a livello di bit. Scopo di questa piccola trattazione è quello di individuare le “operazioni maggiormente ricorrenti” e come vengono affrontate utilizzando le cosiddette “maschere”. Questi operatori logici vengono utilizzati su dati a 8 bit (byte) a 16 bit (word) e anche a 32 bit (longword).

Per semplicità, supponiamo di avere due dati a 8 bit:

```
A = $37  %0011 0111
B = $46  %0100 0110
```

Le tre operazioni OR, AND, XOR vengono eseguite fra i bit del dato A e i corrispondenti bit del dato B; i risultati delle operazioni saranno i seguenti:

```
A OR B = $77 %0111 0111
```

```
A AND B = $06 %0000 0110
```

```
A XOR B = $71 %0111 0001
```

L'operazione NOT viene eseguita su ogni singolo bit del dato pertanto avremo:

```
NOT A = $C8 %1100 1000
```

```
NOT B = $B9 %1011 1001
```

Le operazioni SHR/SHL provocano lo scorrimento del dato verso destra/sinistra di un certo numero di posti specificato; ad esempio:

A SHR 4 = \$03 %0000 0011      scorrimento verso destra di A (4 posti)

B SHL 4 = \$60 %0110 0000      scorrimento verso sinistra di B (4 posti)

Durante una operazione di “shift” i bit svuotati vengono sempre riempiti con “0”.

### Come “forzare” alcuni bit ad 1 con una operazione OR

Supponiamo che sia necessario forzare a 1 i bit: B0, B5 e B7 del dato A = \$ 37. L’operazione è molto semplice si costruisce una “maschera” ossia una costante M che ha i tre bit da forzare a livello 1 e tutti gli altri a livello 0;

M = \$A1 %1010 0001

La forzatura sul dato A viene effettuata eseguendo la istruzione

A := A OR M;      che da il risultato: \$B7 %1011 0111

### Come “forzare” alcuni bit ad 0 con una operazione AND

Supponiamo che sia necessario forzare a 0 i bit: B1, B5 e B7 del dato B = \$ 46. L’operazione è molto semplice si costruisce una “maschera” ossia una costante M che ha i tre bit da forzare a livello 0 e tutti gli altri a livello 1;

M = \$5D %0101 1101

La forzatura sul dato B viene effettuata eseguendo la istruzione

B := B AND M;      che da il risultato: \$44 %0100 0100

### Come “negare selettivamente” alcuni bit con una operazione XOR

Supponiamo che sia necessario negare i bit: B1, B5 e B7 del dato B = \$ 46. L’operazione è molto semplice si costruisce una “maschera” ossia una costante M che ha i tre bit da negare a livello 1 e tutti gli altri a livello 0;

M = \$A2 %1010 0010

La negazione selettiva sui bit prescelti del dato B viene effettuata eseguendo la istruzione

B := B XOR M;      che da il risultato: \$E4 %1110 0100

Alcune semplici proprietà

A XOR A      da sempre risultato 0 (A è sempre uguale a sé stesso)

A XOR (NOT A)      da come risultato \$FF

### Impulso in logica negativa, sull’uscita DB7

Per “generare” un impulso di breve durata, in logica negativa, sull’uscita DB7 della porta parallela dovrò in sequenza compiere le seguenti operazioni:

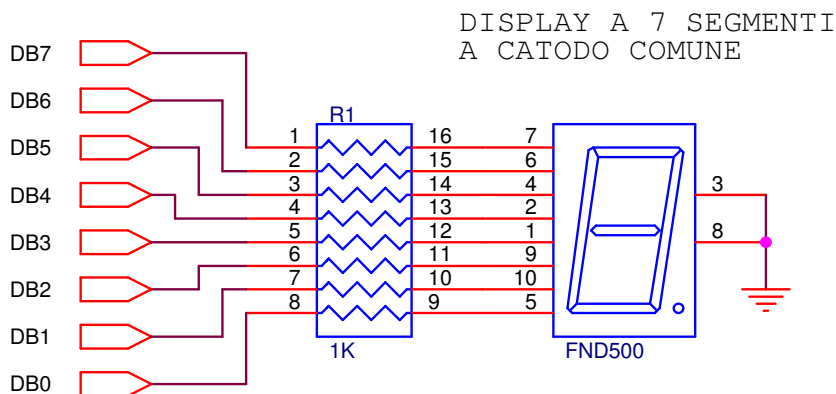
- 1) Leggo stato porta parallela;
- 2) Forzo a livello 1 il bit B7 del dato; (OR \$80)
- 3) Scrivo il dato sulla parallela;
- 4) Forzo a livello 0 il bit B7 del dato; (AND \$7F)
- 5) Scrivo il dato sulla parallela;
- 6) Forzo a livello 1 il bit B7 del dato; (OR \$80)
- 7) Scrivo il dato sulla parallela;

### Impulso in logica positiva, sull'uscita DB2

Per “generare” un impulso di breve durata, in logica positiva, sull'uscita DB2 della porta parallela dovrò in sequenza compiere le seguenti operazioni:

- 1) Leggo stato porta parallela;
- 2) Forzo a livello 0 il bit B2 del dato; (AND \$FB)
- 3) Scrivo il dato sulla parallela;
- 4) Forzo a livello 1 il bit B2 del dato; (OR \$04)
- 5) Scrivo il dato sulla parallela;
- 6) Forzo a livello 0 il bit B2 del dato; (AND \$FB)
- 7) Scrivo il dato sulla parallela;

### Interfaccia con Display a 7 segmenti non decodificato



Nello schema sopra riportato si nota che un display a 7 segmenti a catodo comune (tipo FND500 o equivalente) viene interconnesso direttamente alla porta parallela del PC senza l'utilizzo di alcuna decodifica. Il display è costituito da 8 diodi led: 7 per rappresentare la cifra (segmenti da A a G) e uno per la gestione accensione del “punto decimale” DP nel nostro caso “punto decimale destro”. La corrente necessaria per il funzionamento del display viene per così dire prelevata direttamente dalla porta parallela senza l'utilizzo di alcun circuito alimentatore esterno attraverso la interposizione fra porta parallela e display di una resistenza da 1K (una per ogni segmento). Ciò consente di limitare la corrente assorbita a pochi mA per ogni segmento; ovviamente a farne le spese sarà la “luminosità” del segmento acceso (per il quale sarebbero richiesti 10 mA) ma d'altro canto questa scelta salvaguarda l'hardware del PC e soprattutto semplifica molto il circuito di interfaccia rendendolo anche estremamente affidabile. Il circuito funziona in “logica positiva” ossia livello alto o “1” corrisponde allo stato di “segmento acceso”, livello basso o “0” corrisponde allo stato di “segmento spento”.

		SEGMENTI								CODICE
		A	B	C	D	E	F	G	DP	
DATO		DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0	HEX
0		1	1	1	1	1	1	0	0	\$FC
1		0	1	1	0	0	0	0	0	\$60
2		1	1	0	1	1	0	1	0	\$DA
3		1	1	1	1	0	0	1	0	\$F2
4		0	1	1	0	0	1	1	0	\$66
5		1	0	1	1	0	1	1	0	\$B6
6		1	0	1	1	1	1	1	0	\$BE
7		1	1	1	0	0	0	0	0	\$E0
8		1	1	1	1	1	1	1	0	\$FE
9		1	1	1	1	0	1	1	0	\$F6
A		1	1	1	0	1	1	1	0	\$EE
b		0	0	1	1	1	1	1	0	\$3E
C		1	0	0	1	1	1	0	0	\$9C
d		0	1	1	1	1	0	1	0	\$7A
E		1	0	0	1	1	1	1	0	\$9E
F		1	0	0	0	1	1	1	0	\$8E

La tabella sopra riportata presenta le varie combinazioni che devono assumere gli 8 bit della porta parallela per fare apparire sul display i 16 simboli esadecimali da “0” ad “F”. Nella colonna codice è riportato in esadecimale la codifica a 8 bit corrispondente alle varie situazioni prese in esame nella tabella. Sarà opportuno precisare che, come si può notare dalla tabella, il punto decimale del display rimane sempre spento anche per non gravare inutilmente sulle limitate capacità di pilotaggio della parallela.

I 16 simboli esadecimali non sono gli unici che si possono visualizzare sul display ve ne sono altri che seguito riportiamo a titolo di esempio:

		SEGMENTI								CODICE
		A	B	C	D	E	F	G	DP	
DATO		DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0	HEX
		0	0	0	0	0	0	0	0	\$00
H		0	1	1	0	1	1	1	0	\$6E
h		0	0	1	0	1	1	1	0	\$2E
n		0	0	1	0	1	0	1	0	\$2A
r		0	0	0	0	1	0	1	0	\$0A

Per “spegnere il display” dovremo inviare alla porta parallela il dato esadecimale \$00, per fare apparire la lettera “H” dovremo utilizzare il codice esadecimale \$6E, per la lettera “h” dovremo utilizzare il codice esadecimale \$2E e così via.

## Driver per la gestione del Display a 7 segmenti (non decodificato)

Scriviamo innanzitutto una procedura **Dato\_display** attraverso la quale sia possibile visualizzare su display un dato numerico compreso fra 0 e 15 (esadecimale da \$0 a \$F) supponendo di impiegare la libreria ZLPORTIO (uses ZLPORTIO).

```
{
  Visualizza su Display a 7 segmenti
  Un dato compreso fra 0 e 15
}
procedure Dato_display(N : byte);
  const TAB : Array[1..16] of byte = ($FC, $60, $DA, $F2, $66, $B6, $BE, $E0,
                                     $FE, $F6, $EE, $3E, $9C, $7A, $9E, $8E);

  var x : byte;
Begin

  N := N AND $0F; // evita l'overrange dell'indice del vettore
  x := TAB[N+1]; // recupero del codice dalla tabella
  PortWriteB($378, x); // PortWriteB(888, x);

end;
```

In questa procedura, i dati riportati nella tabella di pilotaggio del display a 7 segmenti vengono collocati all'interno di un vettore (Array[1..16] of byte).

Il dato, contenuto entro una variabile byte (range 0 – 255) viene mascherato con la quantità \$0F (%0000 1111); ciò evita che introducendo un dato errato si generi una condizione di errore di overrange dell'indice del vettore (che deve essere sempre compreso fra 1 e 16) con conseguente blocco del programma.

L'istruzione `x := TAB[N+1]` provvede a recuperare all'interno del vettore, che funge da “lookup-table” il codice esadecimale opportuno da inviare sulla porta parallela LPT1 (\$378/888).

Scriviamo una procedura **Clear\_display** attraverso la quale sia possibile spegnere il display.

```
{
  Spegne il Display a 7 segmenti
}
procedure Clear_Display;

Begin

  PortWriteB($378, 0);

end;
```

Scriviamo una procedura **Test\_display** attraverso la quale sia possibile accendere tutti i segmenti del display verificando la corretta funzionalità dell'hardware.

```
{
  Test sul Display a 7 segmenti
}
procedure Test_Display;

Begin

  Dato_Display(8);

end;
```

## Applicativo demo per Display a 7 segmenti

Terminato il Driver, costituito dalle tre procedure sopra riportate, ora scriviamo un piccolo applicativo che, effettui il test del display per 5 sec, faccia uscire in sequenza da 0 a F tutti i simboli del sistema esadecimale intervallandoli di 1 sec. e poi ripeta l'operazione in senso inverso; alla fine il display dovrà essere spento.

```

{
  Applicativo dimostrativo su Display a 7 segmenti
}
procedure Applicativo1;
  var k : byte;

Begin

  Test_Display; // accende tutti i segmenti del Display

  Sleep(5000); // ritardo di 5000 ms. pari a 5 sec.

  For k := 0 to $F do
    Begin
      Dato_Display(k);
      Sleep(1000); // ritardo di 1 sec.
    End;

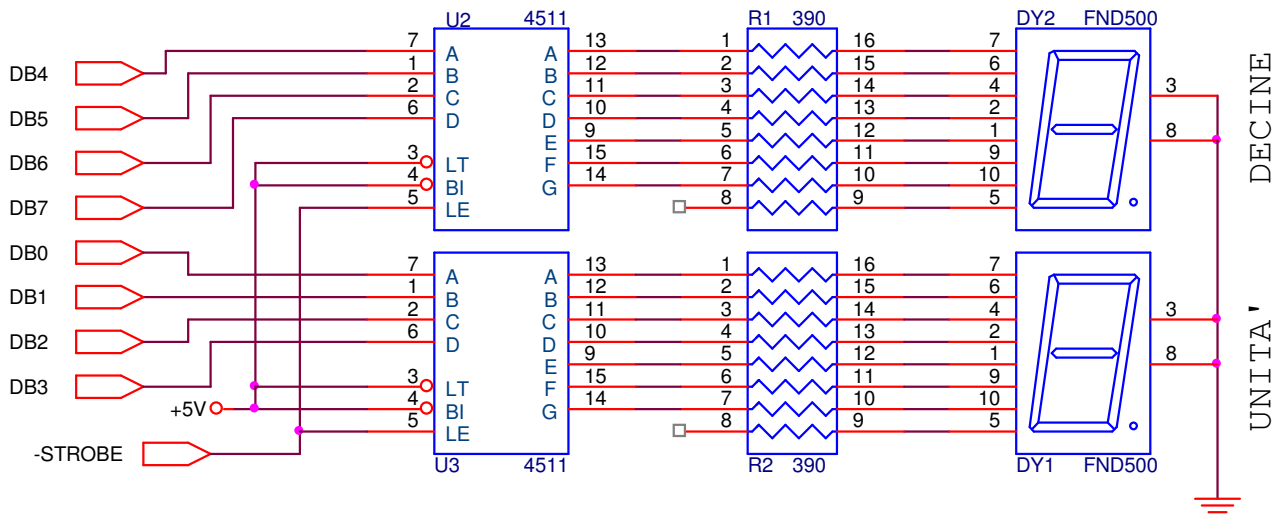
  For k := $F downto 0 do
    Begin
      Dato_Display(k);
      Sleep(1000); // ritardo di 1 sec.
    End;

  Clear_Display; // spegne il Display

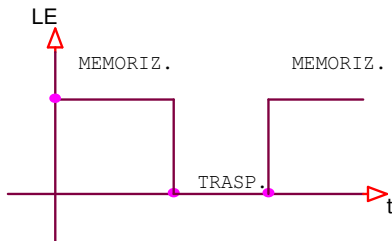
end;

```

## Interfaccia con 2 Display a 7 segmenti decodificati



Nello schema sopra riportato si nota la presenza di 2 display a 7 segmenti a catodo comune (tipo FND500 o equivalente) connessi alla porta parallela del PC tramite 2 decoder con latch tipo 4511. DY2 e U3 sono connessi ai 4 bit meno significativi della porta-dati DY1 e U2 sono connessi ai 4 bit più significativi della porta-dati. DY2 ha il compito di visualizzare le unità, DY1 ha il compito di visualizzare le decine.



Il segnale LE (Latch Enable) richiesto dai decoder 4511, o si mantiene fisso a livello logico basso “0” (4511 in “stato di trasparenza”) oppure viene mantenuto normalmente a livello alto “1” (4511 in fase di memorizzazione) e portato a livello basso “0” solo per il tempo strettamente necessario per memorizzare il dato presente sugli ingressi A-B-C-D nel latch del decoder (vedi grafico temporale sopra riportato).

### Driver per la gestione dei 2 Display con decoder

Scriviamo innanzitutto una procedura **InitLE** attraverso la quale i due decoder 4511 vengono posti o in stato di trasparenza LE=0 o in stato di memorizzazione LE=1.

```
{
  Pone i due decoder 4511 in stato di trasparenza o di memorizzazione
  Se d = 0   LE = 0
  Se d <>0   LE = 1
}
procedure InitLE(d:byte);

  var x : byte;
Begin
  x := PortReadB($37A); // Leggo lo stato del registro di controllo

  If d=0 Then
    Begin
      x := x OR $01; // Pongo a livello alto il bit C0 del registro di controllo
                  // il segnale di strobe passa perciò a livello basso STROBE = C0-
    End
  Else
    Begin
      x := x AND $FE; // Pongo a livello basso il bit C0 del registro di controllo
                  // il segnale di strobe passa perciò a livello alto STROBE = C0-
    End;
  PortWriteB($37A, x);
end;
```

Scriviamo una procedura **Vis\_dato** attraverso la quale il dato BCD viene inviato ai due display.

```
{
  Visualizza su 2 Display a 7 segmenti
  Un dato compreso fra 0 e 99 (in formato BCD)
  In caso di dato errato vengono visualizzate su display due lettere E
}
procedure Vis_Dato(dato : byte);

  var x, d, u : byte;
Begin

  If Dato>99 Then x := $EE
  Else
    Begin
      d := dato DIV 10; // ricavo decine
      u := dato MOD 10; // ricavo unità
      x := (d LSR 4) OR u;
    End;
  PortWriteB($378, x);
end;
```

## Applicativo demo per 2 Display decodificati

Terminato il Driver, costituito dalle due procedure sopra riportate, ora scriviamo un piccolo applicativo che, inizializzi l'interfaccia, faccia uscire in sequenza i numeri in codice BCD compresi fra 00 a 99 intervallandoli di 1 sec. e poi ripeta l'operazione in senso inverso; alla fine il display dovrà essere azzerato.

```

{
  Applicativo dimostrativo su 2 Display decodificati
}
procedure Applicazione2;
  var k : byte;

Begin

  InitLE(0); // pone LE a livello 0

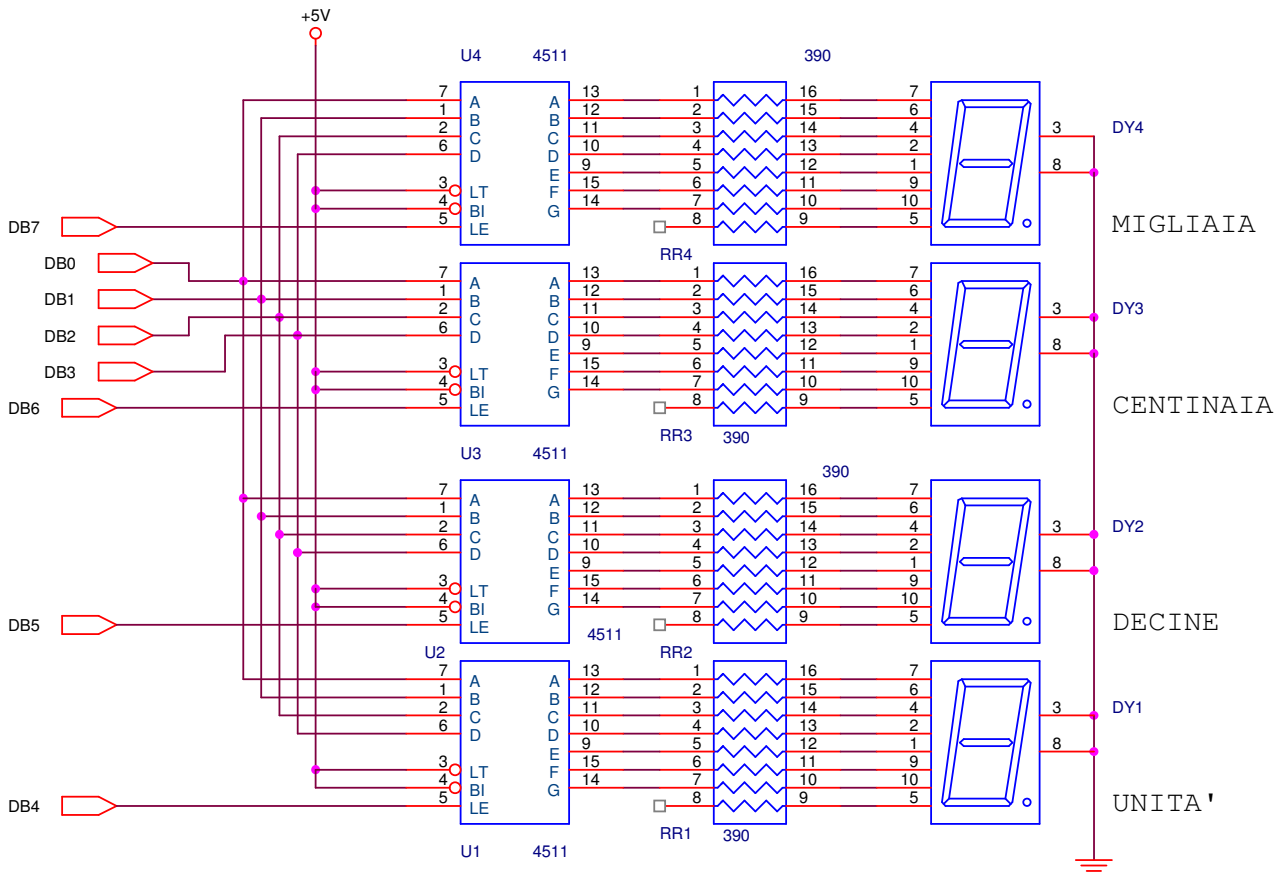
  For k := 0 to 99 do
    Begin
      Vis_Dato(k);
      Sleep(1000); // ritardo di 1 sec.
    End;

  For k := 99 downto 0 do
    Begin
      Vis_Dato(k);
      Sleep(1000); // ritardo di 1 sec.
    End;

  InitLE(1); // pone LE a livello 1

end;
```

## Interfaccia con 4 Display a 7 segmenti decodificati



Nello schema sopra riportato si nota la presenza di 4 display a 7 segmenti a catodo comune (tipo FND500 o equivalente) connessi alla porta parallela del PC tramite 4 decoder-latch tipo 4511. Le linee DATI A-B-C-D dei 4 decoder sono connesse ai 4 bit meno significativi della porta-dati che pertanto fungono da BUS DATI da DB0 a DB3. DB4 è connesso a LE1, DB5 a LE2, DB6 a LE3 ed infine DB7 a LE4 queste linee servono per “indirizzare” in modo opportuno i dati all’interno di un determinato decoder e di conseguenza farli apparire sul corrispondente display.

LE4	LE3	LE2	LE1						
1	1	1	1	0	0	0	0	0	\$F0
1	1	1	0	0	0	0	0	0	\$E0
1	1	0	1	0	0	0	0	0	\$D0
1	0	1	1	0	0	0	0	0	\$B0
0	1	1	1	0	0	0	0	0	\$70

Se pongo sulla porta-dati la quantità esadecimale \$F0, tutti i 4 decoder sono posti in stato di memorizzazione; per trasferire il dato “0” nel latch 1 devo collocare sulla porta-dati la quantità esadecimale \$E0; per trasferire il dato “4” nel latch 2 devo utilizzare il codice \$D4 e così via dicendo.

### Driver per la gestione dei 4 Display con decoder

Scriviamo una procedura **Clear\_Display** che faccia apparire la scritta 0000 sui quattro display.

```

{
  Cancella i 4 display
}
procedure Clear_display;

Begin
  PortWriteB($378, 0); // Invio il dato "0" a tutti i decoder-latch

  PortWriteB($378, $F0); // Pongo I 4 decoder in stato di memorizzazione

end;
```

Scriviamo una procedura **Out\_Dato** attraverso la quale un valore esadecimale a 4 bit venga inviato ad uno fra i display presenti nell’interfaccia (identificato tramite il suo indirizzo).

```

{
  Visualizza un valore esadecimale a 4 bit
  all'interno di uno dei 4 display;
  dato : byte contiene il valore da visualizzare compreso fra $0 e $F
  addr : byte contiene l'indirizzo del display 0 per DY1, 1 per DY2,... 3 per DY4
}
procedure Out_Dato(dato, addr : byte);

  var mask, x : byte;
Begin

  Dato := dato AND $F ; // evita l'overrange del dato (range $0 - $F)
  Addr := Addr AND $3 ; // evita overrange di Addr (range $0 - $3)

  Case Addr of
    0 : mask := $E0;
    1 : mask := $D0;
    2 : mask := $B0;
    3 : mask := $70;
  End;
```

```

X := mask OR Dato; // incastro fra maschera e dato
PortWriteB($378, x); // uscita sulla porta
X := $F0 OR Dato; // ripristino stato di memorizzazione nei latch
PortWriteB($378, x);
end;

```

Scriviamo una procedura **Display\_Dato** attraverso la quale un valore numerico compreso fra 0 e 9999 venga inviato sui 4 display dell'interfaccia.

```

{
  Un valore numerico BCD max 4 cifre (da 0000 a 9999)
  Deve essere visualizzato sui 4 display dell'interfaccia;
  dato : integer contiene il valore da visualizzare
}
procedure Display_Dato(dato : word);
  var n,r : integer;
Begin
  If Dato > 9999 Then ShowMessage('Dato troppo elevato')
  Else
    Begin
      N := Dato DIV 1000; // migliaia
      Out_Dato(N, 3);
      R := Dato MOD 1000; // resto migliaia

      N := R DIV 100; // centinaia
      Out_Dato(N, 2);
      R := R MOD 100; // resto centinaia

      N := R DIV 10; // decine
      Out_Dato(N, 1);
      R := R MOD 10; // resto decine

      Out_Dato(R, 0);
    End;
end;

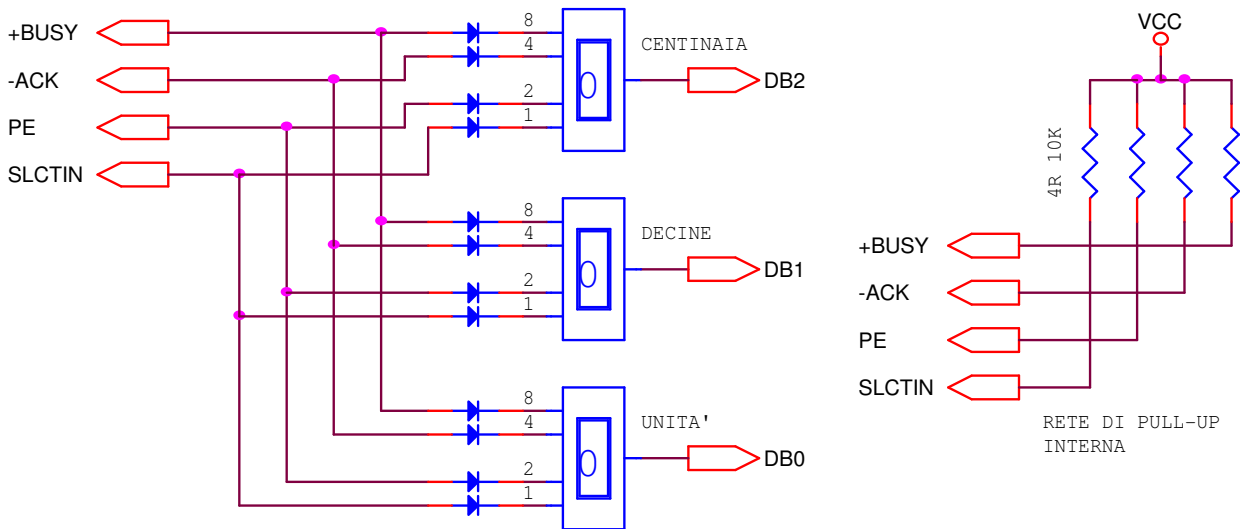
```

N.B.: rifare la procedura in caso si voglia un output esadecimale (con decoder opportuno)

### Applicativo demo per 4 Display decodificati

Terminato il Driver, costituito dalle tre procedure sopra riportate, ora scriviamo un piccolo applicativo che, azzeri la visualizzazione, faccia uscire in sequenza i numeri pari compresi fra 0000 1234 intervallandoli di un ritardo pari a 250 msec. e poi ripeta l'operazione facendo uscire i numeri dispari fra 0001 e 2345. (Da cOncludere)

## Interfaccia per la lettura di 3 Selettori digitali



Nello schema sopra riportato si nota la presenza di 3 selettori digitali (comunemente detti “contraves” dal nome della società che li ha introdotti sul mercato per la prima volta). I selettori sono connessi agli ingressi della porta (8 BUSY, 4 ACK, 2 PE, 1 SLCTIN) mediante 4 diodi al silicio 1N4148 che impediscono conflitti hardware. Ogni selettore viene selezionato da una uscita della porta parallela connessa al “comune” C del selettore digitale: le unità da DB0, le decine da DB1, le centinaia da DB2.

DB2	DB1	DB0	SELEZIONE
1	1	1	NESSUNA
1	1	0	UNITA'
1	0	1	DECINE
0	1	1	CENTINAIA

Le modalità di selezione sono riportate in tabella. Si noti che quando le tre linee DB0, DB1, DB2 sono a livello alto, i diodi risultano tutti interdetti e, grazie alla rete di pull-up interna gli ingressi della porta ricevono 4 livelli alti. Ponendo a livello basso l’uscita DB0, vengono selezionate le unità, ponendo a livello basso l’uscita DB1, vengono selezionate le decine e ponendo a livello basso l’uscita DB2, vengono selezionate le centinaia.

Per spiegare come funziona il circuito poniamo che sia selezionato il selettore delle unità e che su di esso sia impostata la cifra “7” 0111 ciò significa che l’ingresso (8) risulta sconnesso ed il diodo D(8) interdetto mentre gli ingressi (4) (2) e (1) risultano collegati al “comune” C e quindi i Diodi D(4), D(2) e D(1) conducono. Il codice binario letto sulle 4 linee (BUSY, ACK, PE, SLCTIN) è allora 1000. Come si vede il circuito lavora completamente in logica negativa (sia sulla selezione, sia sui dati); questa è tuttavia una scelta obbligata dal momento che internamente alla porta è presente una rete pull-up che non può essere staccata.

Nel prospetto che segue sono riportate, in corrispondenza ad ogni cifra impostata sulla finestrella del “contraves” le connessioni esistenti fra gli ingressi 8-4-2-1 e il terminale “comune C”.

CIFRA	CODICE	PIN CONNESSI A C
0	0000	NESSUNO
1	0001	1
2	0010	2
3	0011	1-2
4	0100	4
5	0101	1-4
6	0110	2-4
7	0111	1-2-4
8	1000	8
9	1010	2-8

### Driver per la gestione dei 3 Selettori digitali

Scriviamo ora tre piccole funzioni che ci consentano di leggere le unità, le decine, le centinaia.

```
{ Legge il valore impostato sul selettore delle Unità }

Function Unita : byte;
  var x : byte;
Begin
  X := PortReadB($378);
  X := X AND $F8; // pongo a 000 I tre bit meno significativi
  X := X OR $06; // Incastro il codice di selezione delle unità
  PortWriteB($378,X);

  X := PortReadB($379); // S7-S6-S5-S4
  X := X XOR $80 ; // S7 viene riportato a Logica Positiva
  X := X SHR 4;

  Result := X;
End;

{ Legge il valore impostato sul selettore delle Decine }

Function Decine : byte;
  var x : byte;
Begin
  X := PortReadB($378);
  X := X AND $F8; // pongo a 000 I tre bit meno significativi
  X := X OR $05; // Incastro il codice di selezione delle decine
  PortWriteB($378,X);

  X := PortReadB($379); // S7-S6-S5-S4
  X := X XOR $80 ; // S7 viene riportato a Logica Positiva
  X := X SHR 4;

  Result := X;
End;

{ Legge il valore impostato sul selettore delle Centinaia }

Function Centinaia : byte;
  var x : byte;
Begin
  X := PortReadB($378);
  X := X AND $F8; // pongo a 000 I tre bit meno significativi
  X := X OR $03; // Incastro il codice di selezione delle centinaia
  PortWriteB($378,X);

  X := PortReadB($379); // S7-S6-S5-S4
  X := X XOR $80 ; // S7 viene riportato a Logica Positiva
  X := X SHR 4;

  Result := X;
End;
```

Scriviamo ora una funzione che ci consenta di leggere il valore numerico impostato sui tre contraves.

```
{ Legge il valore numerico impostato sui tre selettori }
Function Leggo_contraves : integer;

Begin
  Result := Centinaia * 100 + Decine * 10 + Unità;
End;
```

## ESERCIZI MANCANTI:

- 1) GESTIONE 4 DISPLAY CON DECODER LS139
- 2) PILOTAGGIO CONTATORE A 2 CIFRE UP 4518
- 3) PILOTAGGIO CONTATORE A 3 CIFRE UP/DOWN 4029 O 4516
- 4) PILOTAGGIO 74LS259
- 5) LETTORE EPROM CON COUNTER E MPX
- 6) Interfaccia convertitore A/D e D/A

## UTILIZZO DELLA PORTA PARALLELA COME INPUT

Indirizzi dei Registri interni della Porta Parallela della Stampante:

Port	R/W	I/O Addr.	Bits		Function
Data out	W	Base+0	D0-D7	OUT	8 LS TTL
Status In	R	Base+1	S3-S7	IN	5 LS TTL
Control Out	W	Base+2	C0-C3	OUT	4 TTL OPEN COLLECTOR
<i>Control Out</i>	<i>W</i>	<i>Base+2</i>	<i>C4</i>		<i>INTERNAL, IRQ ENABLE</i>
<b>Control Out</b>	<b>W</b>	<b>Base+2</b>	<b>C5</b>		<b>INTERNAL, TRISTATE DATA (*)</b>
<i>Data Feedback</i>	<i>R</i>	<i>Base+0</i>	<i>D0-D7</i>		<i>MATCHES DATA OUT</i>
<i>Control Feedback</i>	<i>R</i>	<i>Base+2</i>	<i>C0-C3</i>		<i>MATCHES CONTROL OUT</i>
<i>Control Feedback</i>	<i>R</i>	<i>Base+2</i>	<i>C4</i>		<i>INTERNAL, IRQ ENABLE REEDBACK</i>

Il bit C5 del registro di controllo se posto a 0 fa funzionare la porta del PC come output mentre se posto a 1 fa funzionare la porta del PC da INPUT.

Da test condotti su hardware di tipo vecchio (schede madri Pentium/AMDK6) con montato Win98SE la porta Data Out funziona anche da ingresso a condizione che nel BIOS sia stata settata come SPP (Standard Parallel Port) oppure come EPP oppure ECP + EPP.

## Appendice B: Connettore standard DB25F

Interfaccia Parallela PC/XT/AT			
DB25-F	Signal	I/O	Reg.Bit
1	-STROBE	OUT	C0-
2	DATA-0	OUT	D0
3	DATA-1	OUT	D1
4	DATA-2	OUT	D2
5	DATA-3	OUT	D3
6	DATA-4	OUT	D4
7	DATA-5	OUT	D5
8	DATA-6	OUT	D6
9	DATA-7	OUT	D7
10	-ACK	IN	S6+
11	+BUSY	IN	S7-
12	+PaperEnd	IN	S5+
13	+SELCTIN	IN	S4+
14	-AutoFd	OUT	C1-
15	-Error	IN	S3+
16	-Init	OUT	C2+
17	-Select	OUT	C3-
18	GND		
19	GND		
20	GND		
21	GND		
22	GND		
23	GND		
24	GND		
25	GND		

Nome	Indirizzo BASE
LPT1	\$378
LPT2	\$278

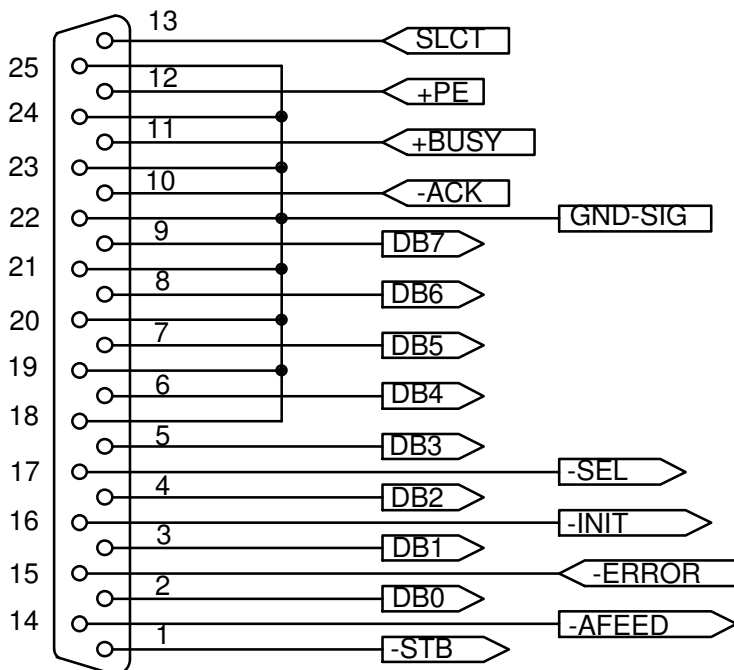
### Indirizzi dei Registri interni della Porta Parallela della Stampante:

Port	R/W	I/O Addr.	Bits		Function
Data out	W	Base+0	D0-D7	OUT	8 LS TTL
Status In	R	Base+1	S3-S7	IN	5 LS TTL
Control Out	W	Base+2	C0-C3	OUT	4 TTL OPEN COLLECTOR
<i>Control Out</i>	<i>W</i>	<i>Base+2</i>	<i>C4</i>		<i>INTERNAL, IRQ ENABLE</i>
<i>Control Out</i>	<i>W</i>	<i>Base+2</i>	<i>C5</i>		<i>INTERNAL, TRISTATE DATA (*)</i>
<i>Data Feedback</i>	<i>R</i>	<i>Base+0</i>	<i>D0-D7</i>		<i>MATCHES DATA OUT</i>
<i>Control Feedback</i>	<i>R</i>	<i>Base+2</i>	<i>C0-C3</i>		<i>MATCHES CONTROL OUT</i>
<i>Control Feedback</i>	<i>R</i>	<i>Base+2</i>	<i>C4</i>		<i>INTERNAL, IRQ ENABLE REEDBACK</i>

## Interfaccia Parallela lato PC

I/O	DB25 PIN	CENT PIN	NAME SIGNAL	OF REGISTER BIT	FUNCTION
O	1	1	-STROBE	C0-	Invia un segnale a l.l. basso di durata >0,5 us
O	2	2	D0	D0	Bit meno significativo
O	3	3	D1	D1	..
O	4	4	D2	D2	..
O	5	5	D3	D3	..
O	6	6	D4	D4	..
O	7	7	D5	D5	..
O	8	8	D6	D6	..
O	9	9	D7	D7	Bit piu' significativo
I	10	10	-ACK	S6+	Impulso d'IRQ basso di ~ 0,5us dopo accettazione
I	11	11	+BUSY	S7-	Alto per Busy/Offline/ Error
I	12	12	+PAPER END	S5+	Alto per fine carta
I	13	13	+SLCT IN	S4+	Alto per selezione stampante
O	14	14	-AUTOFEED	C1-	Basso per a capo linea automatico
I	15	32	-ERROR	S3+	Basso per Error/Offline/Fine carta
O	16	31	-INIT	C2+	Genera impulso basso >50us per inizializzare
O	17	36	-SELECT	C3-	Basso per selezione stampante
//	18-25	19-30	GND		
		33,17,16	GND		

### CONNETTORE PARALLELA LATO PC



DB25F - 25 POLI Femmina

D	C	B	A
8	4	2	1
0	0	0	0
0	0	0	1
0	0	1	0
0	0	1	1
0	1	0	0
0	1	0	1
0	1	1	0
0	1	1	1
1	0	0	0
1	0	0	1
1	0	1	0
1	0	1	1
1	1	0	0
1	1	0	1
1	1	1	0
1	1	1	1

0	CODIFICA - BCD	\$0	CODIFICA ESADECIMALE
1		\$1	
2		\$2	
3		\$3	
4		\$4	
5		\$5	
6		\$6	
7		\$7	
8		\$8	
9		\$9	
10	\$A		
11	\$B		
12	\$C		
13	\$D		
14	\$E		
15	\$F		