Interfacciamento tra PC/XT/AT e sistema di sviluppo Motorola M6809SBC (Microlab sas) progetto HDW e SFTW

(rel.1999)

Ing. Cleto Azzani

INTRODUZIONE ALLE PROBLEMATICHE 3
Idee per lo sviluppo software : *4*
HANDSHAKE o "stretta di mano" *10*
DEFINIZIONE DEI SEGNALI DI CONTROLLO RELATIVI ALLA MODALITÀ "STROBED INPUT" 11
STB (Strobe Input) *11*
IBF (Input Buffer Full F/F) *11*
INTR (Interrupt Request) *11*
INTE A *11*
INTE B *11*
OUTPUT CONTROL SIGNAL DEFINITION 11
***OBF** - Output Buffer Full F/F).* *11*
***ACK** - Acknowledge Input).* *12*
***INTR** - (Interrupt Request).* *12*
INTE A *12*
INTE B *12*
SEZIONE ASSEMBLER MC6809 13
SEZIONE PASCAL OMEGASOFT MC6809 20
SEZIONE TURBO PASCAL (LATO PC/XT/AT) 30

Introduzione alle problematiche

Il sistema di sviluppo M6809SBC progettato da Microlab sas prima che venisse introdotto sul mercato Italiano il PC IBM deriva la sua architettura dai più noti e diffusi sistemi EXORciser M6800 e M6809 ed EXORset M6809 prodotti dalla Divisione Microcomputer di Motorola.

M6809SBC nasce governato dalla CPU MC6809 (Motorola, Hitachi, Thomson) è dotato di ben 56KRAM statica 6K EPROM (Monitor debugger in codice macchina, gestione delle I/O, gestione emulatore, gestione del FDC, etc.) 2 canali seriali RS232 programmabili software ma con velocità settabile solo in modo Hardware (jumper) ; una porta parallela per stampanti con interfaccia Centronics. La memoria di massa che nel sistema EXORciser è costituita da due unità floppy da 8" con capacità da 250KB a 500KB, nel sistema EXORset 30 da due floppy da 5.25" SS40 con capacità paria 80KB ; nel sistema Microlab M6809SBC raggiunge il valore di 460KB con due unità floppy FD da 3.5" DD MFM a 128 Bytes/settore.

Tipo	Cilindri	Testine	Settori	Tot.Sett.	Bytes/sett.	Registr.	Capacità	KB
8" SS	77	1	26	2002	128	FM128	256256	250,25
8" DS	77	2	26	4004	128	FM128	512512	500,5
5.25" SS 40	40	1	16	640	128	FM128	81920	80
5.25" DS 40	40	2	16	1280	128	FM128	163840	160
5.25" SS 80	80	1	16	1280	128	FM128	163840	160
5.25" DS 80	80	2	16	2560	128	FM128	327680	320
3.5" SS 80	80	1	16	1280	128	FM128	163840	160
3.5" DS 80	80	2	16	2560	128	FM128	327680	320
Standard a "Doppia Densità"								
5.25" SS 80	80	1	16	1280	256	MFM256	327680	320
5.25" DS 80	80	2	16	2560	256	MFM256	655360	640
3.5" SS 80	80	1	16	1280	256	MFM256	327680	320
3.5" DS 80	80	2	16	2560	256	MFM256	655360	640
Standard Microlab per M6809SBC								
5.25" SS 80	80	1	23	1840	128	MFM128	235520	230
5.25" DS 80	80	2	23	3680	128	MFM128	471040	460
3.5" SS 80	80	1	23	1840	128	MFM128	235520	230
3.5" DS 80	**80**	**2**	**23**	**3680**	**128**	**MFM128**	**471040**	**460**

Lo standard adottato (ultima riga tabella precedente) consente di coniugare la necessità di ottenere il maggiore spazio su disco possibile mantenendo però un formato settore compatibile con quello previsto dai sistemi operativi MDOS (sia M6800 che M6809) e XDOS pari a 128 bytes/settore.

Ora si pone la necessità di trasferire tutto il software di sviluppo o quello sviluppato (sorgente e OBJ code) su PC per necessità di archiviazione definitiva su CD-ROM.

Non è possibile leggere direttamente i dischetti formattati con standard Microlab su PC (almeno io non ci sono riuscito) per cui si pone il problema di effettuare un trasferimento sicuro e veloce fra sistema M6809DS e PC. Ciò è possibile attraverso una interfaccia parallela e bidirezionale. Il sistema M6809DS è dotato di due canali uno per la trasmissione dati da M6809 verso PC ed uno

indipendente per il trasferimento dati in senso opposto. La soluzione Hardware per quanto concerne il PC può essere quella di seguito riportata che fa uso di una scheda LX833 sulla quale viene montato un chip dedicato ad interfacciamenti di tipo parallelo : l'82C55. Connettendo il PORTA del sistema M6809 con il PORTB dell'82C55 e il PORTA dell'82C55 con il PORTB del sistema M6809 ; connettendo opportunamente i segnali di controllo per consentire un corretto protocollo Handshake fra i due sistemi è possibile trasferire in modo asincrono dati ad elevata velocità fra i due sistemi.

Si pone poi il problema di corredare i due sistemi di adatto software che consenta una comunicazione efficace semplice . Sul PC si può sviluppare il pacchetto in Turbo Pascal Borland ; su sistema M6809 si può analogamente sviluppare in Omegasoft Pascal o in Assembler.

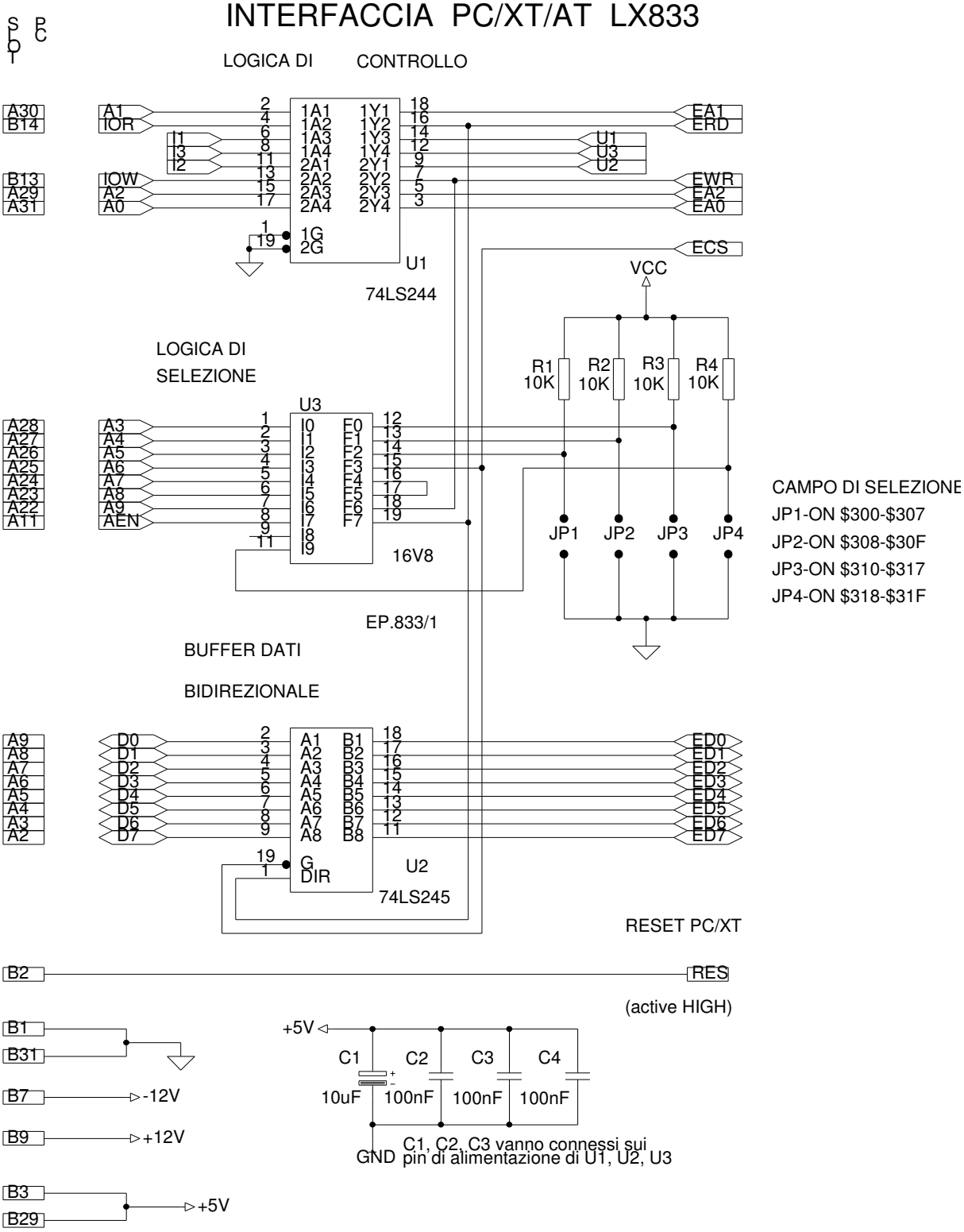
Idee per lo sviluppo software :

a) Tutte le funzioni primitive che operano su disco floppy su M6809 (quelle della EPROM FDC) devono essere accessibili da PC ; così facendo si potrà da PC leggere e/o scrivere uno o più settori su disco. A questo punto si può pensare di "copiare" da M6809 a PC il contenuto di ciascun disco (8", 5.25", 3.5") su di un file binario. Su PC si può poi scrivere un semplice emulatore XDOS che estragga dal file binario i files contenuti dal dischetto.

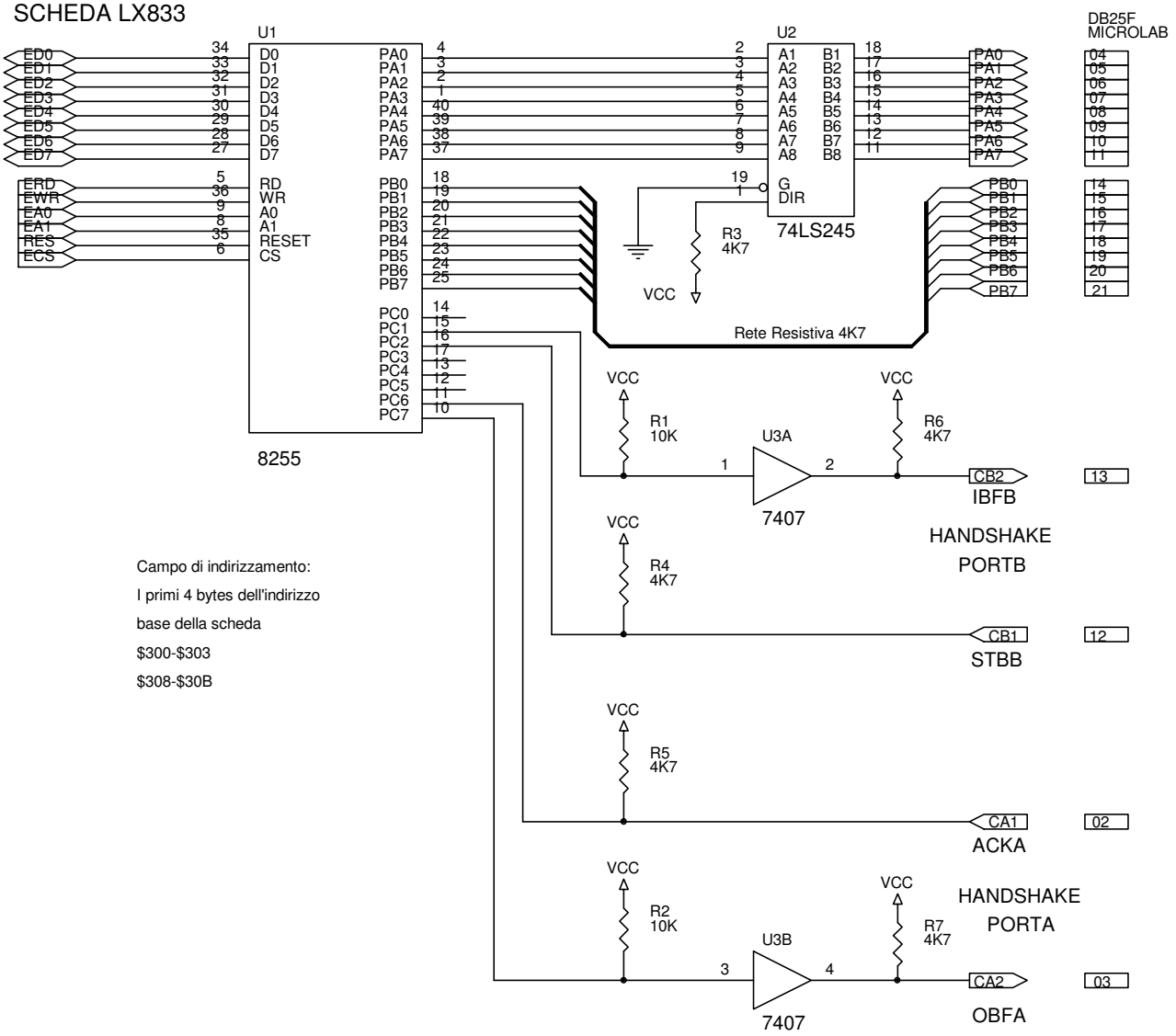
b) Su Pc devono essere disponibili tutte quelle funzioni che consentono di analizzare il contenuto di un dischetto : DIR ; DUMP ; LIST ; COPY ; etc. Da PC si può quindi ordinare il trasferimento dati da M6809 a PC ma basandosi sui files presenti sul dischetto. Non ci dovrebbero essere problemi per i files ASCII (text files) ; potrebbero esserci problemi per i files binari che in XDOS hanno la necessità di avere memorizzato lo Start Loading Address, l'End Loading e lo Start Execution (PSN $FFFF) e forse per i files *.LX (relocatable object).

c) Il PC potrebbe funzionare da HD nei confronti del sistema M6809. Definiti quindi 4 files binari dell'ampiezza esatta pari a un drive FD da 460K (FD3 DS 80), il sistema M6809 scarica i dati su queste 4 unità vrtuali. In quest'ultimo caso andrebbe riscritta la Eprom del FDC.

INTERFACCIA PC/XT/AT LX833



SCHEDA LX833



Campo di indirizzamento:

I primi 4 bytes dell'indirizzo base della scheda

$300-$303

$308-$30B

Connessioni LPT Motorola M6809SBC

(vaschetta femmina DB25)

PIN	PIA		FUNCTION
1	GND		
2	CA1	IN	ACK
3	CA2	OUT	
4	PA0	OUT	DB0
5	PA1	OUT	DB1
6	PA2	OUT	DB2
7	PA3	OUT	DB3
8	PA4	OUT	DB4
9	PA5	OUT	DB5
10	PA6	OUT	DB6
11	PA7	OUT	DB7
12	CB1	IN	
13	CB2	I/O	
14	PB0	I/O	SELECT
15	PB1	I/O	PE
16	PB2	I/O	BUSY
17	PB3	I/O	
18	PB4	I/O	
19	PB5	I/O	
20	PB6	I/O	
21	PB7	I/O	
22	+5V		
23	GND		
24	GND		
25	GND		

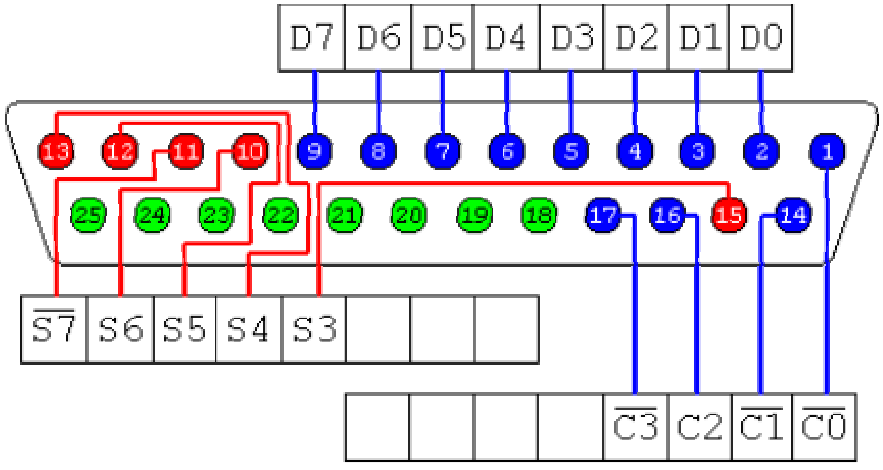
DB25F Connettore

Connessioni LPT IBM PC/XT/AT

(vaschetta femmina DB25)

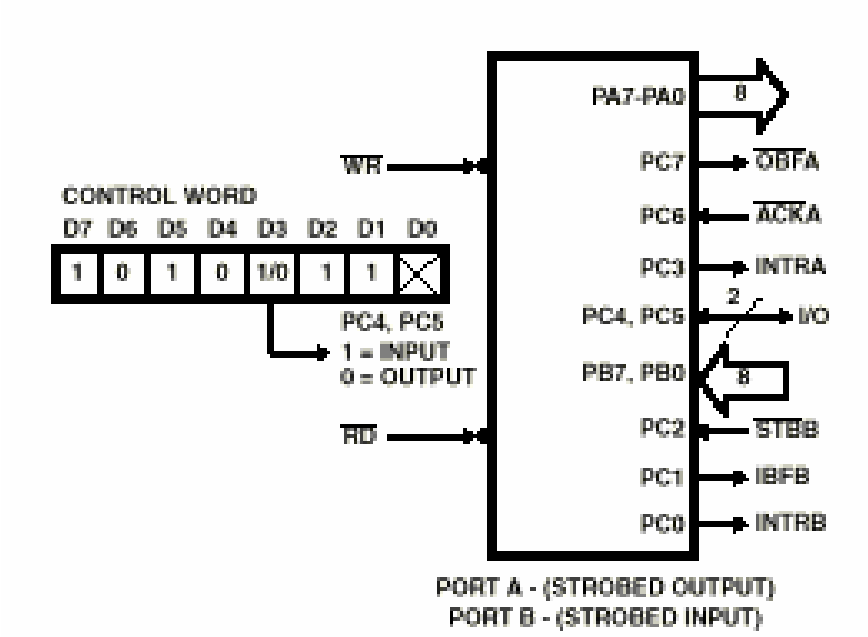
PIN	LPTx		FUNCTION
1	C0-	OUT	-STRB
2	D0	OUT	DB0
3	D1	OUT	DB1
4	D2	OUT	DB2
5	D3	OUT	DB3
6	D4	OUT	DB4
7	D5	OUT	DB5
8	D6	OUT	DB6
9	D7	OUT	DB7
10	S6+	IN	-ACK
11	S7-	IN	BUSY
12	S5+	IN	PAPER
13	S4+	IN	SELIN
14	C1-	OUT	-AFEED
15	S3+	IN	-ERROR
16	C2+	OUT	-INIT
17	C3-	OUT	-SEL
18	GND		
19	GND		
20	GND		
21	GND		
22	GND		
23	GND		
24	GND		
25	GND		

DB25F Connettore



82C55A BASIC OPERATION

A1	A0	RD	WR	CS	MODE	OPERATION
0	0	0	1	0	READ	PortA --> Data Bus
0	1	0	1	0	READ	PortB --> Data Bus
1	0	0	1	0	READ	PortC --> Data Bus
1	1	0	1	0	READ	Control Word --> Data Bus
0	0	1	0	0	WRITE	Data Bus --> PortA
0	1	0	0	0	WRITE	Data Bus --> PortB
1	0	0	0	0	WRITE	Data Bus --> PortC
1	1	0	0	0	WRITE	Data Bus --> Control
X	X	X	X	1	DISABLE	Data Bus Three State
X	X	1	1	0	DISABLE	Data Bus Three State



Nella “modalità 1” (MODE 1 o Strobed I/O) il circuito integrato consente di effettuare operazioni di I/O in tecnica “handshake” attraverso i due port (A e B) ; le linee del Port C vengono utilizzate per generare o ricevere i segnali di “handshake”.

Nella figura a fianco è riportato, un esempio di impiego dell’8255 in cui il PortA funziona da uscita, il Port B da ingresso le linee PC7 (Strobe) e PC6 (Acknoledge) realizzano l’hanshake per il Port A, le linee PC1 (STB) e PC2 (ACK) l’hanshake per il Port B.

Sono disponibili pure uscite di Interrupt : PC3 (INTRA) e PC0 (INTRB) attraverso le quali si può pilotare un apposita logica di generazione di richieste di “interrupt da inoltrare alla CPU.

HANDSHAKE o "stretta di mano"

Quando due dispositivi asincroni devono trasferire tra loro informazioni attraverso una porta di comunicazione (di tipo parallelo 4 o 8 bit), per sincronizzare il trasferimento dati utilizzano il metodo handshake o della "stretta di mano". Il dispositivo che trasmette il dato lo colloca sulla porta di comunicazione ; poi fornisce su una linea di controllo ausiliaria, un segnale cosiddetto di "strobe" o di "conferma dato" (in genere un impulso attivo a livello logico basso della durata approssimativa di 1 μs). La linea di controllo distinta dalle linee dati e denominata linea di "strobe" è uscita nel circuito trasmettitore e ingresso nel ricevitore. Il ricevitore avvertito della presenza del dato sulla porta lo legge, lo trasferisce "al sicuro" e ad operazione ultimata conferma con una segnalazione di Data Acknowledge che il dato è stato correttamente acquisito. La segnalazione Data Acknowledge viene inoltrata dal circuito ricevitore che la genera, al circuito trasmettitore che la riceve attraverso una apposita linea di controllo indipendente dalle linee dati. Il trasmettitore può quindi mettere così a disposizione del ricevitore altri dati che vengono sempre trasferiti al ricevitore con questa tecnica della doppia conferma denominata handshake.

Un classico esempio di "handshake" si incontra nello studio del modo di funzionamento della interfaccia Centronics che sta alla base delle stampanti connesse ai PC con interfacciamento parallelo. Il trasmettitore è il PC o meglio l'interfaccia parallela del PC, il ricevitore è la stampante.

Modi di funzionamento

Mode 1 - (Strobed Input/Output). Questa configurazione funzionale fornisce un metodo per trasferire in modalità I/O dati verso o da una porta specifica unitamente a segnali di strobe secondo una tecnica denominata "hamd-shake"

Nella "modalità 1" (MODE 1 o Strobed I/O) il circuito integrato consente di effettuare operazioni di I/O in tecnica "handshake" attraverso i due port (A e B) ; le linee del Port C vengono utilizzate per generare o ricevere i segnali di "handshake".

Definizioni Base per la Modalità 1 :

- Due Gruppi (Gruppo A e Gruppo B)
- Ciascun gruppo è costituito da 8 linee dati (porta ad 8 bit) e 4 linee di controllo;
- Le porte ad 8 bit possono essere programmate sia da Input che da Output; in entrambe le modalità i dati vengono memorizzati.
- Le quattro linee di controllo vengono utilizzate per controllare e monitorare lo stato della porta dati da 8 bit.

Definizione dei segnali di controllo relativi alla modalità “strobed input"

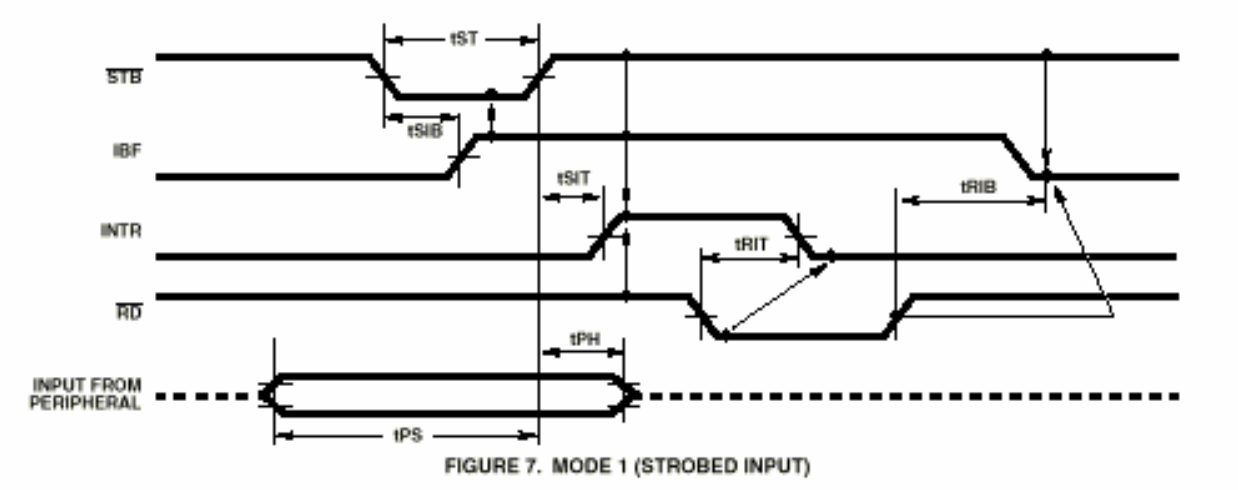


FIGURE 7. MODE 1 (STROBED INPUT)

STB (Strobe Input)

Un livello basso su questo ingresso carica il dato all’interno del “latch” di ingresso.

IBF (Input Buffer Full F/F)

Un livello “alto” su questa uscita, indica che il dato è stato caricato all’interno del “latch” di ingresso : in sostanza si tratta di un segnale di “acknowledge”. IBF viene settato dal passaggio a “0” del segnale STB e viene “resettato” dal fronte di salita del segnale RD se l’operazione READ viene effettuata sulla porta di ingresso su cui è giunto il dato.

INTR (Interrupt Request)

Un livello “alto” su questa uscita può essere utilizzato per inoltrare una richiesta di “interrupt” alla CPU quando un dispositivo di ingresso ravvisa tale necessità (requesting service). INTR viene “settato” dalla condizione: STB a livello “alto”, IBF a livello “alto” e INTE a livello “alto”. INTR viene “resettato” dal fronte di discesa del segnale RD. Questa procedura consente ad un dispositivo di input di richiedere un servizio alla CPU semplicemente inviando in modalità “strobed” i suoi dati alla porta di ingresso.

INTE A

Controlled by bit set/reset of PC4.

INTE B

Controlled by bit set/reset of PC2.

Output Control Signal Definition

OBF - Output Buffer Full F/F).

L'uscita OBF si porta a livello basso per indicare che la CPU ha effettuato una operazione di scrittura sulla porta dati corrispondente.
I dati sono da considerare validi solamente sul fronte di salita del segnale OBF. Il flip flop associato all'uscita OBF viene "settato" dal fronte di salita del segnale esterno WR (Write) e "resettato" dal passaggio a 0 del segnale esterno ACK .

ACK - Acknowledge Input).

Un livello basso ("0") su questo ingresso comunica al chip 82C55A che la periferica esterna ha acquisito correttamente il dato precedentemente trasmesso ed è quindi è pronta a ricevere un altro dato.

INTR - (Interrupt Request).

Un livello alto ("1") su questa uscita può venire utilizzato per inviare una richiesta di interrupt alla CPU proprio nel momento in cui il dispositivo esterno ha correttamente acquisito il dato trasmesso dalla CPU. L'uscita INTR viene "settata" quando ACK, OBF e INTE si trovano tutti a livello alto; viene "resettata" sul fronte di discesa del segnale WR.

INTE A

Controlled by Bit Set/Reset of PC6.

INTE B

Controlled by Bit Set/Reset of PC2.

NOTE: To strobe data into the peripheral device, the user must operate the strobe line in a hand shaking mode. The user needs to send OBF to the peripheral device, generates an ACK from the peripheral device and then latch data into the peripheral device on the rising edge of OBF.

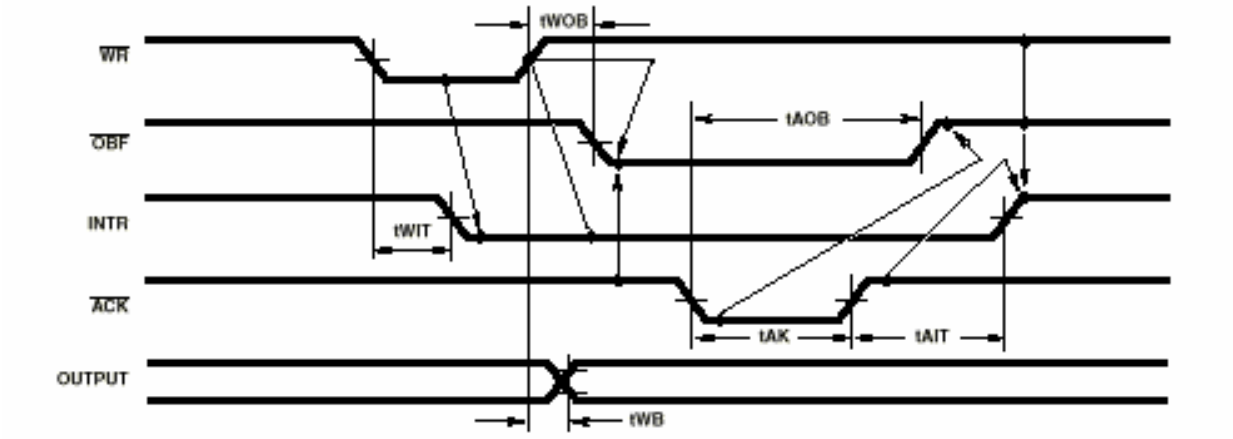


FIGURE 9. MODE 1 (STROBED OUTPUT)

SEZIONE ASSEMBLER MC6809

```
NAM     SYSASM    file  ASM.SA

        TTL     Assembler M6809 Interface with Pascal

*
*       MSDOS/XDOS System Call from Pascal M6809 Omegasoft
*
*       procedure syscall (fcode : byte ; var regs : regtyp);
*
*        regtyp  =  record
*          cc, a, b            : byte;
*            x                 : hex;
*       end;
*
*       La procedura passa i valori di inizializzazione dei registri della CPU
*       definiti nel sistema operativo, attraverso la variabile regs e utilizzando
*       la chiamata individuata da fcode. Al rientro viene aggiornato il valore dei
*       registri contenuti nel record.
*

     XDEF    SYSC

SYSC    EQU     *
        PSHS    CC,U    Salvataggio CC, U
        LDA     2,U     Recupero da stack U fcode
        LDB     #$39    Codice RTS
        PSHS    D       Salvo nello stack S prima B poi A
        LDA     #$3F    Codice SWI
        PSHS    A       Salvo nello stack S codice SWI
*       a questo punto nello Stack puntato da S mi trovo:
*    S ----->       SWI
*                fcode
*                RTS
        LDU     0,U     Recupero indirizzo del record Registri
        LDD     1,U     Recupero il valore dei registri A e B
        LDX     3,U     Recupero il valore di X
        JSR     0,S     Eseguo la subroutine costruita nello stack S
        STD     1,U
        STX     3,U
        TFR     CC,A
        STA     0,U     Aggiorno CCR
        LEAS    3,S     Elimino subroutine nello Stack S
        PULS    CC,U    Recupero i registri CC,U
        LEAU    3,U     Rimuovo parametri dallo stack U
        RTS


    XDEF   CRC_BY,B_TEST

*
*       Elaborazione del CRC su un byte
*
*       Tipo_crc= Array [0..2] of char; { buffer deposito Crc }
*
*       Function Crc_byte(Old_crc:tipo_crc ; V:byte) : tipo_crc ; external ;
*
CRC_BY LDA    ,U+    legge byte  v  e posiziona lo stack U
        EORA   ,U
```

```
        PSHS   A
     LDA    1,U
     LDB    2,U
     STA    ,U
     STB    1,U
     CLRA
     LDB    ,S
     ASLB
     ROLA
     EORA   1,U
     STA    1,U
     STB    2,U
     CLRB
     LDA    ,S
     LSRA
     RORB
     LSRA
     RORB
     EORA   1,U
     EORB   2,U
     STA    1,U
     STB    2,U
     LDA    ,S
     ASLA
     EORA   ,S
     STA    ,S
     ASLA
     ASLA
     EORA   ,S
     STA    ,S
     ASLA
     ASLA
     ASLA
     ASLA
     EORA   ,S+
     BPL    CRCBR
     LDA    #$80
     LDB    #$21
     EORA   ,U
     STA    ,U
     EORB   2,U
     STB    2,U
CRCBR  RTS

*
* stringa di test
*
B_TEST FCB $55,$80,$40,$20,$10,$8,$4,$2,1,0,$7F,$BF,$DF,$EF,$F7,$FB
     FCB $FD,$FE,$FF
```

```
*
*   Calcolo il numero complessivo di settori contenuti in un dischetto
*   XDOS : se il drive number e' maggiore di 3 si deve passare sul
*          controller alternativo con la routine XTOGL prima di
*          eseguire la routine
*
*   FUNCTION SecMax(drv:byte):hex; external ;
*
*

      XDEF    SECMAX,STATUS

*
* D I S K     E R O M     E Q U A T E S
*
CURDRV EQU    0       CURRENT DRIVE NUMBER
FDSTAT EQU    8       DISK TRANSFER STATUS
SIDES  EQU    $D      - ->SINGLE; + -> DOUBLE SIDED
CURTRK EQU    $14

*
* E R O M    E N T R Y    P O I N T S
*
CLOCK   EQU     $E887
FDCODE  EQU     $EBFE   Posizione Codice EPROM FDC
COD5P   EQU     $5855
COD8P   EQU     $5043
XTOGL   EQU     $F036    Switch from *E to *A map and viceversa

*
* E R O M     E R R O R     E Q U A T E S
*
ER$CRC EQU    '1      DATA CRC ERROR
ER$WRT EQU    '2      WRITE PROTECTED DISK
ER$RDY EQU    '3      DISK NOT READY
ER$MRK EQU    '4      DELETED DATA MARK ENCOUNTERED
ER$TIM EQU    '5      TIMEOUT
ER$DAD EQU    '6      INVALID DISK ADDRESS
ER$SEK EQU    '7      SEEK ERROR
ER$DMA EQU    '8      DATA ADDRESS MARK ERROR
ER$ACR EQU    '9      ADDRESS MARK CRC ERROR
*-------------------------------------------------
*  Parametri tipo disco
*-------------------------------------------------
D5     EQU    %01000000 drive a 3 o 5 pollici
D8     EQU    %00000000 drive a 8 pollici
HD     EQU    %00010000 indica Hard-disk
FM     EQU    %00100000 singola densita'
MFM    EQU    %00000000 doppia densita'
SINST  EQU    %00000000 disco single step (solo per 3 o 5 pollici)
DOUST  EQU    %10000000 disco doppio step (solo per 3 o 5 pollici)
DSIDE  EQU    %00000000 Disco a doppia faccia 5 pollici
SSIDE  EQU    %00000100 Disco a singola faccia 5 pollici
SRAT3  EQU    0       Stepping-rate 3/6 milli-sec
SRAT6  EQU    1       Stepping-rate 6/12 milli-sec
SRAT10 EQU    2       Stepping-rate 10/20 milli-sec
SRAT15 EQU    3       Stepping-rate 15/30 milli-sec

*------------------------------
* Parametri drive corrente
```

```
*-----------------------------
FREQ   EQU     $1A    CLOCK FREQUENCY FLAG
RATE   EQU     $1B    STEPPING RATE
DTYPE  EQU     $1C    TIPO DI DISCHETTO
SCTRK  EQU     $1D    SETTORI PER TRACCIA
MAXSEC EQU     $1E    MASSIMO NUMERO DI SETTORI

*
*  Sezione del Codice
*
SECMAX  EQU    *
      PSHS    D      Salvo prima D
      PSHS    U      poi salvo U
      LDA     0,U    Recupero da stack U drv
      ANDA    #$0F   Converto da ASCII a hex
      STA     DRBUF  memorizzo drive
      CMPA    #3     confronto con 3
      BLS     SEC1   se minore o uguale controller corrente

      PSHS    A      salvo A
      JSR     XTOGL  accedo mappa alternativa
      PULS    A      recupero A

      SUBA    #4     (drive 4 corrisponde a 0 nella mappa *A)

SEC1    STA     CURDRV
      LDD     FDCODE  leggo codice Eprom FDC
      CMPD    #COD8P  Codice Eprom XDOS 8"  ?
      BEQ     FDPS3
      CMPD    #COD5P  Codice Eprom WD 5" o 3.5" ?
      BEQ     FDPS4

*
* Controller Sconosciuto :  errore nella EPROM del FDC
*
FDPER   LDA    #'ER$RDY    no standard FDC drive non pronto
      STA     STATUS,PCR  segnala errore
      BRA     FDPA7

*
* Controller WD per FD da 3.5" o 5.25"
*
FDPS4   JSR    CLOCK
      LDA     FDSTAT
      STA     STATUS,PCR
      BRA     FDPA7

*
* Controller Motorola per FD a 8" FDC3 XDOS
*
FDPS3   LDA    #77     cilindri in caso di 8"
      STA     CURTRK

      LDA     #SRAT6+D8+FM+SSIDE
      STA     DTYPE

      JSR    CLOCK
      LDA     FDSTAT
      STA     STATUS,PCR
```

```
FDPA6   LDD    #2000    Total sct SS 8"
        TST    SIDES    - single + double
        BMI    FDPA5
        LDD    #4004    Total sct DS 8"

FDPA5   STD    MAXSEC
        LDA    #26
        STA    SCTRK
        LDA    DTYPE
        ANDA   #$FF-SSIDE
        STA    DTYPE

*
* Uscita dalla Routine
*
FDPA7   PULS   U        Recupero U
        LEAU   1,U      Rimuovo parametro dallo stack U

        LDA    STATUS
        CMPA   #ER$RDY  Verifico Status
        BEQ    FDPA8

        LDD    MAXSEC
FDPA9   PSHU   D

        LDA    DRBUF
        CMPA   #3
        BLS    FDPAA
        PSHS   A
        JSR    XTOGL
        PULS   A

FDPAA   EQU    *
        PULS   D,PC

FDPA8   LDD    #0       In caso di errore azzero MAXSEC
        BRA    FDPA9

STATUS  RMB    1
DRBUF   RMB    1


CKBRK   EQU    $F045

        XDEF   BREAK

*
* Break Routine
*
BREAK   EQU    *
        PSHS   A
        CLRA   Valore corrispondente a False
        JSR    CKBRK
        BCC    BRK1
        INCA   Valore corrispondente a True
BRK1    PSHU   A
        PULS   A,PC

*
* Definizioni associate alla Porta Parallela LPT
```

```
*
PORTA   EQU     $EFC0
CRA     EQU     PORTA+1
PORTB   EQU     CRA+1
CRB     EQU     PORTB+1

*
* Trasmette Byte su PortA parallelo 8 bit
* Il byte da trasmettere viene collocato sul PORT-A
* si procede alla generazione del data strobe : impulso su CA2 a LLB
* si attende su CA1 il segnale di ACK da parte del ricevitore .

* NB. non sono stati attivati controlli di time-out

        XDEF    TXBY

TXBY    PSHS    D
        LDA     ,U+     Recupero Dato e sistemo Stack U (Pascal)
        STA     PORTA
* Strobe on CA2
        LDA     CRA
        ANDA    #%11110111
        STA     CRA
        ORA     #%00001000
        STA     CRA
* ACK from CA1 ?
TXBLP   TST     CRA     ACK  from receiver ?
        BPL     TXBLP
        TST     PORTA   Reset IRQ flag
        PULS    D,PC

*
* Riceve Byte su PortB parallelo 8 bit
* Si attende il data strobe da parte del trasmettitore connesso a CB1;
* Il dato viene quindi letto dal PORTB;
* si procede alla generazione del segnale di ACK impulso a LLB su CB2.
*
* NB. non sono stati attivati controlli di time-out }
*
        XDEF    RXBY

RXBY    PSHS    D
* Attesa Strobe da TX
RXBL    TST     CRB
        BPL     RXBL
        LDA     PORTB   Leggo il dato
        PSHU    A       Sistemo nello stack U per Pascal
* Strobe on CB2
        LDB     CRB
        ANDB    #%11110111
        STB     CRB
        ORB     #%00001000
        STB     CRB
        PULS    D,PC

       XDEF   CRCB,CRCBF

CRCBF   RMB     3

*
```

```
*  Elaborazione del CRC su un byte
*
*  Procedure Crcb(V:byte) ; external ;
*
CRCB   LDA    ,U+    legge byte  v  e posiziona lo stack U
       EORA   CRCBF
       PSHS   A
       LDA    CRCBF+1
       LDB    CRCBF+2
       STA    CRCBF
       STB    CRCBF+1
       CLRA
       LDB    ,S
       ASLB
       ROLA
       EORA   CRCBF+1
       STA    CRCBF+1
       STB    CRCBF+2
       CLRB
       LDA    ,S
       LSRA
       RORB
       LSRA
       RORB
       EORA   CRCBF+1
       EORB   CRCBF+2
       STA    CRCBF+1
       STB    CRCBF+2
       LDA    ,S
       ASLA
       EORA   ,S
       STA    ,S
       ASLA
       ASLA
       EORA   ,S
       STA    ,S
       ASLA
       ASLA
       ASLA
       ASLA
       EORA   ,S+
       BPL    CRCR
       LDA    #$80
       LDB    #$21
       EORA   CRCBF
       STA    CRCBF
       EORB   CRCBF+2
       STB    CRCBF+2
CRCR   RTS

       END
```

SEZIONE PASCAL OMEGASOFT MC6809

File PGPASCAL.SA

{ Programma per la gestione del trasferimento veloce, attraverso porta parallela dei dati presenti su FD 3.5", 5.25", 8" formato EXORciser EXORset M6809, o M6809DS Microlab sas.
Il programma e' redatto in Pascal Omegasoft M6809 ed e' corredato di Procedure e Funzioni predisposte in Assembler M6809. I dati trasmessi su link parallelo ad 8 bit vengono ricevuti da un PC che e' corredato di software strutturato in modo analogo.
Lo scopo fondamentale del lavoro e' quello di archiviare su CDROM tutto il lavoro software fatto in ambiente Microlab sas.

Copyright by Cleto Azzani 31 luglio 1997 Microlab sas }

```
Program PGpascal(input,output) ;

 Type
  regtyp  =  record
              cc, a, b  : byte;
              x         : hex;
             end;

  packetyp = record
              status : byte ;
              sector : hex ;
              ad_buf : hex ;
              nsect  : hex ;
             end ;

  disktyp = record
              name : string[16] ;
              date : string[10] ;
              user : string[20] ;
              apps : string[16] ;
              sec  : hex ;
            end;

  Tipo_crc= Array [0..2] of char; { buffer deposito Crc }

 CONST
    ver = '7.00'   ;  { versione software }
    mered  = #$3d  ; { lettura settori multipli da disco }
    mewrt  = #$3e  ; { scrittura settori multipli su disco }
    bmax   = 250   ;  { blocco settori in lettura }
    scsize = 128   ;  { capienza di un settore }
    maxbuf = bmax*scsize - 1 ; { massimo indice elementi in buffer 128*128 -1 }
    ACK    =  Chr(6);
    NACK   =  Chr(21);
    BS     =  #8;
    BEL    =  #7;
```

```
    Readsc =  $E869 ;
    ReadPs =  $E86D ;
    RdCRC  =  $E86F ;
    Restor =  $E875 ;
    WritSc =  $E884 ;
    Retry  =  5;

  VAR              { Variabili  Globali }

    PortA    : byte at $EFC0 ;
    ControlA : byte at $EFC1 ;
    PortB    : byte at $EFC2 ;
    ControlB : byte at $EFC3 ;

    Curdrv   : byte at $0 ;  { Current Drive Number }
    STRSCT   : hex  at $1 ;  { Starting PSN }
    NUMSCT   : hex  at $3 ;  { Number of sectors to process }
    LSCTLN   : byte at $5 ;  { Number of bytes to load from last Sector }
    CURADR   : hex  at $6 ;  { Current Load Address }
    FDSTAT   : byte at $8 ;  { Disk Status upon return }

    drive : char ;
    fderror : byte ;
    sect, sect1, num, res  : hex;
    block  : integer ;  { blocco settori in lettura }
    I, J, K, N, x : integer ;
    s1,s2,s3 : string;
    regs  : regtyp ;
    packet : packetyp ;
    disk : disktyp ;
    Buffer : Array[0..maxbuf] of byte ;
    Crc : Tipo_crc;  { Deposito crc routine 1 }
    flack : boolean ;

    B_test : Array [0..18] of byte pcr; { stringa di test }
    status : byte  pcr ;{ for XREF }
    CRCBF : Tipo_crc  pcr ; { Deposito CRC routine 2 }

  { PROCEDURE E FUNZIONI ESTERNE stese in Assembler M6809 }

   PROCEDURE Txby(d:byte); external ;  { Trasmissione byte da ASM09 }

   FUNCTION RXBY : byte ; external ;   { Ricezione byte da ASM09 }

   PROCEDURE Crcb(V:byte) ; external ; { Altra Routine per calcolo CRC }

   PROCEDURE Sysc(fcode : byte ; var regs : regtyp) ; external ;

   FUNCTION SecMax(drv:byte):hex; external ;
```

```
 FUNCTION BREAK: Boolean ; external ;

 FUNCTION Crc_byte(Old_crc:tipo_crc;V:byte):tipo_crc;external;
{ calcola il crc di un byte }

{ PROCEDURE E FUNZIONI INTERNE stese in Pascal M6809 }

   Function Bin(i : hex) : String[16];
  { return bin representation of word or byte }
  var
    j : hex;
    h,k : Byte;
  begin
    j := $8000 ;
    K := #16 ;
    If ((I and $FF00) = $0) Then
      Begin
        J := $0080 ;
        K := #8 ;
      End;
    For h := #1 to k Do
      Begin
        Bin[h] := '0' ;
        If (i And j) <> $0
          Then
            Begin
              Bin[h] := '1' ;
            End;
        j := j >> $1 ;
      End ;
      Bin[0] := K ;
  End { Bin } ;

 Function RX_ACK : Boolean ;
    Var a : byte;
  Begin
    RX_ACK := False ;
    If Flack Then
      Begin
        a := RXBY;
        If a = ACK Then RX_ACK := True ;
      End;
  End;

{ Azzera eventuali flag di interrupt presenti sulla porta
  parallela prima della trasmissione }
Procedure Init_Port;
  Var a : byte;
  Begin
    ControlA := #$3C;
```

```
   ControlB := #$3C;
   a := PortA;
   a := PortB;
 End;

 Procedure Init_crc;  { inizializza il CRC }
  begin
   CRCBF[0]:=#$FF;
   CRCBF[1]:=#$FF;
   CRCBF[2]:=#$FF;
  end;

{ Trasmissione di una stringa di caratteri
  se code = #0 senza CRC
  se code = #1 con CRC24  }

Procedure Tx_string(data:string;code:byte);
 Var i : integer;
    d : byte;
 Begin
   If code = #1 Then Init_Crc;
   For i:=1 to length(data) Do
    Begin
      d := Chr(ord(data[i]));
      Txby(d);
    {  Crc := Crc_byte(Crc,d); }
      Crcb(d);
    End;
   If code = #1 Then
      Begin
        For I := 0 to 2 Do Txby(Not (Crcbf[I]));
      End;
 End;

 { Trasmetti Buffer e CRC
   'H1'
    Numero di Bytes (2 bytes)
    < Dati >
    CRC24          (3 bytes)                    }

 Procedure TX_Buffer(N : integer);
  Var I : integer;
  Begin

    s3 := concat('H1',chr(N>>8),chr(N));
    Tx_string(s3,#0);
    Init_Crc;
    For I := 0 To N-1 do
      Begin
        { Crc := Crc_byte(Crc,Buffer[I]); }
```

```
      Crcb(Buffer[I]);
      Txby(Buffer[I]);
    End;
  For I := 0 to 2 Do Txby(Not (Crcbf[I]));
 End;

PROCEDURE Read_Sector(drv:byte;SPSN,NSCT:Hex; var error : byte);
 Begin
  StrSct := SPSN;  { Start of PSN to read }
  NumSct := NSCT;  { Number of SCT to read }
  Curadr := Addr(buffer) ;
  ! JSR $E869  ;
  error := fdstat;
 End;

PROCEDURE Write_Sector(drv:byte;SPSN,NSCT:Hex; var error : byte);
 Begin
  StrSct := SPSN;  { Start of PSN to write }
  NumSct := NSCT;  { Number of SCT to write }
  Curadr := Addr(buffer) ;
  ! JSR $E884    ;
  error := fdstat;
 End;

PROCEDURE Read_PSN(drv:byte;SPSN,NSCT:Hex;var error : byte);
 Begin
  Packet.sector := SPSN;  { Start of PSN to read }
  Packet.nsect  := NSCT;  { Number of SCT to read }
  Packet.ad_buf := Addr(buffer) ;
  { Entry della syscall }
  Regs.b := drv - '0' ;
  Regs.x := Addr(Packet) ;
  Sysc(mered,regs) ;
  error := Packet.status ;
  If odd(regs.cc)       { carry set ? }
    Then  Writeln('Read_ PSN-Return status = ',packet.status,Bel);
 End;

PROCEDURE Write_PSN(drv:byte;SPSN,NSCT:Hex; var error : byte);
 Begin
  Packet.sector := SPSN;  { Start of PSN to write }
  Packet.nsect  := NSCT;  { Number of SCT to write }
  Packet.ad_buf := Addr(buffer) ;
  { Entry della syscall }
  Regs.b := drv - '0' ;
  Regs.x := Addr(Packet) ;
  Sysc(mewrt,regs) ;
  error := Packet.status ;
  If odd(regs.cc)       { carry set ? }
    Then  Writeln('Write_ PSN-Return status = ',packet.status,Bel);
```

```
 End;

PROCEDURE Namedisk ; {legge settore 0 ricava nome disco}
Var
 c  : integer ;

begin
  disk.name := '' ;
  disk.date := '' ;
  disk.user := '' ;
  disk.apps := '' ;
  disk.sec  := SecMax(drive) ;
 If disk.sec <> $0 Then
   Begin
    C :=0 ;
    Repeat
      c := c + 1 ;
     Read_Sector(drive,$0,$1,fderror) ; { Read_PSN legge settore 0 dati dischetto}
          {  Testare errore fderror  }
    Until (fderror = #$30) Or (c > 5) ;
    If (C > 5) Then
      Begin
        Writeln(' Fatal Error in Read_Sector ',fderror,Bel);
        Txby('H');
        Txby('9');
        If Rx_ack Then Write(' ACK OK ');
      End
     Else
      Begin
        If ((Buffer[31] =#0) And (Buffer[30] = #0)) Then
          Begin
            For c := 1 to 16 do disk.name[c] := Buffer[c-1];
            disk.name[0] := char(16);

            For c := 1 to 10 do disk.date[c] := Buffer[c+20-1];
            disk.date[0] := char(10);

            For c := 1 to 16 do disk.user[c] := Buffer[c+48-1];
            disk.user[0] := char(16);

            For c := 1 to 16 do disk.apps[c] := Buffer[c+32-1];
            disk.apps[0] := char(16);
          end
         Else
          Begin
            For c := 1 to 8 do disk.name[c] := Buffer[c-1];
            disk.name[0] := char(8);

            For c := 1 to 6 do disk.date[c] := Buffer[c+12-1];
            disk.date[0] := char(6);
```

```
            For c := 1 to 20 do disk.user[c] := Buffer[c+18-1];
            disk.user[0] := char(20);

            disk.apps :='';
           end;
     End;
   End ;

End ;

Procedure TX_h0;
  var h : char;
 Begin

   { Definisco i parametri del packet }
   s1 := '';
   Namedisk;   { individua parametri disco }
   Writeln('D.Name .......:  ',disk.name);
   s1 := concat(s1,disk.name,';') ;
   Writeln('D.Date .......:  ',disk.date);
   s1 := concat(s1,disk.date,';') ;
   Writeln('D.User .......:  ',disk.user);
   s1 := concat(s1,disk.user,';') ;
   Writeln('D.Apps .......:  ',disk.apps);
   s1 := concat(s1,disk.apps,';') ;
   Writeln('Sct :',Integer(disk.sec),'=',Integer(disk.sec) div 8,'K');
   sect := disk.sec;  { totale settori da leggere }
    s1 := concat(s1,'D',Drive,';',string(integer(sect)),';',string(block),';');
    If Flack Then  s1 := concat(s1,'ACK',';',ver)
           Else  s1 := concat(s1,' ;',ver);
    n := length(s1);
    s2 := 'H0';
    s2 := concat(s2,chr(n>>8),chr(n));
    Tx_string(s2,#0);
    Tx_string(s1,#1);
    If Rx_ack Then Write(' ACK OK ');
 End;

Procedure Tx_H9;
  Var h : char ;
 Begin
   Txby('H');
   Txby('9');
   If Rx_ack Then Write(' ACK OK ');
 End;

Procedure TX_H1;
  var h : char ;
      I,m : Integer;
```

```
  Begin
   If sect <> $0 Then
      Begin
       num := sect div hex(block) ;  { quante letture da 128 settori }
       res := sect mod hex(block) ;  { quanti settori nell'ultima lettura }
       sect1 := $0 ;
       For I := 1 to Integer(num) Do
        Begin
         Write(I:2,'* PSN $',sect1,' '); If ((I mod 5) = 0) Then Writeln;
         m := 0;
         Repeat
          m := m + 1 ;
          Read_Sector(drive,sect1,hex(block),fderror) ; { Read_PSN legge block settori dati
dischetto}
           {  Testare errore fderror  }
         Until (fderror = #$30) Or (m > 5) ;
         If (m > 5) Then
           Begin
            Writeln('TX-H1 Fatal Error in Read_Sector ',fderror,Bel);
            Txby('H');
            Txby('9');
            If Rx_ack Then Write(' ACK OK ');
           End;  { non esco da loop troppo complicato }

         TX_Buffer(scsize*block);
         sect1 := sect1 + hex(block) ;
         If Rx_ack Then Write(' ACK OK ');
        End;
       Writeln((I+1):2,'* Last Block SPSN $',sect1);
       m := 0;
       Repeat
        m := m + 1;
        Read_Sector(drive,sect1,res,fderror) ;
        {  Testare errore fderror  }
       Until (fderror = #$30) Or (m > 5) ;
         If (m > 5) Then
           Begin
            Writeln('TX-H1 Fatal Error in Read_Sector ',fderror,Bel);
            Txby('H');
            Txby('9');
            If Rx_ack Then Write(' ACK OK ');
           End;  { non esco da loop troppo complicato }

       TX_Buffer(Integer(res*hex(scsize))) ;
       If Rx_ack Then Write(' ACK OK ');
      End { if sect }
      Else Writeln('Drive or disk not READY ! ',#7);
    TX_H9;
  End;
```

```
 Procedure Test;
   var a, b : char;
   var i,j : integer;
  Begin
   Repeat
    Writeln;
    Writeln('M6809 in RX');
    i := 0;
    Repeat
     i := i +1 ;
     Write(I:3);
     a := rxby;
     Writeln(' $',hex(a));
    Until a=#0 ;

    Writeln('Premi Enter per passare in TX'); Readln;
     a := '0';
     for i :=1 to 30 do
       Begin
         TxBy(a); Write(i :4);
         a := a + #1;
         For j :=1 to 5000 do;
       End;
      TxBy(#0);
     Writeln('Premi Enter per passare in RX'); Readln;
   Until Break;
  End;

 Procedure ChangeTBS;
  Begin
   Writeln('Transmission Block Size TBS = ',block);
   Repeat
     Write('Input new value (>128 <250) '); Readln(block);
   Until (Block>=128) And (Block<=250);
  End;

 Procedure ChangeDrive;
  Begin
   Repeat
     Write('Change Default FD Drive Number : ',drive,BS);
     Readln(drive);
   Until (Drive >='0') And (Drive <'3');
  End;


{  MAIN  PROGRAM  }
 BEGIN
    Drive := '1' ; { default drive number }
    Block := 250 ; { default block value }
    Flack := False;
```

```
  REPEAT
    Init_Port;
    Writeln('Programma per il trasferimento HS dischetto MLDOS');
    Writeln('da M6809DS a PC via link parallelo 8 bit LL8');
    Writeln('ver : ',ver,' Copyright by Cleto Azzani 1997 (ml sas)');
    Writeln;
    Write('Transmission Block Size TBS : ',block);
    If Flack Then Writeln(' ACK ON',#7)
           Else Writeln(' ACK OFF');
    Writeln('Default FD Drive Number     : ',drive);
    Writeln;

Repeat
{     Writeln('1.........: record H0 --> PC '); }
{     WRITELN('2.........: RECORD h1 --> pc ');  }
{     Writeln('3.........: record H9 --> PC ');  }
    Writeln('4.........: FD Intero --> PC ');
    Writeln('5.........: Test RX-TX');
    Writeln('6.........: Flag ACK Toggle' );
    Writeln('7.........: Change TBS ');
    Writeln('8.........: Change Default Drive number ');
    Writeln('9.........: ???');
    Writeln('0.........: Dos');
    Readln(x);
   Until (x>=0) And (x < 10) ;
   Case x of
{     1 : Tx_H0;  }
{     2 : Tx_H1;  }
{     3 : Tx_H9;   }
    4 : Begin
          TX_H0;
          TX_H1; { COMPRENDE ANCHE TX_H9 }
        End;
    5 : Test;
    6 : Begin
          Flack := Not Flack ;
        End;
    7 : ChangeTBS;
    8 : ChangeDrive;
    0 : ;
   End;
UNTIL X=0 ;
 END. { Main }
```

SEZIONE TURBO PASCAL (LATO PC/XT/AT)

```
Program linkm09(input,output);     file  linkm09.pas

 { Il programma si prefigge di realizzare un link parallelo fra PC MSDOS
   e M6809DS Microlab attraverso il port parallelo Centronics Compatibile
   realizzato mediante scheda LX833 con a bordo 8255 Intel. Il PORT-A viene
   usato come Output Stobed dei Dati; il PORT-B viene usato come Input Strobed
   dei dati. Il cavo usato e' un cavo std Motorola Microlab DB25M per porta
   parallela incrociato (Port A --> Port B e viceversa) }

 USES Dos,Crt,Bios;

   Type
     str6 = string[6] ;
     str8 = string[8] ;

     disktyp = record
                 newname : str8;
                 date    : str6;
                 oldname : string[16] ;
                 Oldate  : string[10] ;
                 user    : string[20] ;
                 apps    : string[16] ;
                 sec     : word ;
               end;

 Const
     Base = $308   ; { Base I/O address of LX833 NE card Centronics Interface }
     PA   = Base   ; { Port A Buffered Output from PC }
     PB   = Base+1 ; { Port B Input from M6809SBC }
     PC   = Base+2 ; { Port C Not Available used for handshaking }
     CW   = Base+3 ; { Control Register 8255 }
     ACK  = Chr(6) ; { Carattere ACK  }
     NACK = Chr(21); { Carattere NACK }
     BEL  = Chr(7) ; { Carattere BEL }
     bmax  = 250   ;  { blocco massimo settori per ogni lettura }
     scsize = 128  ; { ampiezza del singolo settore }
     maxbuf = bmax*scsize ;
     Par : Array[1..10] of string =('Disk Name  : ',
                                    'Disk Date  : ',
                                    'Disk User  : ',
                                    'Disk Apps  : ',
                                    'Drive M6809: ',
                                    'Sectors    : ',
                                    'TX Block   : ',
                                    'TX Mode    : ',
                                    'PGP09 ver. : ',
                                    'Source FD  : ');
```

```
Var
 d, s1, s2, s3, s4, s5, a : string;
 f1, f4 : text;
 f2, f3 : file;
 nb,nbw : word;
 j,k,l,m,n : integer;
 I : longint;
 z : char;
 Flack, Rxf : boolean;
 Buffer : Array[1..maxbuf] of byte;
 Crc  : Array[0..2] of byte;
 CrcR : Array[0..2] of byte; { Buffer del CRC in ricezione Diagnostica }
 pardis : Array[1..10] of string[20];

 Function Valore(s:string):integer;
   Var  i, code : integer;
   Begin
     Val(s, i, code);
     { Error during conversion to Integer? }
     if code <> 0 then
     WriteLn('Error at position: ', code);
     Valore := i;
   End;

 Function RX_byte:byte;  { Manca gestione Timeout di Ricezione }
   var a : byte;
   Begin
     Repeat
       a := Port[PC];
     Until ((a and $10) <>  0);   { attendo CB1 a 1 }
        a:= Port[PB];  { reset flag di ricezione }
     Repeat
     a := Port[PC];
     Until ((a and $02) <>  0);   { attendo IBF1 a 1 }
     RX_byte := Port[PB] ;
   End;

 Procedure TX_byte(data:byte);
  var a : byte;
   Begin
      Repeat
      a := Port[PC];
      Until ((a and $20) <> 0);   { ricevitore in attesa}
      Port[PA] := data ;
      Repeat
      a := Port[PC];
      Until ((a and $80) <> 0);   { ACK da ricevitore }
   End;
```

```
Procedure TX_ACK ;
  Begin
    If Flack Then  Tx_byte(Ord(Ack));
  End;

 { Test di RX TX  PC <---> M6809 }
 Procedure Test;
   var a, b : char;
   var i : integer;
  Begin
   Repeat
     Writeln('Premi Enter per passare in TX'); Readln;
       a := '0';
     for i :=1 to 30 do
       Begin
         TX_byte(Ord(a));
         a := Chr(Ord(a) + 1);
         Delay(2000);
       End;
      TX_byte(0);

    Writeln('PC486 in RX');
    i := 0;
    Repeat
      i := i +1 ;
      Write(I:3);
      a := Chr(RX_byte);
      Write(' $',Ord(a));
    Until a=Chr(0) ;
  Until keypressed;
  End;

{ Visualizza lo stato del Port C }
Procedure DisPc;
  var a : byte ;
  Begin
    ClrScr;
  Repeat
    Delay(5000);
    Port[PA]:= 0;  Gotoxy(23,10);Write('PC7 Low ');
    Repeat
      Gotoxy(10,10);
      a := Port[PC] ;
      Write(hex(a),'-',Bin(a));
    Until ((Port[PC] and $80) <> 0);   { ACK da ricevitore }
    Gotoxy(23,10); Write('PC7 High');
    Until Keypressed ;
  End;
```

```
{ Calcolo del CRC di un file utilizzando il polinomio
  generatore $800FE3: Il CRC generato Š costituito da una stringa a 24
  bit : 3 bytes. La prima release del programma Š stata redatta in
  Omegasoft Pascal M6809. La attuale versione Š stata convertita in
  Turbo Pascal 6.0

  Copyright by Microlab sas (ing. Cleto Azzani) 1985
  Copyright ing. Cleto Azzani                1997       }

Procedure UpdCRC(dato: byte);
  { convertita da ASM6809 by Microlab sas 85 (CA ml 97)}
  var a,b,c : byte;
      d : word;
  Begin
    a := dato ;

    { EORA ,U      PSHS A  }
    a := a xor CRC[0];
    c := a ;

    CRC[0] := CRC[1];
    CRC[1] := CRC[2];

    { CLRA         LDB  ,S       }
    d := c;

    { ASLB         ROLA     EORA  1,U     STD   1,U }
    d := d SHL 1;
    a := Hi(d) xor CRC[1];
    CRC[1] := a;
    CRC[2] := Lo(d);

    { CLRB  LDA  ,S   LSRA    RORB   LSRA  RORB   EORA  1,U   EORB  2,U
      STD   1,U }
    d := c;
    d := swap(d);
    d := d SHR 2;
    a := Hi(d) xor CRC[1];
    b := Lo(d) xor CRC[2];
    CRC[1] := a;
    CRC[2] := b;

    { LDA  ,S   ASLA   EORA ,S    STA  ,S     }
    a := c;
    a := a SHL 1;
    a := a xor c;
    c := a;

    { ASLA  ASLA   EORA  ,S   STA   ,S  }
    a := a SHL 2;
```

```
    a := a xor c;
    c := a;

    { ASLA  ASLA   ASLA   ASLA   EORA  ,S+  }
    a := a SHL 4;
    a := a xor c;
    If ((a AND $80) <> 0) Then
      Begin
        a := $80 ;
        b := $21 ;
        CRC[0] := a xor CRC[0];
        CRC[2] := b xor CRC[2];
      End;
  End;

 Procedure Init_crc;
{ inizializza il buffer del CRC }
  begin
    CRC[0]:= $FF;
    CRC[1]:= $FF;
    CRC[2]:= $FF;
  end;

 Procedure Dis_Crc; { Visualizza contenuto buffer CRC }
  begin
    Writeln('Crc = $',hex(Crc[0]),hex(Crc[1]),hex(Crc[2]));
  end;

 { La funzione RX_buffer riceve un buffer di dati da M6809.
   Il valore restituito rappresenta il tipo di buffer ricevuto:
   0 se buffer ASCII intestazione;
   1 se buffer BINARIO ;
   9 se buffer vuoto finale     }

 Procedure RX_buffer(var typerec : char ; var k : word; var flag : boolean) ;
   Var   c, d : byte ;
         I  : word   ;

  Begin
    Init_Crc;
    c := RX_byte ;   { primo carattere obbligatoriamente H }
    If c <> Ord('H') Then Writeln(Ord(c),': H Error RX_buffer',Bel);
    typerec := Char(RX_byte) ;   { secondo carattere : tipo di record }
    If (Typerec = '0') Or (Typerec = '1') Then
      Begin
        Flag := False;   { flag di CRC }
        c := RX_byte ;   { MSD lunghezza blocco in hex }
        d := RX_byte ;   { LSD lunghezza blocco in hex }
        k := Ord(d)+Ord(c)*256;  { lunghezza del blocco }
```

```
        For I := 1 to K do
          Begin
            Buffer[I] := RX_byte;
            UpdCrc(Buffer[I]);
          End;
{         Write('-EOB- ');   }  { Ricevo CRC }
        For I := 0 to 2 do
          Begin
            CrcR[I] := RX_byte;
            UpdCrc(CrcR[I]);
          End;
        If ((CRC[0]=$80) and (CRC[1]=$0F) and (CRC[2]=$E3))
          Then
            Begin
              Flag := True;
              Write('CRC OK ');
            End
          Else Writeln('CRC K.O. !',Bel);
       End
      Else  If (Typerec <> '9') Then Writeln('Type of Record Unknown ',Bel);
    End;

  { Restituisce la data attuale AAMMGG stringa di 6 caratteri }

  Function Data_attuale : string ;
    Var
      year, Month, Day, Dayweek : word;
      ystr,mstr,dstr : string[4];

    Begin
      GetDate(year,Month,Day,Dayweek);
      Str(year,ystr);
      Str(month,mstr); If length(mstr) =1 Then mstr:= '0'+mstr;
      Str(Day,dstr); If length(dstr) =1 Then dstr:= '0'+dstr;
      Data_attuale:=Copy(ystr,3,2)+mstr+dstr;
    End;

 { Restituisce l'orario corrente in formato HHMMSS stringa di 6 caratteri }

   Function Ora_attuale : string ;
    Var
      Hour, Min, Sec, Sec100 : word;
      Houstr,minstr,Secstr : string[4];

    Begin
      GetTime(Hour, Min, Sec, Sec100);
      Str(Hour,Houstr); If length(houstr) =1 Then houstr:= '0'+houstr;
      Str(Min,minstr); If length(minstr) =1 Then minstr:= '0'+minstr;
      Str(Sec,Secstr); If length(Secstr) =1 Then secstr:= '0'+secstr;
      Ora_attuale:=Houstr+minstr+secstr;
```

```
   End;

{ Generatore automatico sequenziale del nome di dischetti :
  nome costituito da un prefisso di due caratteri e seguito da sei
  caratteri alfanumerici da 0 a 9 e da A a Z (solo maiuscole)
  Due caratteri costituiscono il prefisso FD }

Function Succ_disco(disco : str8) : str8;
 var k : integer ;
     ovf : boolean ;
 Begin
   k := length(disco)+1;
 Repeat
    Dec(K);
    Ovf := False;
    If K=2 Then
       Begin
         Insert('0',Disco, 3);
         Inc(k);
       End
      Else
       Begin
         Disco[k] := Chr(Ord(Disco[k])+1);
         If (Disco[K]>'9') And (Disco[K]<'A') Then Disco[K] := 'A';
         If (Disco[K]='O') Then Disco[K] := 'P';
         If (Disco[k] > 'Z') Then
            Begin
              ovf := True;
              Disco[K] := '0' ;
            End;
       End;
  Until (Not Ovf) Or ((k=3) and (length(disco)=8));
  If ((k=3) and (length(disco)=8))
     Then Writeln('Error in Disco ADJ ',Bel);
  Succ_Disco := disco ;
 End;

Function NomeDisco : str8 ;
  Var f4 : file of str8;
      d  : str8 ;
      s4 : string;

  BEGIN
    s4 := 'LINKM09.DSK';
    Assign(f4,s4);
    If FilSrc(s4)
       Then
         Begin
           Reset(f4);
           Read(f4,d);
```

```
          Seek(f4,0);
          d := Succ_Disco(d);
        End
      Else
        Begin
          Rewrite(f4);
          Repeat
            Write('Nome disco iniziale (max.6 ch.) : '); Readln(d);
          Until length(d)<=6;
          For i := 1 to length(d) do D[i] := upcase(D[i]) ;
          d := 'FD'+d ;
        End;
        Write(f4,d);  { Memorizza l'ultimo nome fornito }
        NomeDisco := d;
  Close(f4);
END;

Procedure Open_Log;
  Begin
    { Creazione Logfile o Append su Logfile }
   s1 := 'LINKM09.LOG';
   Assign(f1,s1);
   If FilSrc(s1) Then
     Begin
       Append(f1);
       Writeln(f1,'Append   :',Data_Attuale:6,'-',Ora_Attuale:6);
     End
    Else
      Begin
        Rewrite(f1);
        Writeln(f1,'Created : ',Data_Attuale:6,'-',Ora_Attuale:6,' by Parallel Link-M6809');
      End;
  End;

Function TrimSpace(str1:string) : string ;
  Var st : string;
  Begin
    st := str1 ;
    { Elimina gli SPACE iniziali in str1}
    While Copy(st,1,1) = ' '
         do  Delete(st,1,1);
    While Copy(st,Length(st),1) = ' '
         do  Delete(st,Length(st),1);
    TrimSpace := st;
  End;

Procedure Parametri;
  var  k : integer;
       a : string;
  Begin
```

```
  For k := 1 to 10 do pardis[k] := '';

  k := 0 ; a := '';
  For j := 1 to nb do
    Begin
      If (Buffer[j] <> Ord(';'))
        Then a := a + Chr(Buffer[j])
        Else
          Begin
            Inc(k);
            pardis[k] := TrimSpace(a);
            a := '';
          End;
    End;
  Inc(k);
  pardis[k] := TrimSpace(a);

End;

Procedure Ricevi;
  var block, nl : integer;
     Dsize, sect  : longint ;
  BEGIN
   s3 := Concat(Nomedisco,'.XDI');
   Assign(f2,s3);
   Rewrite(f2,1);
   Writeln(f1,'Open file  : ',s3,'  ',Data_Attuale:6,'-',Ora_Attuale:6);
   Writeln(f4,'Open file  : ',s3,'  ',Data_Attuale:6,'-',Ora_Attuale:6);
   Writeln(f4);
   Writeln; Writeln;
   Writeln('Open file  : ',s3,'  ',Data_Attuale:6,'-',Ora_Attuale:6);
   I :=0;
   Repeat
    Write(I:2,'-');
    Rx_Buffer(z,nb,Rxf);
    If ((I mod 5) = 0) Then Writeln; { a capo }
    CASE Z Of
     '0' :  Begin
              Writeln(f1,'RX_H0 :',Data_Attuale:8,'-',Ora_Attuale:6);
              Writeln('Inizio RX : ',Ora_Attuale:6);
              Writeln;

              { Recupera parametri dal buffer }
              Parametri ;
               s5 := pardis[5];
               sect  := Valore(pardis[6]);
               block := Valore(pardis[7]);
               nl := sect div block;

              If sect <> 3680 Then
```

```
            Begin
              Case sect of
                2000    :  Pardis[10] := 'FD 8-SS';
                4004    :  Pardis[10] := 'FD 8-DS';
              End;
            End
           Else
            Begin
              Case s5[2] of
                '0'     : Pardis[10] := 'FD 3.5-DD';
                '1'     : Pardis[10] := 'FD 3.5-DD';
                '2'     : Pardis[10] := 'FD 5.25-DD';
              End;
            End;
           Writeln('RX disco : ',pardis[1],' ',pardis[2],' ',pardis[3],' ',pardis[4]);
           Writeln('Settori da leggere   : ',Sect,' N. sett/blocco : ',block);
           Writeln('Totale trasferimenti : ',(nl+1)); Writeln;

          { Scrive parametri su file LOG }
          For j := 1 to 10 do Writeln(f4,Par[j],pardis[j]);

          Writeln(f4,'----------------------------------------');

          Tx_Ack;
        End;

   '1' : Begin
          If I = 1
            Then  Writeln(f1,'RX_H1 :',Data_Attuale:8,'-',Ora_Attuale:6);
          If I = nl Then Write(Bel);
          BlockWrite(f2,Buffer,nb,nbw); If(nbw<>nb) Then Writeln('Blockwrite ERROR');
          Tx_Ack;
        End;

   '9' : Begin
          Writeln('Fine Record H0..H1..H9 ');
          Writeln(f1,'RX_H9 :',Data_Attuale:8,'-',Ora_Attuale:6);
          Writeln('Fine RX : ',Ora_Attuale:6);
          Tx_Ack;
          {  Port[CW] := $AF ;
            z := Chr(Port[PB]); }
        End;
  End;
  Inc(I) ;
Until Z = '9' ; { fine trasmissione }
Writeln('Size of ',s3,' = ',FileSize(f2));
Dsize := sect * scsize ; Writeln('Dsize : ',Dsize);
If FileSize(f2) <>  Dsize
   Then
    Begin
```

```
          Writeln('Error in RX Disk',Bel);
          Readln;
         End;
    Close(f2);
  END;

 Procedure RX6809;
  BEGIN
   Writeln('Attivare M6809 in TX-mode ',Bel );
   Readln;

   { Creazione Catalogo file ricevuti o Append su Catalogo }
   s4 := 'LINKM09.LST';
   Assign(f4,s4);
   If FilSrc(s4) Then
      Begin
       Append(f4);
       Writeln(f4,'Append   :',Data_Attuale:6,'-',Ora_Attuale:6);
      End
     Else
      Begin
       Rewrite(f4);
       Writeln(f4,'Created : ',Data_Attuale:6,'-',Ora_Attuale:6,' by Parallel Link-M6809');
      End;
   Writeln(f4,'----------------------------------------');
   Writeln(f1,'Open Catalog file  : ',s4,'  ',Data_Attuale:6,'-',Ora_Attuale:6);

   Writeln('PC in RX-Mode');
   Repeat
    Ricevi;
    Writeln('Premere F10 per interrompere la ricezione continua',Bel,Bel);
    Delay(50000);
   Until FKB(10);
   Writeln(f1,'Close Catalog file  : ',s4,'  ',Data_Attuale:6,'-',Ora_Attuale:6);
   Writeln(f4,'Closed : ',Data_Attuale:6,'-',Ora_Attuale:6);
   Close(f4);
  END;

BEGIN { inizio programma }
  Port[CW] := $AF ;  { Port A Strobed Output; Port B strobed input }
  ClrScr;
  Writeln('Link PC M6809SBC Utility for Transfer File');
  Writeln('LL8P mode - LX833-8255 Base $',Hex(Base));
  Writeln('ver. 2.0  Copyright by CA ml sas 1997');
  Writeln;
  Open_log;
  Flack := False;
  Repeat
   Repeat
    z := Chr(Port[PB]);
```

```
        Writeln('1.....Ricezione  Record Hx da M6809DS');
        Writeln('3.....Test di TX-RX - PC <--> M6809DS');
        Writeln('4.....Flag ACK toggle ');
        Writeln('5.....Generazione Disk Name ');
        Writeln('6.....Display Port C ');
        Writeln('0.....DOS');
        Readln(k);
      Until  (K>=0) And (K<=6) ;
      Case K Of
        1 :  Rx6809;
        3 :  Test;
        4 :  Begin
               Flack := Not Flack;
               If Flack Then Writeln('Flag ACK ON ');
             End;
        6 :  DisPC;
        5 :  Begin
               Repeat
                 Writeln(Nomedisco);
                 Readln;
               Until False;
             End;
        0 :   ;
        Else Writeln('Unimplemented function',Bel);
      End;
Until k=0 ;
Close(f1);
END .
```