PICCOLINE

Programmer's

Guide

PICCOLINE

Programmer's Guide

PICCOLINE Programmer's Guide Version 2.0

PN: 9900[illegible]

Keywords: PICCOLINE, RC759, operating system, CCP/M-86, software, hardware.

Abstract: This manual describes the peripheral devices used in PICCOLINE. The manual contains information that enables an advanced user to implement his own drivers for PICCOLINE peripherals.

Throughout this manual it is assumed that the reader is familiar with the ASM86 or the RASM86 assembler, with the Concurrent CP/M operating system and with peripheral device interfacing (interrupts etc.).

The manual is intended for use in connection with version 3.1 of the CCP/M operating system and version 2.3 of the XIOS.

Table of Contents:

Appendices:

1. Introduction

The intention of this manual is to enable programmers to use the PICCOLINE peripherals in their own ways.

The following peripherals, which are either part of the CPU or devices connected to the CPU, are standard in a PICCOLINE system:

2 DMA channels (integrated on CPU)

3 Timers (integrated on CPU)

1 Interrupt controller (integrated on CPU)

1 Intel 8259A programmable interrupt controller

CRT controller based on Intel 82730

Keyboard interface

Real time clock

Sound device

Non volatile memory (NVM)

Local parallel printer interface

Cassette tape interface

Connector to PICCOLINE Disk/Printer-Adaptor

Connector to micronet interface

iSBX-connector

The interconnection of these peripherals is shown on fig.1.1 page 2.

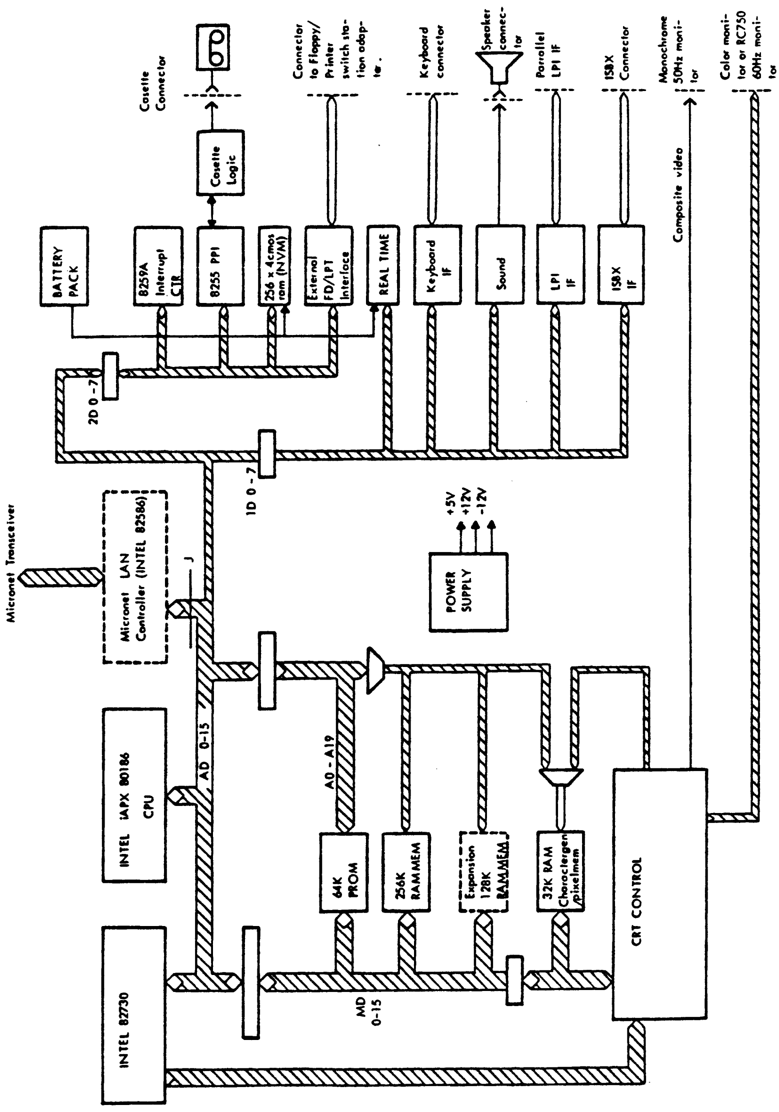


Fig.1.1 Block Diagram.

Besides these standard peripherals the PICCOLINE system may be enhanced with

- A local area network controller based on the Intel 82586 ethernet controller
- A PICCOLINE Disk/Printer-Adaptor (if PICCOLINE is connected to disks resident inside the PICCOLINE Disk/Printer-Controlunit (DPC))
- Other controllers connected to the iSBX-connector. This connector is supported in accordance to the iSBX bus standard from INTEL. In this manual only the iSBX serial multimodule board is described.

In this manual the software interface to the above mentioned peripherals will be described.

1.1 XIOS Overview

The XIOS (eXtended Input/Output System) is the lowest layer of software in the PICCOLINE.

The XIOS consists of a set of routines, each controlling a specific hardware component, which together constitutes a welldefined interface to the CCP/M operating system (see ref.3)

A XIOS routine is executed as part of the user programs as a consequence of operating system calls. When a user program has requested a service by means of an operating system call, the program will be suspended (i.e. the program will not return from the XIOS routine) until the requested service can be fullfilled (e.g. a sector on the floppy disk has been read).

Table 1.1 is an overview of the available XIOS routines.

Routine Name	Routine Number
IO_CONST	0
IO_CONIN	1
IO_CONOUT	2
IO_LISTST	3
IO_LIST	4
IO_AUXIN	5
IO_AUXOUT	6
IO_SWITCH	7
IO_STATLINE	8
IO_SELDSK	9
IO_READ	10
IO_WRITE	11
IO_FLUSHBUF	12
IO_POLL	13
Not used	14
Not used	15
WW_POINTER	16
WW_KEY	17
WW_STATLINE	18
WW_IM_HERE	19
WW_NEW_WINDOW	20
WW_CURSOR_VIEW	21
WW_WRAP_COLUMN	22
WW_FULL_WINDOW	23
WW_SWITCH_DISPLAY	24
Not used	25
Not used	26
Not used	27
Not used	28
Not used	29
GET_SCREEN_MODE	30
Not used	31
Not used	32
Not used	33
Not used	34
Not used	35
Not used	36

Table 1.1. XIOS routines.

The routines with numbers from 0 to 13 is described in ref.3, while a description of the remaining routines may be found in chapter 4.6.

All the above mentioned XIOS routines have a common convention concerning the contents of the registers when the routines are entered. The convention is as follows:

Register AL contains the routine number

Register ES contains the paragraph address of the calling process' User Data Area (UDA)

Register DS contains the SYSDAT segment address

When the XIOS routines are entered as a consequence of a CCP/M operating system call, CCP/M manages the above mentioned conventions. On the other hand, when the XIOS routines are entered directly from a user program it is the responsibility of this program to establish the register contents before entering the routine.

Besides the common register contents, a XIOS routine may require some parameters which for some of the routines are transferred in a register and for other routines are transferred on the stack. A detailed description may be found in ref.3.

Example

This example shows how a program can initialize the ES and DS register with the UDA and SYSDAT values and how the standard XIOS routines are entered.

```
; Get Process Descriptor Address.
; The address segment is returned in ES and
; in BX (used later)
Mov  CL,156
Int  224

; Initialize DS to SYSDAT segment using
; the fact that the process descriptor
; segment is the same as the SYSDAT segment
Push ES
Pop  DS

; Initialize ES with UDA address. UDA address
; is taken from the process description word 10H
Mov  ES,10HÆBXÅ

; Now initialize all routine dependent parameters
; (either register parameters or parameters on stack).
```

```
    ; Enter the routine via the XIOS entry field in SYSDAT
    Mov       AX,routine_number
    Callf     DS:Dword Ptr .28h
```

As an extension to the standard XIOS routines some extra routines have been implemented. Opposed to the standard routines, which are entered through a far call via the XIOS entry field in the SYSDAT area, these extra routines are entered by executing a software interrupt on level 28h. A detailed description of the extra routines may be found in appendix A. In the remaining chapters the extra routines will be denoted as 'Int-28h functions'.

The synchronization between the interrupt service routines for the different peripherals and the programs using the peripherals is done by means of the CCP/M flag mechanism (see refs.2,3).

Table 1.2 shows how these flags are assigned on the PICCO-LINE.

Flag number	Use
0	Reserved by CCP/M
1	Tick
2	Second
3	Minute
4	Scroll synchronization
5	Key available flag
6-7	Reserved
8	Floppy disk
9	Scroll synchronization
10	Scroll synchronization
11	Floppy motor
12	Local parallel interface
13-18	Reserved
19	Error key flag
20-21	Reserved
22	Net transmitter
23	Net receiver
24	Window manager
25	DPC parallel printer
26-63	Reserved for future use
64-127	Free
128-255	Reserved by DR Net

Table 1.2. Flag Assignments

In order to manage reservation of different resources, the operating system maintains a number of queues. As queue names must be unique, the names of these queues are reserved by the operating system. A list of reserved queue names may be found in table 1.3.

Name	Number of messages	Message length	Usage
Tmp0	1	112	See below
Tmp1	1	112	See below
Tmp2	1	112	See below
Tmp3	1	112	See below
VCMXQ0	1	0	Virtual console 0
VOUTQ0	16	2	- do -
VINQ0	64	2	- do -
VCMXQ1	1	0	Virtual console 1
VOUTQ1	16	2	- do -
VINQ1	64	2	- do -
VCMXQ2	1	0	Virtual console 2
VOUTQ2	16	2	- do -
VINQ2	64	2	- do -
VCMXQ3	1	0	Virtual console 3
VOUTQ3	16	2	- do -
VINQ3	64	2	- do -
MXalt	1	0	Alt. charset reservation
XMIT_REQ	10	15	Net driver (Net system only)
link_req	1	15	Net driver (Net system only)
MXdma1	1	0	DMA channel 0 reservation
MXdma2	1	0	DMA channel 1 reservation
MXsound	1	0	Sound device reservation
MXLoad	1	0	Used during program load
MXdisk	1	0	Disk system reservation
MXcass	1	1	Cassette tape reservation

Table 1.3. Reserved Queue Names.

The Tmp queues (Tmp0, Tmp1, Tmp2 and Tmp3) are primarily intended for use in connection with the menu system to facilitate the loading of menu programs and the return to the outermost menu level, but may also be used by ordinary programs.

The function of the Tmp queues is as follows:

When a Tmp succeeds in the attempt to attach to its default console, the first step is to make a conditional queue read on the relevant Tmp queue. If this read is successfull the Tmp will use the data read as if it was a command line read from the keyboard (i.e. the same syntax as for command lines is valid, including multible commands separated with the sequence '//'). If no data was read the Tmp makes a 'read console buffer' operating system call to get the command line from the keyboard.

2. CPU

The PICCOLINE system is based on an Intel 80186 single chip CPU with the following integrated peripherals:

- Programmable interrupt controller
- 2 Independent DMA channels
- 3 Programmable 16-bit timers

All the integrated peripherals are controlled via 16-bit registers contained within an internal 256-byte control block. The base address of this control block is 0FF00H.

The following three chapters give a description of how these peripherals are used in the PICCOLINE.

2.1 Interrupt System

The peripherals which are able to interrupt the CPU are connected to the internal interrupt controller via an Intel 8259A Programmable Interrupt Controller which uses the following I/O addresses:

Initialization command word:	0H
Operation command word:	2H

The IR inputs to the Intel 8259A are connected as follows:

IR0:	Floppy controller (from DPC unit)
IR1:	Keyboard interface
IR2:	Parallel (printer) interface (from DPC unit)
IR3:	Real Time Clock
IR4:	CRT controller
IR5:	NET controller
IR6:	Parallel (printer) interface (from CPU unit)
IR7:	not used

The Intel 8259A is connected to the INT0 and INTA0 terminals of the CPU.

INTR0 from the iSBX connector is connected to INT1 on the CPU (vector type 13) and INTR1 to INT3 (vector type 15)

The internal interrupt controller is initialized to cascade mode and level triggered interrupts.

The Intel 8259A is initialized to buffer mode, master, no slaves connected, fully nested interrupts, specific end of interrupt, level triggered and first vector 80H.

A list of interrupt vector assignments may be found in appendix C.

Details about the interrupt controllers may be found in the Intel reference documentation.

2.2 Direct Memory Access

The two integrated DMA channels are able to transfer data between memory and I/O space (e.g. Memory to I/O) or within the same space (e.g. Memory to memory or I/O to I/O). Data can be transferred either in bytes (8 bits) or in words (16 bits) to or from even or odd addresses.

DMA channel 0 is connected to the iSBX connector while DMA channel 1 is used as floppy tranfer channel.

Detailed information about the DMA channels may be found in the Intel reference documentation.

2.2.1 DMA Channel Reservation

As the two DMA channels are shared among different peripheral devices, it is necessary to reserve a channel before using it. The reservation of the channels are done by means of two mutual exclusion queues, 'MXdma0' and 'MXdma1'. When a program succeeds in reading one of these queues, it has got the right to use the corresponding DMA channel. The DMA channel is released by writing to the relevant mutual exclusion queue.

2.2.2 DMA Request Line Setup

Each of the two DMA channels can handle DMA requests from the following different sources:

DRQSEL	DMA0 source	DMA1 source
0 0	iSBX	DPC (external floppy)
0 1	0	0
1 0	DPC (external floppy)	iSBX
1 1	1	0

When a program has succeeded in reserving a DMA channel, it must set up a connection for the DMA request signal, between the peripheral device and the DMA controller. This is done by writing a control byte to a parallel port located at I/O address 74H. The format of the control byte is as follows:

Bit 0	ENABLE CAS(sette)
Bit 1	MOTOR OFF
Bit 2-3	DRQ0-1
Bit 4-5	NVMA0-1 (NVM bank)
Bit 6-7	must have the binary value 11.

2.2.3 DMA Interrupt Handling

The two DMA channels are connnected to the internal 80186 interrupt controller. The interrupt level of DMA channel 0 and 1 is 10 and 11.

Example

```
DMAInterruptService:
    ; save context
    Push DX
    Push AX

    ; Non specific end of interrupt
    ; to internal interrupt controller
    Mov  DX,0FF22H
    Mov  AX,8000H
    Out  DX,AX
```

```
; restore context
Pop  AX
Pop  DX
Iret
```

2.3 Timers

The three 16-bit timers are used for the following purposes:

Timer 0 is used to generate serial output to the audio cassette interface

Timer 1 is used to generate audio output (the 'BELL')

Timer 2 is reserved for future use

Detailed information about the timers may be found in the Intel reference documentation.

3. Configuration

The basic configuration of the PICCOLINE has two forms:

1. During the initialization after power up or any kind of reset, the software investigates the hardware environment to determine the size of the main memory, the number of disks attached etc. This kind of configuration is called the auto configuration.

2. During system initialization the operating system initializes the serial communication controller if any, the cursor representation, the floppy motor timer etc. This initialization is done on the basis of the contents of the non volatile memory (NVM). The content of the NVM is normally only modifiable by the KONFIG program (ref.5).
   If the file KDEF.SYS exist on the load device, the file is read during system initialization and the content of KDEF.SYS is written in the NVM.
   The file KDEF.SYS is modified by the program OKONFIG.

3.1 Auto Configuration

The hardware configuration map is accessible for the programmer by means of the Int-28h function 4.

This function returns a pointer to the configuration map (see appendix A).

NOTE

The contents of the configuration map must not be modified.

The configuration map has the following format:

Byte offset	explanation
0-3	This double word contains the main memory size in bytes.
4-7	This double word contains the total memory size in bytes (including the CRT pixel memory).
8-11	Reserved.

12	The value of this byte is 0FFH if the real time clock second source is installed (see 5.1). Otherwise the value is 0.
13	The value of this byte is 0FFH in case the local area net work controller is installed. Otherwise the value is 0.
14	The value of this byte is 0FFH in case any iSBX module is installed. Otherwise the value is 0.
15	This byte is set to 0FFH if the Disk/Printer (DPA) is installed. Otherwise the value is 0.
16-17	Reserved.
18	This byte specifies the type of attached monitor:

0H	15.625 kHz	colour
1H	15.625 kHz	monochrome
2H	22 kHz	colour
3H	22 kHz	monochrome

19	This byte hold the number of floppy drives connected to the system.
20-21	Reserved.
22	This byte contains the value of the nationality code switch of the keyboard (range 0-15).

3.2 Non Volatile Memory

The function of the NVM is to keep various system parameters during power down periods.

The NVM is made up of a 256 by 4 bit CMOS RAM with battery backup.

The NVM is divided into 4 blocks each containing 64 4-bits nibbles. A block is selected by means of bit 4 and bit 5 in the I/O port at address 74H. Please note that a block select operation must not affect the other bits in the I/O port.

After a block has been selected, the 64 nibbles in the block are accessible on the even I/O addresses from 80H to 0FEH. When an IN or OUT instruction is executed with one of these addresses, the four least significant bits of register AL will be transferred to/from the NVM.

A copy of the NVM is accessible for the programmer by means of Int-28h function 3:

Registers at entry:

AL	3

Registers at return:

ES	NVM copy pointer segment
SI	NVM copy pointer offset

The NVM layout is as follows (seen as bytes):

byte number	description
0	Checksum (see below).
1-2	Type number.
3-4	0
5-6	Serial number.
7-12	Reserved.
13	Baud rate and mode for the iSBX351/V24 channel. High nibble is baudrate (receive baudrate = transmit baudrate). The nibble coding is:

0:	75	baud
1:	75	-
2:	110	-
3:	150	-
4:	300	-
5:	600	-
6:	1200	-
7:	2400	-
8:	4800	-
9:	9600	-

Low nibble designates channel usage:

0:	virtual console
1:	printer

14 Stop bit and parity.
Parity is coded in bit no 0 and 1:

0:	no
1:	odd
3:	even

Stop bit is coded in bits no 2 and 3:

1:	1
2:	1.5
3:	2

15 Initial value of RTS. Bit no 1 is coded as follows:

0:	low
1:	high

16 Reserved.

17 Number of bits pr. char. The bits no 6 and 7 are encoded as follows:

0	5
1	6
2	7
3	8

18 4 most significant bits hold CRT scroll mode. (0=jmp mode; 1=soft scroll mode).

19 4 most significant bits hold the cursor height (1 to 10 video lines). 4 least significant bits holds the cursor blink mode (0=solid; 1=blinking).

20	Number of idle seconds before the floppy motor stops (0-255). Must not be less than 2. It is also the time before the floppy disks is released.
21	Reserved.
22	Default foreground colour. The bits are encoded as follows: Bit 0 blue beam on/off Bit 1 yellow beam on/off Bit 2 red beam on/off Bit 3 high intensity on/off Bit 4-7 0
23	The month of the last power on (1H-12H).
24	Current year (78H-99H).
25	Load device (i.e. the device from which the operating system is loaded). The value is the disc drive letter 'A' and 'B' or the letter 'N' which means load via the local area network. It is also possible to load from prom on special PICCOLINEs and in this case the value is 'P'.
26	Number of disk buffers (0-255).
27	Memory disk size: 0: 0 Kbytes 1: 64 Kbytes 2: 128 Kbytes 3: 192 Kbytes 4: 256 Kbytes
28	Hardcopy printer type: 0: All characters in the range 32 to 126 are printed without conversion. All other characters are converted to blanks. 1: All characters are printed without conversion.

29	DR Net node id (0-254).
30	DR Net default server id (0-254).
31	Autologon mask: Bit 0=1: Virtual console 0 is automatically logged on to default server when the system is started. Bit 1=1: Same as above for console 1. Bit 2=1: Same as above for console 2. Bit 3=1: Same as above for console 3.
32-33	Reserved.
34-41	DR Net server password (8 ascii characters).
42	Reserved.
43-50	Name of file to load when load from the local area network is used (8 ascii characters).
51	Identification of the iSBX module: 0: Ingen 1: 351 2: 488
52	System disk number (0-15).
53	CPU identification. FFH if PICCOLINE and 0H if Partner.
54-127	Reserved.

Byte number 14, 15, 16 and 17, which hold the format of the iSBX351/V24 channel, has the same format as Intel 8274 serial controller write registers 4, 5, 1 and 3, which is used in the Partner (RS232C/V24 channel)

The checksumbyte is used to ensure data integrity in the NVM. The checksum is calculated so that if the bytes in NVM block 0, 1 and 2 (not block 3) are added (modulo 256) the sum should be 0AAH. The checksum must be maintained when the NVM contents are changed.

Example

This example shows how to read and write in the NVM while maintaining the checksum.

```
;procedure write_nvm(block,offset,value);
;entry    : al: offset from block base to the desired byte
;           ah: block_number (0,1,2 or 3)
;           cl: byte to be written
;
;exit     : the nvm checksum (0AAH) are maintained
;
;detroyed: none

    checksum_block       equ         0
    checksum_offset      equ         0
    block_base           equ        80H
    nibble_msk           equ        0FH

write_nvm:
    push dx                         ; save registers
    push bx                         ;
    push cx                         ;
    push ax                         ;
    call read_nvm                   ; read the old value
    mov  bl,al                      ; save old value in bl
    mov  ah,checksum_block          ;
    mov  al,checksum_offset         ;
    call read_nvm                   ; read the old checksum
    mov  bh,al                      ; save it in bh
    pop  ax                         ;
    push ax                         ; save byte number
    call address_block              ; address the block to
                                    ; be written
    pop  ax                         ;
    mov  dx,block_base              ;
    shl  al,1                       ;
    shl  al,1                       ;
    xor  ah,ah                      ;
    add  dx,ax                      ; address of first nible
    pop  cx                         ; retrieve value to be
                                    ; written
    push cx                         ;
    mov  al,cl                      ;
    mov  cl,4                       ;
    shr  al,cl                      ; strip least signifi-
                                    ; cant nibble
    out  dx,al                      ;
    pop  cx                         ;
    mov  al,cl                      ;
    and  al,nibble_msk              ; strip most significant
                                    ; nibble
    add  dx,2                       ;
    out  dx,al                      ;
                                    ; checksum update new val
                                    ; in cl old val in bl old
                                    ; sum in bh
    sub  bl,cl                      ; oldval-newval
    add  bh,bl                      ; sum:=sum
                                    ;      +(oldval-newval)
    mov  ah,checksum_block          ;
    call address_block              ; address the checksum
                                    ; block
    mov  dx,block_base              ;
    mov  al,bh                      ;
    mov  cl,4                       ;
    shr  al,cl                      ; strip least signifi-
                                    ; cant nibble
    out  dx,al                      ;
    mov  al,bh                      ;
    and  al,nibble_msk              ; strip most significant
                                    ; nibble
    add  dx,2                       ;
    out  dx,al                      ;
    pop  bx                         ;
    pop  dx                         ;
    ret


;procedure read_nvm(block,offset,value);
;entry:
; al: offset from block base to the desired byte
; ah: block number (0,1,2 or 3)
;
;exit:
; al: the desired byte


read_nvm:
    push dx                         ; save registers
    push cx                         ;
    push ax                         ;
    call address_block              ; select block
    mov  dx,block_base              ;
    pop  ax                         ;
    shl  al,1                       ; convert byte to nibble
                                    ; offset
    shl  al,1                       ;
    xor  ah,ah                      ;
    add  dx,ax                      ;
    in   al,dx                        ; get high nibble
    add  dx,2                         ;
    xchg ah,al                        ;
    in   al,dx                        ; get low nibble
    mov  cl,4                         ;
    shl  ah,cl                        ; shift high nibble to
                                      ; the correct byte
                                      ; position
    and  al,nibble_msk                ; clear high nibble in al
                                      ;
    or   al,ah                        ; transform nibbles to
                                      ; bytes
    pop  cx                           ;
    pop  dx                           ;
    ret



    nvm_control_port     equu          74H
    clear_msk            equ          0CFH

address_block:                        ; select block number(ah)
    mov  dx,nvm_control_port          ;
    in   al,dx                        ;
    and  al,clear_msk                 ;
    mov  cl,4                         ;
    shl  ah,cl                        ;
    or   al,ah                        ;
    out  dx,al                        ;
    ret
```

4. Console Module

The console module handles the virtual consoles and the keyboard. The operating system accesses the console module through the XIOS conin and conout calls.

The operating system may also be bypassed and the console module accessed directly, and for special purposes the application program may access the hardware directly e.g. by supplying its own interrupt routines.

This section contains a description of the software interface to the console module and a brief description of the associated hardware.

4.1 CRT controller

The CRT controller is built around an Intel 82730 text processor. For a complete description of this chip please refer to the relevant Intel documentation.

This section contains information for programmers who want to make special use of the PICCOLINE hardware including the Intel 82730 text processor.

To access the CRT controller, palette and pixel memory directly it is required that

1. The process is executing in the foreground

2. The console is locked (console switching inhibited)

3. The CRT controller environment is restored before program termination

4.1.1 82730 Command Block

Communication between the 82730 and the CPU takes place through a command block placed in main memory. The address of the command block is returned by an Int-28h function accessed with the following register contents:

Registers on entry:

```
    AL = 21
```

Registers on return:

```
    ES = segment register of the command block
    SI = offset register of the command block
```

4.1.2 Character Format

The 82730 fetches characters from main memory and outputs these to the CRT controller. The characters have the following format in alphanumeric mode:

bit 0-9	character address
bit 10-14	palette select
bit 15	0

and in graphics mode:

bit 0-9	pixel block address
bit 10-13	palette select
bit 14	0 = high resolution graphics 1 = medium resolution graphics
bit 15	0

If bit 15 is a 1, the 82730 interprets the character as a character stream command.

In both alphanumeric and graphics mode bit 0-9 of the character addresses a pixel block in the 32k pixel memory located at address D000:0000H

In alphanumeric mode the pixel blocks function as character generators. One pixel on the screen corresponds to one bit in the pixel memory. The width of the character may vary from 7 to 15 pixels depending on the contents of the pixel memory (see 4.3). The height of one character row is 10 videolines in the standard configuration.

In graphics mode the pixel blocks are normally organized so that the 32k pixel memory makes up a complete bitmap of the screen. One pixel on the screen corresponds to one bit in the pixel memory in high resolution graphics mode, and to two bits in medium resolution. The pixel blocks are 16 pixels high by 16 pixels wide in high resolution and 8 by 16 pixels in medium resolution corresponding to 16 words of memory.
The total resolution is 560 by 250 pixels in alphanumeric mode, 560 by 256 pixels in high resolution graphics mode and 280 by 256 pixels in medium resolution graphics mode.

4.1.3 Palette

The output from the pixel memory is used to select one of two (alphanumeric and high resolution graphics mode) or one of four (medium resolution graphics mode) colours from a palette.

The palette has room for 32 bytes each containing two 4-bit nibbles, which are interpreted as follows:

bit 3:	I, if set the intensity is increased
bit 2:	R, if set the red beam is turned on
bit 1:	G, if set the green beam is turned on
bit 0:	B, if set the blue beam is turned on

If a monochrome monitor is connected, only bit 2 is used. The intensity bit, bit 3, has no effect.

The palette is written with an OUT instruction to I/O address 180H to 1BEH (even addresses). In the following table the relation between palette cells and I/O addresses is shown:

I/O address		colour pair	
dec	hex		
384	180	1	0
386	182	3	2
388	184	5	4
.	.		
.	.		
446	1BE	63	62

The following tables show the relation between the value of the palette selector and the palette cells selected:

Alphanumeric mode:

Palette Selector	Pixel = 1	Pixel = 0
0	1	0
1	3	2
2	5	4
.		
.		
31	63	62

High resolution graphics:

Palette Selector	Pixel = 1	Pixel = 0
0	1	0
1	3	2
2	5	4
.		
.		
15	31	30

Medium resolution graphics:

Palette		Pixel pair		
Selector	11	10	01	00
0	33	32	1	0
1	35	34	3	2
2	37	36	5	4
.				
.				
15	63	62	31	30

Examples:

Alphanumeric mode:

character = 0100000010000000B

bit 0-9	character number = 128 address D000:1000H in the pixel memory.
bit 10-14	palette selector = 16: select the colour nibbles 32 and 33 at I/O address 180H + 32.

Graphics mode:

```
    character = 0011010000000000B

    bit 0-9    pixel block number = 0
               address D000:0000H in the pixel memory.

    bit 10-13  palette selector = 13:
               select colour nibbles 26 and 27 at I/O
               address 180H + 26.

    bit 14     resolution select = 0
               select high resolution graphics.
```

Graphics mode:

```
    character = 0100100000010000B

    bit 0-9    pixel block number = 16
               address D000:0200H in the pixel memory.

    bit 10-13  palette selector = 2:
               select colour nibbles 4, 5,  36 and 37 at I/O
               addresses 180H + 4 and 180H + 36.

    bit 14     resolution select = 1
               select medium resolution graphics.
```

4.1.4 Graphics Mode

Graphics mode is selected by outputting the value 0CH to I/O address 76H. Alphanumeric mode is selected by outputting 0DH.

For normal bitmapped graphics, the graphics mode offered by the XIOS is preferred, as this handles all initialization and supports console switching.

4.2 Direct Console Access

The display is normally accessed through the CCP/M Operating System console handling functions (ref.2). In cases where speed has a high priority, the operating system may be bypassed in different ways:

1. Through XIOS Conout entry
2. Through Int-28h function 35
3. Direct manipulation of the display buffer

WARNING!

When the XIOS console driver is accessed directly, the protection offered by the operating system is bypassed, so be sure only to write to consoles that have been attached to the process through a previous operating system call.

4.2.1 XIOS Conout

The XIOS console driver can be accessed directly through a CALLF to the address XIOS_ENTRY found in the SYSDAT area (ref.2) with the following register contents:

```
    AX = 2 (Console output function)
    DS = SYSDAT segment (ref.2)
    ES = UDA segment (ref.2)
    CL = Character to output
    DL = Virtual console number
```

Example

```
; The following subroutine prints a specified number of
; characters on a process's default console. It is assumed
; that DS = SS.
;
;       entry                    exit
; BX    pointer to string        undefined
; CX    length of string         undefined

print_string:
    push     bp                  ; save old stack frame
    push     cx                  ; save length of string
    push     bx                  ; save pointer to string
    mov      cl,153              ;
    int      224                 ; get default console
    mov       def_con,al            ; save default console
    mov       cl,156                ;
    int       224                   ; get PD address

    ; Now use the fact that the PD segment is the same as
    ; the SYSDAT segment

    push      es                    ;
    pop       ds                    ; SYSDAT segment to DS
    mov       es,10HÆbxÅ            ; UDA segment to ES
    pop       bx                    ; get pointer to string
    pop       cx                    ; get length of string
char_loop:
    push      cx                    ; save count
    push      bx                    ; save position
    mov       cl,ss:ÆbxÅ            ; get char from position
    mov       dl,ss:def_con         ; get default console
    mov       ax,2                  ; conout function
    callf     ds:dword ptr .28H     ; callf xios_entry
    pop       bx                    ; get position
    pop       cx                    ; get count
    inc       bx                    ; increment position
    loop      char_loop             ;
    mov       ax,ss                 ;
    mov       ds,ax                 ; get old DS
    pop       bp                    ; get old stackframe
    ret                             ;

def_con       db        0           ; default console number
```

4.2.2 Direct Console Buffer Output

Int-28h function 35 is provided to quickly update large portions of the display. This function stores character strings in the display buffer with the current attribute. If the console is shown in a window, this is automatically updated.

No control character or escape sequence interpretation is done by this routine.

The routine is called with the following register contents:

```
    AL = 35 (function number)
    DX = character position (DH = row, DL = column)
    CX = number of characters in the string
    SI = string address offset
    DS = string address segment
```

Example:

```
; Print the string "RC PICCOLINE" at position (8,20):

    push      DS                      ; save DS
    push      CS                      ; get segment of string
    pop       DS
    mov       SI,offset string_1      ; and offset
    mov       CX,lenght string_1      ; get string length
    mov       DH,8                    ; row number
    mov       DL,20                   ; column number
    mov       AL,35
    int       28h
    pop       DS                      ; restore DS
    ret

string_1      db         'RC PICCOLINE'
```

4.2.3 Display Buffer Manipulation

In some cases it is desirable to manipulate the display buffer directly. For example to dump the screen contents to a file or a printer. It may also be used to modify the screen, e.g. for horizontal scrolling or scrolling part of the screen. Printing to the display buffer is easily done using the Int-28h function 35 (see 4.2.2).

To give a programmer the possibility to manipulate the display, the console driver offers a function that gives access to a table of address offsets to the display line buffers.

Each virtual console is internally represented as 24 (25) display line buffers each describing one character line of the display. A character line consists of one 16 bit word for each of the 80 character positions of the line. Each 16 bit word consists of a character value (low byte) and a set of attribute bits (high byte). Do not use the information in the attribute bytes as the interpretation of these is version dependent.

The address of the table is obtained by means of an Int-28h function with the following register contents:

Registers on entry:

```
    AX = 21
```

Registers on return:

    ES = segment of the table
    BX = offset of the table
    DX = the segment that should be used together with a
         single table entry contents to give the full
         address of one line buffer.

Example

```
; The following routine return a pointer to a specified
; display line buffer.
; At call CX contains the line number (0-24).
; At return ES:SI contains a pointer to the specified
; display line buffer.

get_line_pointer:
    push cx             ; save line number
    mov  ax,21          ; function number
    int  28h            ;

    pop  cx             ; restore line number
    shl  cx,1           ; each table entry is two bytes
    add  bx,cx          ; bx contains offset to display
                        ; line table
    mov  si,es:ÆbxÅ     ; now si contains offset to
                        ; specified display line buffer
    mov  es,dx          ; now es contains segment of
                        ; specified display line buffer
    ret                 ;
```

The screen is automatically updated if the console is in the foreground or when the console is switched to the foreground. If, however, the console is displayed in a window on the screen, the window is not updated when the display buffer is modified.

Instead the window must be updated using an Int-28h function with the following register contents:

    AL = 39

As there are no means to know whether the console is displayed in a window or not, this routine must always be called, if the display buffer is modified.

4.2.4 Get/Set Cursor Position

Another useful console driver function returns the current cursor position.

The cursor position is obtained by means of an Int-28h function with the following register contents:

Registers on entry:

    AL = 22

Registers on return:

    BH = line position of cursor (0-24)
    BL = coloumn position of cursor (0-79)

The cursor position may be changed with an Int-28h function with the following register contents:

    AL = 36
    BH = line (0-23 if 24 line mode, 0-24 if 25 line mode)
    BL = coloumn (0-79)

4.2.5 Get/Set Attribute

The current attribute byte is returned by the following Int-28h function:

    AL = 37

At return register AH contains the attribute byte.

The current attribute byte may be changed by the following Int-28h function:

    AL = 38
    AH = attribute byte

These functions are useful in connection with direct manipulation of the display buffer.

WARNING!

The coding of the attribute byte may be subject to changes in future releases.

4.3 Character Sets

When the display operates in text mode, the character definitions are placed in the 32 Kbytes pixel memory located at address D000:0000H.

The pixel memory has room for 1024 character definitions. As the XIOS handles 8-bit characters, the characters are divided into 4 different character sets:

0 - 255	Lower Standard Character Set
256 - 511	Upper Standard Character Set
512 - 767	Lower Alternative Character Set
768 -1024	Upper Alternative Character Set

The character sets are selected by escape sequences (see 4.4.1):

ESC-P	Select Alternative Character Set
ESC-Q	Select Standard Character Set
ESC-g	Select upper 256 characters
ESC-h	Select lower 256 characters

The default assignment of the alternative character set is as for the standard character set (see App. D). This is convenient when only a few changes from the standard character set are wanted.

When the underline attribute is set, the upper 256 characters of the character set are addressed. So normally the two halves of the character set are identical.

The underline attribute however may be disabled giving a full 512 character alternative character set. This is also true for the standard character set, but other processes may be using the underline attribute, which requires the two halves of the character set to be identical.

If the console is locked (console switching inhibited) and the standard character set restored before termination, all 1024 characters may be altered.

The following escape sequences disables and enables the underline attribute:

ESC-<246>	Disable underline attribute
ESC-<247>	Enable underline attribute

4.3.1 Altering the Character Set

A character definition block consists of 16 words (16 bit memory location), each defining a single video rasterline of the character.

The width of a character is variable from 7 to 15 pixels and is controlled by the contents of the definition for each videoline. The character width is defined by the position of the first zero bit followed by all one's.

Example:

A character 7 bit wide is defined by the following bits:

```
    xxxxxxx011111111B
```

WARNING!

When variable character width is used, it is the responsibility of the programmer to fill the entire line with characters (e.g. by means of variable length space characters).

The standard character width is 7 pixels. The height of one character row is 10 videolines.

When the display operates in graphics mode about 25 Kbytes of the pixel memory is used as bitmap. Consequently the character definitions must be saved each time a process running in graphics mode takes over the display.

This means that the definition may be in one of two places, so the character set cannot be altered simply by modifying the pixel memory. Instead a set of functions is offered in the XIOS.

The functions which are accessed through software interrupt 28h are described in the following.

As the character sets are common to all consoles, it should be insured that only one process is using the alternative character set. This is done by reading the mutual exclusion queue 'MXalt'. The alternative character set is released by a write to the queue.

4.3.2 Define Character Font (Alternative Character Set)

This function defines a character in the alternative character set. The character is defined in both the lower and upper (underlined) character sets.

The function is executed with the following register contents:

```
    AL = 20
    CL = character value (0-255)
    DX = address offset of character definition block
    DS = address segment of character definition block
```

Example

```
; This routine defines the character number 255 in the
; alternative character set.
;
; define_alternative_char:
    mov  dx,offset char_def        ; character definition
    mov  cl,255                    ; character ident
    mov  ax,20                     ; define char. function
    int  28h                       ; call xios function
    ret                            ;

char_def dw    0000000011111111B   ; Character definition.
         dw    0101110011111111B   ;
         dw    0110000011111111B   ; The character is 7 bit
         dw    0100110011111111B   ; wide (As the tail is
         dw    0101000011111111B   ; 011111111).
         dw    0101000011111111B
         dw    0101000011111111B
         dw    0100110011111111B
         dw    0000000011111111B
         dw    0000000011111111B
```

4.3.3 Define Character Font

This function defines a character in the standard or alternative character set.
The function is executed with the following register contents:

```
    AL = 52
    CX = character value (0-1023)
    DX = address offset of character definition block
    DS = address segment of character definition block
```

4.3.4 Get Character Font Definition

This function returns a character definition in the standard or alternative character set.

The function is executed with the following register contents:

```
    AL = 51
    CX = character value (0-1023)
    DX = address offset of character definition block
    DS = address segment of character definition block
```

Example

```
; This routine changes the standard character set to
; US-ASCII

us_ascii:
    mov  cx,9                    ; no. of characters
    mov  si,offset char_table    ; get table address
char_loop:
    push cx
    push si
    lodsw                        ; load an ASCII value
    mov  dx,offset char_buffer
    xchg cl,ah
    push ax
    push dx
    mov  al,51
    int  28h                     ; get the definition
    pop  dx
    pop  ax
    xchg cl,al
    mov  al,52
    int  28h                     ; define the font
    pop  si
    pop  cx
    add  si,2
    loop char_loop                  ; next character
    ret

; the following is a table of characters to alter
; and the corresponding character definitions in
; the standard character set.
;
char_table:
    db   '§',17
    db   'Æ',18
    db   'Ø',19
    db   'Å',20
    db   'Ü',21
    db   'æ',23
    db   'ø',24
    db   'å',25
    db   'ü',26

char_buffer:
    rw   16                         ; room for one definition
```

4.4 Console Control Characters

The XIOS console driver recognizes the following characters as control characters:

Character	Value (decimal)	Meaning
NULL	00	Ignored
BELL	07	Acoustic signal
BS	08	Backspace - Cursor left, if the cursor is at column 0, it is moved to the last position on the previous line.
LF	10	Line feed - Cursor down one row. If the cursor is at the bottom line, the screen is scrolled up one row.
CR	13	Carriage return - move cursor to column 0.
ESC	27	initiate escape sequence (see 4.4.1)

4.4.1 Console Escape Sequences

Escape sequences are used to control the cursor, change colours, programming function keys and various other purposes. An ASCII escape character (dec 27) triggers escape sequence processsing. The character immediately following the escape character indicates which function is to be performed. More characters may follow, depending on the function.

The escape codes and their functions are summarized in Appendix F.

ESC A - Cursor Up

Moves the cursor up one line. If the cursor is already on the top line, this sequence has no effect.

ESC B - Cursor Down

Moves cursor down one line. If the cursor is already at the bottom line, this sequence has no effect.

ESC C - Cursor Forward

Moves the cursor one position to the right. If the cursor is on the rightmost position on the screen, this sequence has no effect.

ESC D - Cursor Backward

Moves the cursor one position to the left. This is a non-destructive move because the characters that the cursor moves over are not erased. If the cursor is in column 0, this sequence has no effect.

ESC E - Clear Screen

Moves the cursor to column 0, row 0 (top-left corner on the screen) and clears the whole screen (filled with blanks).

ESC H - Home Cursor

Moves cursor to colum 0, row 0. The screen is not cleared.

ESC I - Reverse Index

Moves the cursor up one line. If the cursor is on the top line, a scroll down is performed and a blank line is inserted at the top of the screen.

ESC J - Erase to End of Screen
Clears from cursor (including cursor position) to the end of the screen.

ESC K - Erase to end of line

Clears the line, the cursor is on from the cursor position to the end of the line.

ESC L - Insert Line

Inserts a blank line by scrolling the line that the cursor is on and all following lines down one line. The cursor is moved to the beginning of the new line.

ESC M - Delete Line

Moves the cursor to the beginning of the line and deletes the line that the cursor is on by moving all the following lines up one line. A blank line is added at the bottom of the screen.

ESC N - Delete Character

Deletes the character at the cursor position and moves the rest of the line one character position to the left. A blank character is inserted at the end of the line.

ESC O - Insert Character

Inserts a blank character at the cursor position and moves the rest of the line one character position to the right.

ESC P - Select Alternative Character Set

Selects the user definable character set.

ESC Q - Select Standard Character Set

Selects the standard PICCOLINE character set.

ESC Y - Position Cursor

Moves the cursor to the row and column specified by the two characters that follow the "Y". The first character specifies the row, the second specifies the column. Rows are numbered from 0 to 23 (in 24 line mode) or 0 to 24 (in 25 line mode). Columns are numbered from 0 to 79.

The value 32 (20H) is added to the row and column numbers.

Example:

To position the cursor in position (23,79), the sequence is

```
ESC  Y  7 o         dec: 27 89 55 111
                    hex: 1B 59 37 6F
```

ESC b - Set Foreground Colour

The foreground colour displays the character. The colour is specified by a colour selection character, that follows the "b". Only the four least significant bits of the character are used, with the individual bits having the following significance:

Colour Monitor		Monochrome Monitor
0	Blue	
1	Green	
2	Red	
3	High Intensity	2-3 Intensity

Examples of colour select characters:

Colour Monitor	Monochrome monitor
0 - Black	0 - Black
1 - Blue	
2 - Green	
3 - Cyan (Blue + Green)	
4 - Red	4 - Normal Intensity
5 - Magenta (Red + Blue)	
6 - Yellow (Red + Green)	
7 - White (Red + Green + Blue)	
8 - Grey	8 - Low Intensity
9 - High intensity Blue	
: - High intensity Green	
; - High intensity Cyan	
< - High intensity Red	< - High Intensity
= - High intensity Magenta	
> - High intensity Yellow	
? - High intensity White	

NOTE

The monochrome monitor of PICCOLINE is not able to show more than one intensity, - the high intensity. Normal and low intensity are therefore shown as high intensity too. But if the background colour is chosen to have higher intensity than the foreground colour, the foreground is shown as black and the background in high intensity. If the foreground and background have the same intensity they are impossible to distinguish.

At any time 16 combinations of background and foreground colours can be displayed simultaneously. Any escape sequence that would result in more than 16 colour combinations will be ignored.

ESC c - Set Background Colour

This function sets the background colour. The colour is specified by a colour selection character that follows the "c". The colour selection character is interpreted in the same way as described under ESC-b (Set Foreground Colour).

NOTE

At any time 16 combinations of background and foreground colours can be displayed simultaneously. Any escape sequence that would result in more than 16 colour combinations will be ignored.

ESC d - Erase Beginning of Screen

Clears the screen from the home position (0,0) to the cursor position, including the character that the cursor is on.

ESC e - Enable Cursor

This sequence causes the cursor to be visible on the screen.

ESC f - Disable Cursor

This sequence causes the cursor to be invisible. The cursor may still be moved on the screen.

ESC g - Enter Underline Mode

Following the invocation of this sequence, characters are displayed underlined if the underline attribute is enabled (see ESC-<247>).

This sequence also selects the upper 256 characters of the character set.

ESC h - Exit Underline Mode

Exits underline mode.

This sequence also selects the lower 256 characters of the character set.

ESC i - Enter Non-Displayed Mode

This sequence causes characters to be displayed as blanks.

ESC j - Save Cursor Position

This sequence saves the current cursor position. The cursor can be restored to the saved position with ESC-k.

ESC k - Restore Cursor Position

This sequence restores the cursor to a previously saved position. If this sequence is used without a previously saved cursor position, then the cursor will be moved to the home position (0,0).

ESC l - Erase Line

Clears the entire line that the cursor is on.

ESC m - Enable Cursor

Included to be compatible with some CP/M-86 implementations. Use ESC-e under CCP/M-86.

ESC n - Disable Cursor

Included to be compatible with some CP/M-86 implementations. Use ESC-f under CCP/M-86.

ESC o - Erase Beginning of Line

Clears the start of the line to the cursor position, including the cursor position.

ESC p - Enter Reverse Video Mode

Following the invocation of this sequence, the foreground and background colours are reversed. If display is already in reverse video mode, this sequence has no effect.

In reverse video mode, setting foreground colour will effectively set the background colour.

NOTE

> At any time 16 combinations of background and foreground colours can be displayed simultaneously. Any escape sequence that would result in more than 16 colour combinations will be ignored.

ESC q - Exit Reverse Video Mode

Exits the reverse video mode.

ESC r - Enter Intensify Mode

Following the invocation of this sequence, characters are displayed in high intensity.

In reverse video mode the background will be intensified.

NOTE

> At any time 16 combinations of background and foreground colours can be displayed simultaneously. Any escape sequence that would result in more than 16 colour combinations will be ignored.

ESC s - Enter Blink Mode

Causes characters to be displayed blinking.

ESC t - Exit Blink Mode

Causes characters to be displayed not blinking.

ESC u - Exit Intensify Mode

Causes characters to be displayed in normal intensity.

ESC v - Wrap at End of Line

Causes the cursor to move to the beginning of the next line if a character is written in the rightmost position of the line. If at the bottom line, the screen is scrolled up one line.

ESC w - Discard at End of Line

Following the invocation of this sequence, if a character is written in the rightmost position of the line, the cursor remains in the same position. The following characters overprint.

ESC x - Exit Non-Displayed Mode

This sequence causes characters to be displayed normally.

ESC z - Reset Attributes

This sequence turns off the attributes blinking, underline, high intensity, non-displayed mode to the off condition. The background colour is set to black and the foreground to the default colour. Also, cursor is enabled, standard character set is selected, wrap at end of line enabled, function keys are expanded normally, and the status line is enabled (24 line mode).

ESC 0 - Status Line Off (25 Line Mode)

This sequence turns off the status line, thereby leaving all 25 lines for the application.

ESC 1 - Status Line On (24 Line Mode)

This sequence displays the status line at the bottom of the screen, thereby leaving 24 lines for the application.

ESC 2 - Save Current Attributes

Saves the values of the attributes blinking, underline and reverse video, foreground and background colour and character set selection.

ESC 3 - Restore Attributes

Restores the previously saved values of the attributes blinking, underline and reverse video, foreground and background colour and character set selection.

ESC 6 - Function Key Expansion Off

Causes the programmable function keys to return their key identifiers (ref. ESC-:) with the high order bit set instead of the assigned strings.

ESC 7 - Function Key Expansion On

Enables normal function key expansion, so that the programmable function keys return their assigned strings.

ESC : - Program Function Keys

This sequence programs the programmable function keys. The table below lists the keys that are programmable.

The format of this escape sequence is:

```
    ESC  :  <key-id>  <string>  NULL
```

<key-id> is a key identifier that specifies the key to be programmed (see table page 50). <string> is an arbitrary

string of characters; for the F1-F12 keys used alone, strings can be up to 20 characters long. For the remaining function keys, strings can be up to 4 characters. NULL is a character with value 0, that terminates the string.

With the function key expansion disabled by ESC-6, the function keys return the hexadecimal value of the function key identifier with the high order bit set. ESC-7 restores the normal expansion of function keys.

The key identifiers are shown in table 4.1.

Identifier	Function Key	Identifier	Function Key
;	F1	a	alt-F1
<	F2	b	alt-F2
=	F3	c	alt-F3
>	F4	d	alt-F4
?	F5	e	alt-F5
É	F6	f	alt-F6
A	F7	g	alt-F7
B	F8	h	alt-F8
C	F9	i	alt-F9
D	F10	j	alt-F10
E	F11	k	shift-F1
F	F12	l	shift-F2
G	Home	m	shift-F3
H	Up Arrow	n	shift-F4
I	A1	o	shift-F5
J	A2	p	shift-F6
K	Left Arrow	q	shift-F7
L	Return (keypad)	r	shift-F8
M	Right Arrow	s	shift-F9
N	A3	t	shift-F10
O	A4	u	ctrl-F1
P	Down Arrow	v	ctrl-F2
Q	Tab (keypad)	w	ctrl-F3
R	Insert	x	ctrl-F4
S	Delete	y	ctrl-F5
T	Print	z	ctrl-F6
U	shift-A1	æ	ctrl-F7
V	shift-A2	ø	ctrl-F8
W	shift-A3	å	ctrl-F9
X	shift-A4	ü	ctrl-F10
Y	alt-F11		
Z	alt-F12		
Æ	shift-F11		
Ø	shift-F12		
Å	ctrl-F11		
Ü	ctrl-F12		

Table 4.1. Function Key Identifiers

Example:

The following sequence gives function key F2 the value "PICCOLINE":

```
         ESC  :  <  PICCOLINE  NULL

    Dec: 27   58  60  80  73  67  67  79  76  73  78  69  00
    Hex: 1B   3A  3C  50  49  43  43  4F  4c  49  4E  45  00
```

The contents of the function keys will remain valid until the program that defined the keys, is terminated. After the program has terminated the function keys will regain their default values. The default values are common to all the virtual consoles. To change the default assignment use the FUNCTION program.

ESC < - Scroll Window Up

Scrolls a window consisting of a number of consecutive lines one row up. A blank row is inserted at the bottom of the window.

The format of the sequence is:

```
    ESC  <  row-start  row-end
```

Rows are numbered from 0 to 23 (in 24 line mode) or 0 to 24 (in 25 line mode). The value 32 (20H) is added to the row numbers.

Example:

The following sequence scrolls row 4 to row 11 one line up:

```
    ESC  <  $  +                    (dec: 27  60  36  43)
                                    (hex: 1B  3C  24  2B)
```

ESC > - Scroll Window Down

Scrolls a window consisting of a number of consecutive lines one row down. A blank row is inserted at the top of the window.

The format of the sequence is:

```
    ESC  >  row-start row-end
```

Rows are numbered from 0 to 23 (in 24 line mode) or 0 to 24 (in 25 line mode). The value 32 (20H) is added to the row numbers.

Example:

The following sequence scrolls row 4 to row 11 one line down:

```
    ESC  >  $  +                  (dec: 27  62  32  48)
                                  (hex: 1B  3E  20  30)
```

ESC <241> - Set Blinking Cursor

<241> denotes one character with the decimal value 241. Selects a blinking cursor.

ESC <242> - Set Non-Blinking Cursor

<242> denotes one character with the decimal value 242. Selects a non-blinking cursor.

ESC <243> - Set Cursor Representation

<243> denotes one character with the decimal value 243.

Defines the shape of the cursor. The character following ESC-243 specifies the start and end videoline numbers of the cursor. The four least significant bits specifiy the start videoline and the four most significant bit specifiy the end videoline.

The videolines of a row are numbered 0-9. The number specified for the end videoline is 1 greater than the videoline number of the bottom videoline of the cursor.

Examples:

The following sequence selects a block cursor (occupying videolines 0-9):

```
    dec: 27  243  160             hex:  1B     F3    A0
```

The following sequence selects a double underline cursor (occupying videolines 8-9):

```
    dec: 27  243  168             hex:  1B     F3    A8
```

ESC <244> - Set Soft Scroll

<244> denotes one character with the decimal value 244.
Selects soft scroll mode.

ESC <245> - Set Line Scroll

<245> denotes one character with the decimal value 245.
Selects line scroll mode.

ESC <246> - Disable Underline Attribute

<246> denotes one character with the decimal value 246.
Following the invocation of this escape sequence, the underline attribute is disabled.

As the upper 256 characters of the character sets are addressed when the underline attribute is on, the lower and upper 256 characters must be identical in normal uses of the underline attribute. Disabling underline makes it possible to use all 512 characters of the character set.

The escape sequences ESC-g and ESC-h are used to select the upper and lower 256 characters respectively.

ESC <247> - Enable Underline Attribute

<247> denotes one character with the decimal value 247.
Following the invocation of this escape sequence, the underline attribute is enabled.

The escape sequences ESC-g and ESC-h are used to enter and exit underline mode.

ESC <253> - Save Function Keys

<253> denotes one character with decimal value 253.
Saves the current value af the programmable function keys.

NOTE

Only one set af values can be stored.

ESC <254> - Restore Function Keys

<254> denotes one character with decimal value 254.
Restores the saved value of the programmable function keys (see ESC 253).

4.5 Graphics Mode

A function is offered in the XIOS to put a console into graphics mode.

When this function is used, the console module handles transitions between alphanumeric and graphics mode when a console is switched from foreground to background and vice versa. At the same time the console module saves or restores the graphic image on the screen. It also supports console output in graphics mode.

In graphics mode the bitmap for the display occupies the 32k pixel memory. The character definitions therefore have to be saved in a save-buffer elsewhere in memory. The graphics save-buffer must be provided by the application program.

When a console is switched in or out, the console module swaps the contents of the pixel memory and the save-buffer. The application program must provide a variable in which the console module places a pointer to the segment that currently contains the graphic image.

To avoid swapping the segments while the graphics segment is being updated, a semaphore is used to ensure exclusive access to the graphics segment.

NOTE

> In a PICCOLINE 1-console system (i.e. no background consoles) the save buffer will allways contain the character definitions. In order to save memory space only the lower standard character set is saved in the save-buffer. Therefore in this case the save-buffer need only be of size 8k. This also means that only the lower standard character set may be used in graphics mode.

4.5.1 Init Graphics

Graphics mode is entered by an Int-28h function called with the following register contents:

```
    AL = 0    (function number)
    AH = graphics mode  (1 = high resolution)
                        (2 = medium resolution)
    CX = address offset of graphics control block
    DX = address segment of graphics control block
```

The graphics control block has the following format:

```
gcb:
gcb_mx   db   0          ; mutual exclusion semaphore
gcb_seg  dw   seg buffer; segment of savebuffer
```

The gcb_seg field must contain the address segment of a 32k save buffer (8k in case of a 1-console system).

When the gcb_mx field is set to 255 (FFH), the console will not be switched in or out, thus avoiding buffer swapping when the graphics segment is being updated. As the PIN process may be waiting for this semaphore, it should not be set for a longer period.

Example

```
; this routine puts the console in graphics mode.
; it is assumed that the ES register points at the
; extra segment.

init_graphics:
    mov  gcb_seg,es      ; initialize the buffer segment
    mov  al,0            ; function code for init graphics
    mov  ah,1            ; high resolution
    mov  cx,cs:offset gcb        ; get offset and segment
    mov  dx,cs           ;   of the control block
    int  28h             ; do the call
    ret
;
gcb      rb   0          ; graphics control block
gcb_mx   db   0
gcb_seg  rw   1
;
    eseg
buffer   rb   8000h      ; make room for save buffer
                         ; 2000h in case of 1-console system
```

4.5.2 Exit Graphics

Alphanumeric mode is entered by an Int-28h function called with the following register content

    AL = 1

4.5.3 Exclusive Access to Pixel Memory

Exclusive access to the graphics image can be ensured in a number of ways.

1. Use the CHSET utility program to stop program execution while the console is in the background (Suspend= On). This is probably the easiest solution.

2. Inhibit console switching by setting the no-switch bit in the console control block.

3. Disable interrupts while updating the image. Should only be used for very short updates.

4. Use the mutual exclusion semaphore located in the graphics control block. A pointer to the graphics control block is rendered to the XIOS in the init graphics call.

Example

```
; this routine sets the no-switch bit in the console
; control block. The keep flag is set in the process
; descriptor so we cannot be terminated before the
; no-switch bit has been cleared.

os             equ      224
sys_ccb        equu     word ptr 54h
p_flag         equ      word ptr 6
pf_keep        equ      2
ccb_size       equ      2ch
ccb_state      equ      word ptr 14
cf_noswitch    equ      8

lock_console:
    mov  cl,156          ; get process descriptor
    int  os
    mov  sysdat,es
    mov  pd_addr,bx      ; set the keep flag
    or   es:p_flagÆbxÅ,pf_keep
                         ; now get the ccb address
    mov  cl,153          ; get console no.
    int  os
    cbw
    mov  cx,ccb_size
    mul  cx
    add  ax,es:sys_ccb
    mov  ccb_addr,ax
    xchg ax,bx          ; set the noswitch flag
    or   es:ccb_stateÆbxÅ,cf_noswitch
    ret

unlock_console:
    mov  es,sysdat
    mov  bx,ccb_addr    ; clear noswitch bit
    and  es:ccb_stateÆbxÅ,not cf_noswitch
    mov  bx,pd_addr     ; and turn off keep flag
    and  es:p_flagÆbxÅ,not pf_keep
    ret

sysdat   rw   1
pd_addr  rw   1
ccb_addr rw   1
```

Example

```
; this routine demonstrates the use of the mutual exclusion
; semaphore.
; it is assumed that an init graphics call has been made
; and the gcb_seg field initialised.

clear_graphics:
    call get_mx         ; get the semaphore
    mov  es,gcb_seg
    mov  di,0
    mov  ax,0           ; fill with zero's
    mov  cx,4000h       ; 16k words
    rep  stosw
    call free_mx        ; release the semaphore
    ret

get_mx:
    mov  al,0ffh
    xchg al,gcb_mx
    or   al,al          ; is it free?
    jz   got_mx         ; yes - we have it
    push ax             ; no - delay one tick
    push bx             ; save what has to be saved
    mov  cl,141
    mov  dx,1           ; one tick delay
    int  224
    pop  bx
    pop  ax             ; restore saved registers
    jmps get_mx         ; and try again
got_mx:
    ret
```

```
free_mx:
    mov  gcb_mx,0
    ret

gcb_mx   db   0
gcb_seg  rw   1
```

4.5.4 Pixel Address Calculation

The following example shows how the offset in the pixel memory and the bit number is calculated given an X-Y coordinate. The origin is assumed to be in the lower left hand corner.

Example

```
;   entry:     BX = X-coordinate
;              DX = Y-coordinate
;
;   exit:      DI = offset of word containing pixel
;              BX = bit mask
;
;   Algoritm used:
;   word_address   = (X div 16) * 16*16 + Y
;   pixel address = X mod 16

y_max    equ  255
color    equ  false        ; set to true if assembling to
                           ; medium resolution

calc_pixel_addr:
    mov  ax,y_max          ;get the Y size
    sub  ax,dx             ;move 0 for the Y axis
if color
    shl  bx,1
endif
    mov  di,bx             ; save X value
    and  bx,0fff0h         ; mask out bit number
    mov  dx,bx             ;
    mov  cl,4
    shl  bx,cl             ; BX * 16
    add  bx,ax             ; add in Y-addr
    shl  bx,1              ; byte addr
    and  di,0fh            ; mask to pixel address
    xchg bx,di
```

```
if not color
    shl  bx,1
endif
    mov  bx,bit_masksÆbxÅ
    ret

if not color
bit_masks     Dw        1000000000000000B
              Dw        0100000000000000B
              Dw        0010000000000000B
              Dw        0001000000000000B
              Dw        0000100000000000B
              Dw        0000010000000000B
              Dw        0000001000000000B
              Dw        0000000100000000B
              Dw        0000000010000000B
              Dw        0000000001000000B
              Dw        0000000000100000B
              Dw        0000000000010000B
              Dw        0000000000001000B
              Dw        0000000000000100B
              Dw        0000000000000010B
              Dw        0000000000000001B
else
bit_masks     Dw        1100000000000000B
              Dw        0011000000000000B
              Dw        0000110000000000B
              Dw        0000001100000000B
              Dw        0000000011000000B
              Dw        0000000000110000B
              Dw        0000000000001100B
              Dw        0000000000000011B
endif
```

4.6 Window Handling

The Concurrent CP/M Windows are handled by a number of XIOS routines. The routines are called through the normal XIOS entry point.

Some of the routines are used only by the standard window manager, the rest may be of interest to the application programmer. They are described in the following sections.

NOTE

In a PICCOLINE 1-console system it makes no sense to use windows. Therefore, in order to save memory space, the routines are removed. A call of XIOS function number 19 results in an error message in the status line, whereas call of the other functions has no effect.

4.6.1 Return Pointers

This funtion return pointers to two different data structures.

A pointer to the window manager data block is returned by the following call:

```
    entry:     al = 16
               dl = 0FFH
    exit:      ax = window data block pointer
```

The window data block has the following format:

```
state     rb   1          ; window manager state
                          ; 0 = not resident
                          ; 1 = resident but not active
                          ; 2 = resident and active
nvc       db   nvcns      ; number of virtual consoles
priority rb    nvcns      ; list of console numbers from the
                          ; back window to the front window
```

If register DL is a virtual console number the call is similar to Int-28h function 21 (see 4.2.3).

```
    entry:     al = 16
               dl = virtual console number

    exit:      ax = vc structure pointer
               dx = screen segment
               es = vc structure segment
```

The call returns a pointer to a control structure of the following format:

```
        rw   26        ; display line table (see 4.2.3)
        rw   1         ; extra line used when scrolling
        rb   1         ; virtual console number
        rb   1         ; internal XIOS semaphore
        rb   1         ; left column of window
        rb   1         ; top row of window
        rb   1         ; rigth column of window
        rb   1         ; bottom row of window
        rw   1         ; last top-left corner
        rw   1         ; last bottom-right corner
        rb   1         ; actual no. of columns
        rb   1         ; actual no. of rows
        rb   1         ; window view point, column
        rb   1         ; window view point, row
```

4.6.2 Set Window Manager State

This call is used to tell the XIOS the state of the window manager and to change which window is on top (console switch).

```
    entry:    al = 19
              cl = state
              0 => manager not resident
              1 => resident but not active
              2 => resident and active
              3 => leave state unchanged

              dl = vc number to switch to top
              if dl = 0FFH, then no switch

    exit:     none
```

4.6.3 Create a New Window

This call is used to create a new window for a virtual console. The positions of the windows top-left and bottom-right corners on the screen are passed as parameters.

```
    entry:    al = 20
              dl = virtual console number
              cx = top left (row,column)
              bx = bottom right (row,column)
```

4.6.4 Set Cursor Tracking Mode and Viewpoint

This call sets the tracking mode and viewpoint. The tracking mode determines whether the window is fixed or follows the cursor. The viewpoint determines which part of the virtual console is visible in the window.

```
    entry:    al = 21
              dl = vc number
              dh = cursor tracking mode
                   0 => window is fixed on vc image
                   1 => window tracks scrolling
              cx = row,column of top-left viewpoint

    exit:     none
```

4.6.5 Set Wrap Around Column

This call sets the column in which the cursor automatically wraps around if wrap around is enabled.

```
    entry:    al = 22
              dl = vc number
              cl = wrap column number

    exit:     none
```

4.6.6 Switch Between Full Screen and Window

This call toggles the window between full screen and not full.

```
    entry:    al = 23
              dl = vc number

    exit:     none
```

4.7 Keyboard Interface

The keyboard is connected to the system via a special serial port with I/O address 32 (20H). When a character is received, an interrupt is generated, and no further characters will arrive before the character is read.

The interrupt is connected to level 1 of the external interrupt controller 8259A (interrupt level 21H, interrupt vector address 84H).

When a key is pressed, an 8-bit position code is received and when the key is released, the keyboard sends the same code with the high order bit set. The position codes are shown in Appendix E.

4.7.1 Keyboard Driver

In normal applications the XIOS keyboard driver handles all input from the keyboard. The driver converts the position codes into ASCII values and handles special keys (Ctrl, Alt, Shift, Shift Lock and programmable function keys).

The RC739 keyboard includes 98 keys of which 26 are programmable. The values returned by the keyboard driver when a key or combination of keys is pressed, are shown in appendix D.

When a programmable function key is pressed, the driver returns the programmed string of characters. The function keys are programmed by the escape sequence ESC-: (see 4.4.1).

The following key combinations invoke special actions in the driver and no value is returned to the application:

Ctrl+Print	hardcopy of display (in character mode)
Ctrl+A1	enter setup mode
Ctrl+A2	no action
Ctrl+A3	wake up window manager (if installed)
Ctrl+A4	full screen key

4.8 Mouse Interface

The optical mouse is supported by an Int-28h function. This function is called with the following register contents:

```
AL = 30
CL = mouse function number
```

Three mouse functions numbers are provided:

```
CL = 1 : Initialize mouse
CL = 2 : Deinitialize mouse
CL = 3 : Return mouse status
```

When function 3 (return mouse status) is called, the mouse status is returned in the following registers:

Registers on return:

AL = 0: nothing happened

AL = 1: button press

register AH contains a button code:

```
AH = 20H: left button
AH = 21H: middle button
AH = 22H: right button
```

AL = 2: coordinate information

registers BX and CX contain the change in coordinates since the last call of mouse status.

```
BX = delta x
CX = delta y
```

5. Real Time Clock

The PICCOLINE standard configuration includes a real time clock controller (RTC) with battery backup.

The RTC time and date information is read during power up and is used to initialize the time and date fields found in the SYSDAT area (see ref.2).

After power up the RTC generates an interrupt each second and this interrupt is used to update the above mentioned SYSDAT fields.

If a program disables interrupts for more than one second, it will cause a loss of one or more interrupts from the RTC. As a consequence, the time and date fields will not be updated correctly (but the real time clock itself still holds the correct time and date).

5.1 Real Time Clock Controller

RTC controllers from two different manufacturers are used in the PICCOLINE. To distinguish between the two types refer to the KONFIG area byte 'RTC second source' (see 3.1). If this byte is 0 the real time clock is a National Semiconductor chip: MM58167. If the byte has the value 0FFH the real time clock is an RCA chip: CDP1879.
The two real time clock controllers differs in programming and in facilities. A detailed description may be found in the documentation from the manufacturers.

5.2 Reading and Writing Real Time Clock Registers

Although the two RTC controllers are different, they are interfaced in a way, that makes it possible to read and write their control registers using the same software routines. The content of the RTC control registers are coded in BCD-code, which means that a number is stored with the the first ciffer in the four MSB and the last ciffer in the four LSB. For example the number 35 is stored as:

bit no	7	6	5	4	3	2	1	0
	0	0	1	1	0	1	0	1

where bit 0-3 contain the ciffer 5 and bit 4-7 contain the ciffer 3.

The RTC registers are numbered as follow:

RTC control register	
sec	2
min	3
hour	4

The following example shows two routines which can be used to read and write the RTC control registers.

Example

```
    ; Registers at entry:
    ;   AL = RTC register
    ;
    ; Registers at exit
    ;   AL = contents of RTC register
    ;

    rtc_adr             equ        5CH
    read_adr_set_up     equ        80H
    supply_read_pulse   equ       0A0H
    remove_read_pulse   equ        9FH

ReadRTC:

    ; read register address setup
    Mov  DX,rtc_adr
    Or   AL,read_adr_set_up
    Out  DX,AL

    ; generate read pulse
    Or   AL,supply_read_pulse
    Out  DX,AL

    ; Wait at least 1 micro sec
    Nop
    Nop
    Nop
    Nop

    ; read from register
    Xchg AH,AL
    In   AL,DX
    Xchg AH,AL
    ; remove read pulse
    And  AL,remove_read_pulse
    Out  DX,AL
    Xchg AH,AL
    Ret


WriteRTC:

    ; Registers at entry:
    ;   AL = RTC register
    ;   AH = value
    ;

    write_adr_set_up    equ         1FH
    supply_write_pulse  equu        40H
    remove_write_pulse  equ         1FH


    ; write register address setup
    Mov  DX,rtc_adr
    And  AL,write_adr_set_up
    Out  DX,AL

    ; write value to register
    Sub  DX,2
    Xchg AH,AL
    Out  DX,AL
    Xchg AH,AL
    Add  DX,2

    ; generate write pulse
    Or   AL,supply_write_pulse
    Out  DX,AL

    ; wait at least 1 micro sec
    Nop
    Nop
    Nop
    Nop

    ; remove write pulse
    And  AL,remove_write_pulse
    Out  DX,AL
    Ret
```

6. Sound

The sound device produces sound via the loudspeaker located in the CPU unit (some CRT units have a loudspeaker).

The sound device contains four signal sources: three independent generators of single-frequency tones and one generator of noise. In addition, each source has its own attenuator with a 28-dB attenuation range. The output signal from the four attenuators are summed together as a single amplified output.

The sound device contains 8 registers that control the various noise and tone outputs:

R0	R1	R2	Control register
0	0	0	tone 1 frequency
0	0	1	tone 1 attenuation
0	1	0	tone 2 frequency
0	1	1	tone 2 attenuation
1	0	0	tone 3 frequency
1	0	1	tone 3 attenuation
1	1	0	noise control
1	1	1	noise attenuation

R0, R1 and R2 denote bit positions in the control bytes sent to the sound device as described below.

Noise and attenuation parameters are sent to the sound device as 1-byte values, while frequence updates require 2 bytes. To differentiate between the first and second byte of any data transfer, all first-byte or single-byte transfers have the most significant bit equal to a logic 1. The second byte always has the MSB equal to logic 0.

Because the CCP/M operating system does not support such exotic devices as sound generators, this device is accessed through Int-28h function 12:

```
AX = 12
DL = sound device control byte
```

To prevent more than one program from using the sound device at the same time, the programs should reserve the device before using it. This is done with the help of a mutual exclusion queue of the name 'MXsound'. The device is reserved when a queue read from MXsound succeeds.

Example

```
; This piece of code reserves the sound device by reading
; the mutual exclusion queue 'MXsound '.

    mov  cl,135                   ; queue open function
    mov  dx,offset qpb_sound      ; queue parameter block
    int  224                      ;
    mov  cl,137                   ; queue read function
    mov  dx,offset qpb_sound      ; queue parameter block
    int  224                      ;

; the process will not  proceed  before the sound device is
reserved.


qpb_sound      dw        0,0,0,0
               db        'MXsound '
```

After use, the program should release the device as follows:

```
; This piece of code releases the sound device by writing
; to the mutual exclusion queue 'MXsound '.

    mov  cl,139                   ; queue write function
    mov  dx,offset qpb_sound      ; queu parameter block
    int  224                      ;
```

6.1 Programming Tones

Each of the three tone generators cover a range of five octaves: from two octaves below middle C to three octaves above it.

Setting a frequency of 440 Hz for tone generator 1 is done as follows.

First find I

    I = clock rate/(32 * f)
    I = 2 MHz/(32 * 440)
    I = 142.045

Since 'I' must be an integer quantity set it to 142. The actual frequency will be 440.14 Hz.

Next, convert 'I' to a 10-bit binary value:

```
    F0 F1 F2 F3 F4 F5 F6 F7 F8 F9
     0  0  1  0  0  0  1  1  1  0
```

The frequency data for tone generator 1 must be transferred as a 2-byte quantity. The formats of the 2 frequency control bytes are as follows:

```
    byte 1:  1 R0 R1 R2 F6 F7 F8 F9
    byte 2:  0 x  F0 F1 F2 F3 F4 F5    (x = don't care)
```

To address tone register 1 R0, R1 and R2 must be 000. Therefore, to set tone generator 1 at 440 Hz, the first control byte becomes

```
    1 0 0 0 1 1 1 0
```

and the second byte becomes

```
    0 0 0 0 1 0 0 0
```

Once these values have been transferred, tone generator 1 is loaded, but the attenuator has not been set to enable any output. Changing the attenuator setting requires only a single byte of data:

```
    1 R0 R1 R2 A0 A1 A2 A3
```

R0, R1 and R2 address the register as mentioned before, while A0 - A3 determine the attenuation as follows.

A0	A1	A2	A3	Attenuation weight
0	0	0	0	0 dB
0	0	0	1	2 dB
0	0	1	0	4 dB
0	0	1	1	6 dB
0	1	0	0	8 dB
0	1	0	1	10 dB
0	1	1	0	12 dB
0	1	1	1	14 dB
1	0	0	0	16 dB
1	0	0	1	18 dB
1	0	1	0	20 dB
1	0	1	1	22 dB
1	1	0	0	24 dB
1	1	0	1	26 dB
1	1	1	0	28 dB
1	1	1	1	off

A 0 dB setting turns the volume on full. The resulting formatted control byte is

    1 0 0 1 0 0 0 0

Example

```
; The following subroutine simulates the
; ringing of bells.

chime:
    call silence        ;
    mov  dl,140         ;
    call wsg            ; tone 1 = 679 Hz
    mov  dl,5           ;
    call wsg            ;
    mov  dl,170         ;
    call wsg            ; tone 2 = 694 Hz
    mov  dl,5           ;
    call wsg            ;
    mov  t,-1           ;
cloop1:                 ; strike chime 12 times
    cmp  t,12           ;
    jz   cloop_exit     ;
    inc  t              ;
    mov  va,144         ;
cloop2:                 ; step attenuation
    inc  va             ;
    cmp  va,160         ;
    jz   cloop1         ;
    mov  dl,va          ;
    call wsg            ;
    mov  dl,va          ;
    add  dl,32          ;
    call wsg            ;
    mov  cx,0A000H      ;
cloop3:                 ;
    loop cloop3         ; step delay
    jmp  cloop2         ;
cloop_exit:
    ret                 ;

silence:                ; shut off:
    mov  dl,9FH         ; tone generator 1
    call wsg            ;
    mov  dl,0BFH        ; tone generator 2
    call wsg            ;
    mov  dl,0DFH        ; tone generator 3
    call wsg            ;
    mov  dl,0FFH        ; noise generator
```

```
wsg:                          ; write to sound
    mov  ax,12                ; device
    int  28H                  ;
    ret


t        db   0
va       db   0
```

6.2 Programming Noise

The noise generator produces pseudorandom noise by means of a shift register. The rate at which the register shifts determines whether the noise contains a majority of high-frequency or low-frequency components.

To change the output of the noise source, change the noise-control and noise-attenuation registers. Both use single-byte commands with the following format.

    1 R0 R1 R2 x FB NF0 NF1

The FB bit controls the feedback in the noise-generator shift register. If the FB bit is a logical 1, the result is white noise. If the FB bit is a logical 0, the feedback is disabled, and a lower-frequence periodic noise is produced.

Two bits, NF0 and NF1, control the clock frequency fed to the noise-generator shift register. Four options are available:

Three options select fixed rates, the fourth selects the output from tone generator 3 as the noise generator shift register clock.

NF0	NF1	Shift rate
0	0	clock rate/512
0	1	clock rate/1024
1	0	clock rate/2048
1	1	tone generator 3 output

where clock rate = 2 MHz

Example

```
; The following subroutine generates an
; explosion sound
```

```
explosion:
    call silence        ;
    mov  dl,0E4H        ; set high pitched
    call wsg            ; white noise
    mov  va,0EFH        ;
eloop1:
    inc  va             ; step attenuation
    jz   eloop_exit     ;
    mov  dl,va          ;
    call wsg            ;
    mov  cx,0FFFFH      ; step delay
eloop2:
    loop eloop2         ;
    jmp  eloop1         ;
eloop_exit:
    ret                 ;

silence:                ; shut off:
    mov  dl,9FH         ; tone generator 1
    call wsg            ;
    mov  dl,0BFH        ; tone generator 2
    call wsg            ;
    mov  dl,0DFH        ; tone generator 3
    call wsg            ;
    mov  dl,0FFH        ; noise generator

wsg:                    ; write to sound
    mov  ax,12          ; device
    int  28H            ;
    ret                 ;


va       db   0
```

7. Cassette Tape

The PICCOLINE standard configuration includes a cassette tape interface. The interface control is implemented in software by using Timer 0 of the 80186 to control the data of the cassette recorder.
This chapter describes the interface to the cassette tape. The plug definition of the interface can be found in ref.5 (appendix D).

7.1 Cassette Tape Control

The cassette tape control is implemented in software in the following way.
Timer 0 output from the 80186 is used to control the output data to the cassette recorder. The method used in the existing driver is to set Timer 0 to the period of the desired data bit. The timer is set to a period of approximately 1 millisecond for an one bit and 0.5 millisecond for a zero bit. The timer then outputs a square wave with the period given by the count register (see fig. 7.1). When more data bits are written the period of the timer is changed on the fly.

A detailed description of how to program Timer 0 can be found in the Intel documentation of the 80186 CPU.

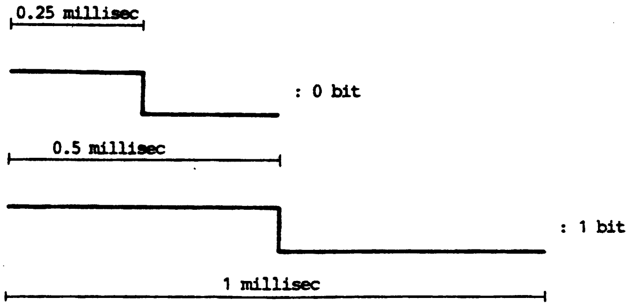


Fig.7.1 Square wave of data bits.

Cassette input data is read on the I/O-address 70H (112) bit 0. A one bit read corresponds to a high pulse of the data bit and a 0 bit read to a low pulse (see fig.7.1). By continously reading the pulse value it is possible to determine the period of the square wave and thereby to determine the value of the data bit. Timer 0 is used for measuring the pulse width of the square wave.

Note

> Since both reading from and writing to the cassette tape is very time dependent it may be necessary to turn off all interrupts while reading or writing.

The casssette drive motor is controlled (on/off) by I/O-address 76H (118), see fig.7.2.
Cassette input and output are enabled or disabled by I/O-address 76H (118). This can be very usefull for instance while accelerating or descelerating the cassette drive motor.

I/O address		Output	Operation
hex	dec	value	
76	118	2	Motor on
76	118	3	Motor off
76	118	1	Enable cassette
76	118	0	Disable cassette

Fig.7.2 Cassette Control.

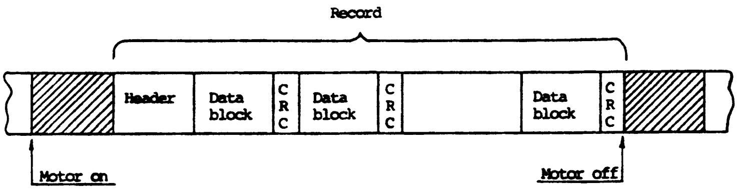
7.2 Cassette Tape Driver

Because the CCP/M operating system does not support a cassette tape this device is accessed through Int-28h function calls.

Before describing the driver functions the general format of the data written on the cassette tape is described.

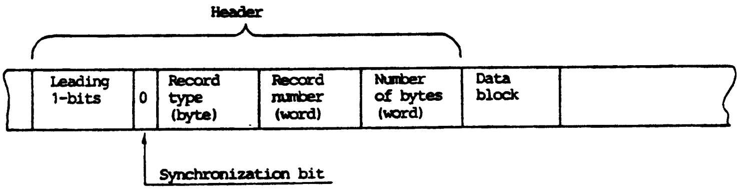
7.2.1 Data Record Architecture

All data written on the cassette tape are written as cassette records. A cassette record consist of following entries:



where each data block contains 128 bytes of data, and the CRC field is a two byte CRC check. The last data block is extended to 128 bytes.
The delay after start of motor is approximately 1.5 seconds.

The header field consist of following entries:



where the leader consist of 2024 ones (used to eliminate problems caused by a slow accelerating cassette motor). The synchronization bit tells when the leader is finished. The record type tells what kind of record this is,- either "file header", "data record" or "end of file".

A data file usually consist of conseqtutive data records. Therefore to keep track of the records a record number is saved in the header. Finally the header contains a field containing the number of bytes in this record.

The normal format of a file written on cassette tape is: First a file header record describing the file (name, type,..). Then a number of data records and finally an end of file record.
Following five Int-28h functions can be used for reading and writing files of the kind mentioned above.

7.2.2 Cassette Write File Header

This function writes data from the specified output buffer into a file header record on the cassette tape.

Registers on entry:

AL = 27
CX = Number of bytes in output buffer
DX = Offset address to output buffer
Topmost element on stack contains segment register of output buffer.

Registers on return:

AX = 0
CX = 0
DX = Offset (in output buffer) to the byte after the last one written
BX is modified.
Stack is unchanged.

7.2.3 Cassette Write Next Data

This function writes data from the specified output buffer into a data record on the cassette tape.

Registers on entry:

AL = 29
CX = Number of bytes in output buffer
DX = Offset address to output buffer
Topmost element on stack contains segment register of output buffer.

Registers on return:

```
AX = 0
CX = 0
DX = Offset (in output  buffer)  to  the byte after the
     last one written
BX is modified.
Stack is unchanged.
```

Note

The cassette driver itself assigns a record number to the record. A file header record is allways assigned record number 0. The succeeding data records are then assigned record number 1, 2, 3,... etc. That is, in order to get a proper record numbering, data records have to be preceeded by a file header record.

7.2.4 Cassette Write End of File

This function writes an end of file record on the cassette tape.

Registers on entry:

```
AL = 40
```

Registers on return:

```
AX = 0
BX, CX and DX are modified
```

7.2.5 Cassette Read File Header

This function reads the data of a file header record on cassette tape into the specified input buffer.

Registers on entry:

```
AL = 26
CX = Max number of bytes to read
DX = Offset address to input buffer
Topmost element on  stack  contains segment register of
input buffer.
```

Registers on return:

```
    AL = Function result
       = 0    ok
       = 1    CRC error
       = 2    no data on tape (time out)
       = 3    no leader found (after 10 tries)
       = 4    wrong record number
       = 5    end of file

    AH and BX are modified
    CX = Number of bytes actually read
    DX = Offset (in input  buffer)  to  the  byte after the
         last byte read
    Stack is unchanged.
```

Note

The number of bytes actually read is the maximum of the specified number of bytes and the number of bytes actually in the record.

7.2.6 Cassette Read Next Data

This function reads the data of the next data record on cassette tape into the specified input buffer.

Registers on entry:

```
    AL = 28
    CX = Max number of bytes to read
    DX = Offset address to input buffer
    Topmost element on  stack  contains segment register of
    input buffer.
```

Registers on return:

```
    AL = Function result
       = 0    ok
       = 1    CRC error
       = 2    no data on tape (time out)
       = 3    no leader found (after 10 tries)
       = 4    wrong record number
       = 5    end of file

    AH and BX are modified
    CX = Number of bytes actually read
    DX = Offset (in input  buffer)  to  the  byte after the
         last byte read
    Stack is unchanged.
```

Note

If the record number of the data record is different from the preceeding record number incremented by one an error occurs (AL = 4).

Example

```
; This program writes a file header record followed by
; a data record to cassette tape. Then it waites for
; user to rewind the tape before reading the two records
; again. Finally the characters read are written on the
; console.

cseg

    mov  al,27                      ; write file header
    mov  cx,length textout1         ; number of bytes in CX
    mov  dx,offset textout1         ; offset in DX
    push ds                         ; segment on top of stack
    int  28h                        ; write it!

    mov  al,29                      ; write next data record
    mov  cx,length textout2         ; number of bytes in CX
    mov  dx,offset textout2         ; offset in DX
                                    ; segment allready on stack
    int  28h                        ; write it!


    mov  cl,9                       ; write string
    mov  dx,offset wait_text        ; on console
    int  224
    mov  cl,1                         ; read character
    int  224                          ; from keyboard


    mov  al,26                        ; read file header
    mov  cx,100                       ; max no of chars
    mov  dx,offset textin
                                      ; segment allready on stack
    int  28h                          ; read it!


    mov  al,28                        ; read next data
    mov  cx,100                       ; max no of chars
                                      ; offset allready in DX
                                      ; segment allready on stack
    int  28h                          ; read it!

    pop  ds                           ; remove segment from stack


    mov  cl,9                         ; write string
    mov  dx,offset textin             ; on console
    int  224


    mov  cx,0                         ; terminate process
    int  224

dseg

textout1      db 'PICCOLINE '
textout2      db 'the best school micro $'

wait_text     db 'rewind tape and press any key',10,13,'$'

textin        rb        100

end
```

8. Disk System

The disk configuration of a PICCOLINE consists of:

0,1 or 2 floppy disk drives
0 or 1 memory disk

Floppy disk drives are always resident inside the PICCOLINE Disk/Printer-Controlunit (DPC). Up to 4 PICCOLINES are able to share the same DPC. Therefore a PICCOLINE has to reserve the floppy disks before accessing them. This is described in section 8.4.

The disk naming conventions are as follows.

1. disk drive	A
2. disk drive	B
Memory disk	M

8.1 Disk Characteristics

The PICCOLINE diskformat uses a sector to sector skew factor of 1, and a track to track skew factor of 0, i.e. no skewing at all.

The PICCOLINE floppy disks, although the size of a 5 1/4" disk, use a format equivalent to an 8" double sided/dual density disk.

Drive performance:

Capacity	1604 Kbytes unformatted 1232 Kbytes formatted
Recording density	9646 BPI
Track density	96 TPI
Cylinders	77
Tracks	154
Encoding method	MFM
Rotational speed	360 RPM
Transfer rate	500 Kbits/sec
Latency (average)	83 msec

Access time	
Average	91 msec
Track to track	3 msec
Settling time	15 msec
Head load time	50 msec
Motor start time	1 sec

Floppy disk format:

Capacity	1232 K bytes formatted
Cylinders	77
Tracks	154
Sectors/track	8
Sector length	1024 Bytes
Precompensation (write)	
Cylinder 0-76	125 nsec

Track format:

No. of bytes		Value (hex)
80	*	4E
12	*	00
3	*	F6 (writes C2)
1	*	FC (index mark)
50	*	4E (gap 1)
8	*	sec (see below)
1150	*	4E (gap 4)
600	*	4E (filler)

Sector format:

12	*	00 (gap1/gap3)
3	*	F5 (writes A1)
1	*	FE (ID addres mark)
1	*	track no.
1	*	sector no.
1	*	03 (sector length)
1	*	F7 (2 CRC written)
22	*	4E (gap 2)
12	*	00
3	*	F5 (write A1)
1	*	FB (data addressmark)
1024	*	E5 (data)
1	*	F7 (2 CRC written)
54	*	4E (gap 3)

CCP/M drive characteristics:

77	cylinders per disk
2	track per cylinder
8	sectors per track
1024	bytes per sector
2	sectors per block (2 K bytes block size)
2	reserved tracks
616	blocks per disk
384	directory entries (FCB's) per disk
128	directory entries (SFCB's) per disk
1200	K bytes total disk capacity

8.2 Floppy Disk Controller

The floppy disk controller is based on the WD2797 controller chip. The floppy disk controller (FDC) and an external control register (FCR) (for precompensation, motor on/off and drive select) are accessed using the following I/O addresses:

Address	Direction	Function
0280H	I	Read FDC status register
	O	Write control command
0282H	I	Read FDC TRACK register
	O	Write FDC TRACK register
0284H	I	Read FDC SECTOR register
	O	Write FDC SECTOR register
0286H	I	Read FDC DATA register
	O	Write FDC DATA register
0288H	O	Write FCR register
	I	Not defined
028EH	O	Reserve floppy
	I	Bus 7 = 0 means wait upon floppy Bus 7 = 1 means ack from floppy
0290H	O	Release floppy

Further information on programming the registers on address 0280H-0286H may be found in the Western Digital documentation.

The FCR register has the following encoding:

Bit	Name	Description
0	Drive select	0 selects drive 0 1 selects drive 1
1	Motor 0	0 Motor off 1 Motor on
2	Motor 1	0 Motor off 1 Motor on
3	Write Precomp. enable	0 Disabled 1 Enabled
4	Not used	
5	Not used	must be 0
6	Not used	must be 1
7	Ready control	0 Ready from drive 1 Ready always set

Precompensation is normally applied in the following way.

Cylinder no.	Precompensation
0-77	125 nsec.

The FDC is normally initialized to transfer data in DMA-mode using DMA channel 0. In order to avoid data overrun, DMA channel 0 is assigned a high priority (see 2.2.3) when it is used by the FDC.

8.3 Floppy Disk Driver

The XIOS floppy disk driver supports the three basic CCP/M disk I/O functions:

IOSELDSK
IOREAD
IOWRITE

See ref.3 for detailed information about these functions.

Additionally the XIOS floppy disk driver supports several int-28h functions which are described in appendix A (functions 5, 8, 9, 10, 11 and 13).

8.4 Reservation of Shared Disks

The floppy disks of a PICCOLINE system are resident inside the PICCOLINE Disk/Printer-Controlunit (DPC), and can be shared by up to 4 PICCOLINES. Therefore a PICCOLINE has to reserve the DPC before accessing it and release it afterwards. Usually this is handled by the disk driver described in section 8.3, but if the disk controller is accessed directly the following int-28h functions can be used.

8.4.1 Reserve Shared Disk

This function reserves the shared disk resident inside the PICCOLINE DPC. The function returns when reservation is done.

```
    AL = 42
    AH = 1     (Reserve)
```

8.4.2 Release Shared Disk

This function releases the shared disks resident inside the PICCOLINE DPC.

```
    AL = 42
    AH = 2     (Release)
```

9. Parallel (Printer) Interfaces

The parallel interfaces on the PICCOLINE system are primarily intended for attachment of printers, but may also be used as general input/output ports.

The CCP/M operating system supports the parallel interfaces as printer devices. If no mapping to the printers (in the connection to the net) is used, printer 0 will be the port on the CPU, called the local interface, and printer 2 will be the DPC (Disk/Printer-Controlunit) interface.

The main difference between the two interfaces is that the DPC interface can be shared by up to 4 PICCOLINES, and therefore it is necessary to reserve the interface before use and release it afterwards.

9.1 Parallel Interfaces Description

An overview of the electrical signals used in the interfaces is shown below.

pin number	name
1	STROBE
2-9	8 data bit
10	ACK
11	BUSY
12	PAPER END
13	SELECTED
14	AUTO LINE FEED
15	FAULT
16	INIT PULSE
17	SELECT
18-25	0 Volt

The interface consists of 4 registers.

- Data output register, directly controlling the data pins if enabled.

- Data input port, reflecting the state of the data pins at the time of reading.

- Control output register, directly controlling 4 control output pins and enabling of data output register and interrupt.

- Status read port, reflecting the state of the 8 control/status pins at the time of reading.

The registers have the following layouts:

Data output register

		dec
Local interface	OUT 250H	592
DPC interface	OUT 28AH	650

Bit	Connector Pin no.	Description
0	2	If the output register is enabled (i.e. control register, bit 4 = 0) then a bit in the register directly controls the corresponding connector pin as follows:
1	3	
2	4	
3	5	
4	6	
5	7	
6	8	
7	9	

Bit state	TTL ouput
0	LOW
1	HIGH

Data input port

		dec
Local interface	IN 250H	592
DPC interface	IN 28AH	650

Bit	Connector Pin no.	Description
0	2	Read back of data output register, or if this is disabled the state of the connector pins.
1	3	
2	4	
3	5	
4	6	
5	7	
6	8	
7	9	

Pin TTL level	Bit state read
LOW	0
HIGH	1

Control register

		dec
Local interface	OUT 260H	608
DPC interface	OUT 28CH	652

Bit 0-3 of this register are connected through open collector inverters to corresponding connector pins (all four having pull up resistors to +5V).

Bit no.	Signal (Pin no.)	Description
0	-,STROBE (1)	See above
1	-,AUTOLF (14)	See above
2	-,INIT (16)	See above
3	-,SELECT (17)	See above
4	OUT DISABLE	Output register disable if 0: enables output register line drivers if 1: three-states the output register and allows pins 2-9 to be used for inputs.
5		NOT USED
6		NOT USED

7	INT DISABLE	Interrupt disable
		if 0: enables interrupts when BUSY input pin (11) is LOW.
		if 1: disables interrupts.

Interrupts from the parallel interfaces has been asigned:

	IR	interrupt vector address
Local interface	6	98H
DPC interface	2	88H

Status input port

		dec
Local interface	IN 260H	608
DPC interface	IN 28CH	652

Each bit in this port represents the inverse state of a pin in the connector. The 5 LSB are inputs only while the 3 MSB inputs the state of 3 of the open collector outputs.

Bit no.	Connector Pin no.	Signal description
0	11	NOT BUSY, 0 when input signal BUSY is high.
1	10	ACK, 0 when input signal is high.
2	15	FAULT, 0 when input signal is high.
3	12	NOT PAPER END, 0 when input signal is high.
4	13	NOT SELECTED, 0 when input signal is high.
5	1	STROBE, 0 when input signal is high.
6	16	INIT, 0 when input signal is high.
7	17	SELECT, 0 when input signal is high.

9.2 Sample Printer Driver Routines

In the following an example of printer driver routines to the local interface is listed

```
list_flag     Equu     12

list_init:
    ; get sysdat segment
    mov  cl,154
    int  224
    mov  sysdat,es

    ; get dispatcher address
    mov  ax,es:.38
    mov  dispatcher,ax
    mov  ax,es:.40
    mov  dispatcher+2,ax

    ; get supervisor address
    mov  ax,es:.0
    mov  supervisor,ax
    mov  ax,es:.2
    mov  supervisor+2,ax
    ; initialize interrupt vector
    xor  ax,ax
    mov  es,ax
    mov  di,98H
    mov  ax,offset parallel_interrupt
    stos ax
    mov  ax,cs
    stos ax
    ret

sysdat        dw        0
dispatcher    rw        2
supervisor    rw        2


list_out:
;   Entry:    CL = character

    ; output character to register
    mov  al,cl
    mov  dx,250h
    out  dx,al

    ; interrupt disabled,SELECT and STROBE on
    mov  al,10001001b
    mov  dx,260h
    out  dx,al

    ; interrupt disabled, SELECT on, STROBE off
    mov  al,10001000b
    out  dx,al

    ; allow printer to activate BUSY before enabling
    ; interrupt (otherwise interrupt will occur at the
    ; moment interrupt is enabled)

    mov  cx,3
list_delay:
    loop list_delay

    ; Interrupt enable, SELECT on, STROBE off
    mov  al,00001000b
    out  dx,al

    ; wait for interrupt
    mov  dx,list_flag
    call flagwait
    ret
```

```
list_status:

;    Exit:     AX =      0 if not ready
;                        0ffffh if ready

    ; test if printer is present and selected

    mov  dx,260h
    in   al,dx
    test al,16
    jnz  not_ready
    test al,8
    jz   not_ready
    mov  ax,0ffffh
    ret
not_ready:
    xor  ax,ax
    ret


parallel_interrupt:

    ; save context
    push ds
    push es
    pusha

    ; set ds to sysdat segment
    mov  ds,sysdat

    ; disable interrupt
    mov  al,10000000b
    mov  dx,260h
    out  dx,al

    ; non specific end of interrupt to external and
    ; internal interrupt controller
    mov  al,20h
    out  0,al
    mov  dx,0ff22h
    mov  ax,8000h
    out  dx,ax

    ; signal interrupt
    mov  dx,list_flag
    call flagset
    ; reestablish old context
    popa
    pop  es
    pop  ds
    jmpf cs:dword ptr dispatcher


dev_flagset             equ 133
dev_flagwait            equ 132

flagset:
    push dx
    mov  cl,dev_flagset
    call supif_1
    pop  dx
    test ax,ax
    jz   flagset_ret

    ; if error = 'ignored' then try again
    cmp  cl,2ah
    jz   flagset
flagset_ret:
    ret

flagwait:
    mov  cl,dev_flagwait

supif:
    ; get running process
    mov  bx,rlr

    ; get process's UDA
    mov  es,cs: 10HÆbxÅ

supif_1:
    xor  ch,ch
    mov  ds,sysdat
    callf     supervisor
    ret
```

9.3 The DPC Interface

Because the DPC interface can be shared by up to 4 PICCOLINEs it has to be reserved before use. After the reservation is acknowledged the interface is owned by the current PICCOLINE until released again.

This section describes two Int-28h functions avaible for reserving and releasing the DPC interface.

9.3.1 Reserve the DPC Interface

This function reserves the DPC interface:

```
    AL = 41
    AH = 1     (Reserve)
```

Note that return take place only when then DPC has been reserved.

9.3.2 Release the DPC Interface

This function releases the DPC interface:

```
    AL = 41
    AH = 2     (Release)
```

10. Serial Interface

This chapter describes the serial communication support on the PICCOLINE.

The PICCOLINE standard configuration does not include a serial communication controller. But if MF905 V24 Serial Interface (iSBX351) is installed the system supports this serial communication channel.

10.1 Standard Serial Communication Support

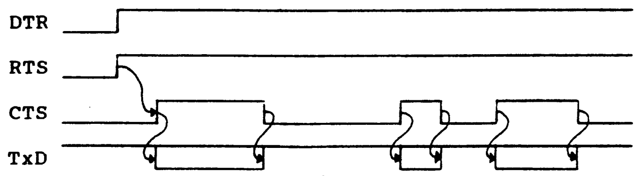
CCP/M supports the serial communication channel either as an extra console device (with console number 5) or as a list device (with device number 1).
When the channel is operated as a console device access is gained in the same way as access to the normal virtual consoles i.e. using CCP/M console input/output functions (see ref.2).

When the channel is operated as a list device, it is accessed through CCP/M's list device functions (ref.2).

The various operating parameters (such as baudrate and selection between printer mode and console mode) are set using the KONFIG program (see ref.5).

10.1.1 V24 Handshake Scheme

When operating the communication channel in the standard asynchronous mode the connected devices must adhere to the handshake scheme, based on the signals RTS (Request To Send), DTR (Data Terminal Ready), CTS (Clear To Send) and TxD (Transmit Data) as illustrated below.



The receiver will start to sample data from the RxD (Receive Data) line when the DCD (Data Carrier Detect) signal becomes active.

10.2 Serial Communication Controller

The channel is only capable of operation in asynchronous mode, but may be strapped to operate in synchronous mode instead. Detailed information of how to programme the channel can be found in ref.11.

The channel consist basically of

- a 8251A USART chip used to convert parallel output data into serial output data and serial input data into parallel input data.

- a 8253 PIT used to generate the baud rate clock of the channel.

These chips are programmed through a sequence of I/O-Read and I/O-Write commands. As shown in the following table each of the chips recognizes eight seperate I/O-addresses used to control the various programmable functions. Where two or four addresses are listed for a single function either addresses may be used.

I/O Address (hex)	Chip Select	Function
300,304, 308,30c	8251A USART	Write: Data Read: Data
302,306, 30a,30e		Write: Mode or Command Read: Status
310 or 318	8253 PIT	Write: Counter 0 (Load Count ÷ N) Read: Counter 0
312 or 31a		Write: Counter 1 (Load Count ÷ N) Read: Counter 1
314 or 31c		Write: Counter 2 (Load Count ÷ N) Read: Counter 2
316 or 31e		Write: Control Read: None

Fig.10.1 I/O Address Assignments

The interrupts from the iSBX351 are connected to the 80186 CPU INT1 and INT3 pins. The receiver and transmitter are assigned interrupt level 0DH and 0FH respectively. This corresponds to the following interrupt vector addresses:

Interrupt	address hex	address dec
receiver	0:34	0:52
transmitter	0:3C	0:60

10.2.1 Asynchronous Communication

Counter 2 of the 8253 PIT is used to generate both receiver and transmitter baud rate. Therefore it is impossible to have different receiver and transmitter baud rate in the standard configuration. If it is necessary to have different baud rates it is possible to strap the channel to use two different counters of the PIT to generate the different baud rates.

In order to get the appropriate baud rate the counter 2 register of the PIT is set to one of the values in the following table.

Value	Baud rate
1024	75
698	110
512	150
256	300
128	600
64	1200
32	2400
16	4800
8	9600

Example

Initialize the 8253 PIT to generate a 2400 baud rate clock for the serial channel (both receiver and transmitter baud rate).

```
    mov  dx,316H                    ; PIT control register
    mov  al,0B6H                    ; select counter 2. Read
    out  dx,al                      ; least sign. byte first
                                    ; then most sign. byte
    mov  dx,316H                      ; PIT counter 2
    mov  ax,32                        ; 2400 baud
    out  dx,al                        ; least significant byte
    exch ah,al                        ;
    out  dx,al                        ; most significant byte
```

Further information about the programming af the 8253 PIT and the 8251A USART can be found in ref.11.

10.3 Initializing the iSBX351

Two Int-28h functions are available for initializing the iSBX351 to operate in standard asynchronous mode.

Int-28h function 24 is used to initialize the channel according to a parameter block with the following format. The format is chosen to be the same as the format used in the Rc Partner. Hereby the serial communication channel of the PICCOLINE system is made work like channel B of the Partner system.

+ 0	1 (allways)
+ 1	Mode (0: Console; 1: Printer)
+ 2	Protocol (0: None; 1: Xon-Xoff)
+ 3	Receiver baud rate (0: 75 1: 75; 2: 110; 3: 150; 4: 300; 5: 600; 6: 1200; 7: 2400; 8: 4800; 9: 9600)
+ 4	Transmitter baud rate (as for receiver)
+ 5	No. of subsequent write register specification (see relevant INTEL doc.)
+ 6	Register no.
+ 7	Register contents
+ 8	Register no.
+ 9	Register contents

.
.
etc.

Since the channel uses only one baud rate generator only the receiver baud rate field has any effect.
The write registers mentioned corresponds to the write registers of the INTEL 8274 serial controller of the Partner system. The format of the registers can be found in section 3.2 (NVM byte 14-17) and in the relevant INTEL documentation (conserning the INTEL 8274 serial controller). Only parameters of write register 3-5 of the 8274 has any effect in the PICCOLINE system.

A pointer to the parameter block must be on the stack when the function is entered.

Registers on entry:

AL = 24

Stack on entry:

+2 = Parameter block segment
+0 = Parameter block offset

Stack on return:

Unchanged.

Example

```
    CSEG
    ORG  100H
SetSIO:
    ; Put pointer to ParamBlock on stack
    Mov  AX,Offset ParamBlock
    Push CS
    Push AX

    Mov  AX,24
    Int  28H

    ; clean-up stack
    Add  Sp,4
    Ret
```

```
ParamBlock:
    Db   1     ;
    Db   0     ; Console mode
    Db   1     ; Xon-Xoff
    Db   8     ; Receive 4800 baud
    Db   8     ; Transmit 4800 baud
    Db   3     ; 3 registers to program
    Db   4,47H; write register 4:
               ;   Clock * 16
               ;   1 stopbit
               ;   Even parity
    Db   3,61H; write register 3:
               ;   Transmit character length = 7 bit
               ;   Receive enable
    Db   5,0AAH;write register 5:
               ;   Data Terminal Ready
               ;   Receive character length = 7 bit
               ;   Transmit enabled
               ;   Request To Send

    END
```

Int-28h function 50 is used to reestablish the standard initialization (as it is done when the PICCOLINE is booted). The function has no parameters.

Example

```
ResetSIO:
    Mov  AX,50
    Int  28H
    Ret
```

Int-28h function 23 is used to read the status of the iSBX351 controller. The status returned is encoded in the following way (in order to be compatible to the rr0 and rr1 registers of the communication channel B of the Partner system).

Registers on entry:

    AL = 23

Registers on return:

    AX = Status

where the status in AX is encoded in the following way:

```
    bit 0  = Receiver ready
    bit 2  = Transmitter empty
    bit 3  = Data set ready
    bit 5  = Transmitter ready
    bit 12 = Parity error
    bit 13 = Overrun error
    bit 14 = Framing error
```

Further details can be found in the relevant Intel documentation.

10.4 Sample Asynchronous Communication Program

All examples in this chapter use the following declarations:

```
P_Flagset      Equ       133; CCP/M flagset function
P_Flagwait     Equ       132;    -   flagwait    -

ReceiveFlag    Equ        13; Flag allocated to receive
TransmitFlag   Equ        14; Flag allocated to transmit

DataPort       Equ       300; channel data port
CommandPort    Equ       302; channel command port
```

Before any data transfer can take place the hardware and software must be initialized.

The responsibility of the initialization routine is to do all hardware and software initialization needed (e.g. setting up the iSBX351 and initialize all driver variables).

Example

```
Initialize:
    ; get sysdat segment
    Mov  CL,154
    Int  224
    Mov  sysdat,ES
    ; get dispatcher address
    Mov  AX,ES:.38
    Mov  dispatcher,AX
    Mov  AX,ES:.40
    Mov  dispatcher+2,AX

    ; get supervisor address
    Mov  AX,ES:.0
    Mov  supervisor,AX
    Mov  AX,ES:.2
    Mov  supervisor+2,AX

    ; initialize interrupt vectors
    Cli
    Xor  AX,AX
    Mov  ES,AX
    Mov  Di,3CH          ; vector for transmit interrupt
    Mov  AX,Offset TransmitInterrupt
    Stos AX
    Mov  AX,CS
    Stos AX
    Mov  Di,34H          ; vector for receive interrupt
    Mov  AX,Offset ReceiveInterrupt
    Stos AX
    Mov  AX,CS
    Stos AX
    Sti

    ; Initialize iSBX controller (see 10.3)
    Call SetSIO

    Ret

sysdat         dw       0
dispatcher     rw       2
supervisor     rw       2
```

The receive routine is executed when the user program needs data from the communication line.

The program waits for data by means of a P_Flagwait operating system call. This operating system call will suspended the program until data has arrived and this has been signalled by the interrupt routine by means of a P_Flagset operating system call. To avoid loss of data it may be necessary to maintain a circular buffer which is filled with received data by the interrupt routine and emptied by the input routine when the user program needs data.

Example

```
Receive:
    ; wait for receive interrupt
    Mov  DX,ReceiveFlag
    Mov  CL,P_FlagWait
    Int  224

    ; get character from buffer
    Mov  AL,char
    Ret

char     db   0
```

The Transmit routine is executed when the user program wants to send data on the communication line.

The Transmit routine sends data and then wait for completion by means of a P_Flagwait operating system call. When the controller completes its task the transmitter interrupt service routine will signal this by means of a P_Flagset operating system call.

Example

```
Transmit:
    Mov  DX,DataPort
    Out  DX,AL

    ; wait for transmitter interrupt
    Mov  DX,TransmitFlag
    Mov  CL,P_FlagWait
    Int  224
    Ret
```

In order to handle the interrupts from the serial communication channel the system has to be enhanced with two interrupt routines:

1) Transmit interrupt routine. This routine will gain control when the controller has sent a character and is ready for the next one. The routine should clear the interrupt by issuing an 'end of interrupt' command to the INTEL 80186 interrupt controller, set the appropriate flag by means of a P_Flagset operating system call (see above) and force a process dispatch to allow a process that waits for the flag to continue execution.

2) Receive interrupt routine. This routine will gain control when the controller has received a character. The routine is responsible for reading and buffering the character, for issuing an 'end of interrupt' command, for setting the appropriate flag and for forcing a process dispatch.

Example

```
TransmitInterrupt:
    ; save context
    Push DS
    Push ES
    Pusha

    ; set ds to sysdat segment
    Mov  DS,CS:sysdat

    ; execute non specific end of interrupt
    Call Sio_EOI

    Mov  DX,TransmitFlag
    Call Flagset
    Jmp  DispatchReturn


ReceiveInterrupt:
    ; save context
    Push DS
    Push ES
    Pusha

    ; set ds to sysdat segment
    Mov  DS,CS:sysdat

    ; read character from sio
    Mov  DX,DataPort
    In   AL,DX

    ; save character in buffer
    Mov  CS:char,AL

    ; execute non specific end of interrupt
    Call sio_EOI

    ; signal program that character is received
    Mov  DX,ReceiveFlag
    Call Flagset
    ; force a dispatch
    Jmp  DispatchReturn


Sio_EOI:
    Mov  DX,0FF22H                    ; non specific end of
    Mov  AX,8000H                     ; interrupt to internal
    Out  DX,AX                        ; interrupt controller.
    Ret

DispatchReturn:
    ; reestablish old context
    Popa
    Pop  ES
    Pop  DS
    jmpf cs:dword ptr dispatcher

NodispatchReturn:
    ; reestablish old context
    Popa
    Pop  ES
    Pop  DS
    Iret

FlagSet:
    Push Dx
    Mov  CL,P_Flagset
    Call Supif_1
    Pop  DX
    Test AX,AX
    Jz   FlagSet_ret

    ; if error code='flag set ignored'
    ; then try again
    Cmp  CL,2aH
    Jz   FlagSet
FlagSet_ret:
    Ret

FlagWait:
    Mov  CL,P_Flagwait

SupIf:
    ; get running process
    ; (sysdat:68 is pointer to running process)
    Mov  BX,.68

    ; get process's UDA
    Mov  ES,CS:10HÆBXÅ
```

```
SupIf_1:
    Xor  CH,CH
    Mov  DS,sysdat
    Callf     supervisor
    Ret
```

Example

The following program uses the routines described above to indefinitely receive and transmit a character on the communication line.

```
    CSEG
    ORG 100H

    Call Initialize
NextChar:
    Call Receive
    Call Transmit
    Jmps NextChar
```

11. Local Area Network

The network software in the PICCOLINE can be considered as a collection of layers. The higher layers network software such as DR-NET (ref.4) and IMC (refs.8,9) utilize a common datalink service.

This section describes in detail the datalink service interface enabling the programmer to implement higher layer network software using the PICCOLINE datalink service. Furthermore a detailed description of the datalink layer protocol, the RCLLC protocol, is given. This decription makes it possible for the programmer to attach non RC-products to the RC Local Area Network (LAN) at the datalink level.

The protocol and service defined in this section form an extension to the proposed ISO LLC type 1 protocol and service (ref.6), viz.:

- all frames which are valid according to the RCLLC protocol are also valid ISO LLC type 1 frames

- RCLLC adds protocol functions and interface service functionality to ISO LLC type 1 in a fashion which one might choose to consider as a sublayer added on top of an LLC type 1 sublayer

The services of LLC type 1 are data transfer on connectionless data links, allowing multiple independent clients within each station, plus facilities for point-to-point loop back test traffic.

The essential service of the RCLLC layer, which constitutes an extension to the type 1 service, is called client network service. This service comprises the dynamic configuration, maintenance and supervision of multiple independent networks of clients. Connection-based data transfer with sequence control and retransmission to avoid loss of or damage to data is performed between any pair of clients belonging to the same client network.

The RCLLC protocol assumes that the services of a Medium Access Control layer are available. The services of this layer are in the PICCOLINE mainly implemented by the Intel 82586 Ethernet controller (ref.10). To enable the programmer to access the controller directly, PICCOLINE specific information about handling the controller is given.

11.1 Fundamental Concepts

The following terms are used in this document with their standard meaning as defined in the ISO model for Open Systems Interconnection (ref.7): station, layer, entity, peer, protocol, service primitive, datalink, connection.

Further concepts and terminology which are not necessarily found in the ISO model, but used in this section, are defined in the following:

An RCLLC station is a station which is attached to the local area network and hosts an RCLLC entity that communicates with peer entities in other RCLLC stations according to the RCLLC protocol. A station which supports only the LLC type 1 protocol and not the full RCLLC protocol is not considered an RCLLC station. Until the RCLLC protocol is adopted by other manufacturers, an RCLLC station will be the same as an RC product attached to the network.

The Medium Access Control (MAC) layer is the only protocol layer between the RCLLC layer and the physical network. Each RCLLC station contains precisely one MAC entity and one RCLLC entity. The station address is a unique address which identifies the station within the local area network. It follows that the station address is also a unique address of the RCLLC entity.

The data units that are transmitted among RCLLC entities using the MAC service are called RCLLC protocol elements.

A client is an RCLLC-user, i.e. an entity making use of the RCLLC service and located in the layer above the RCLLC layer.

An RCLLC Service Access Point (SAP) is the (logical) point at which a client accesses the RCLLC service. Within an RCLLC station each SAP is assigned a local SAP address in the range 1..63. The complete SAP address is the pair (station address, local SAP address) which uniquely identifies a SAP within the local area network.

A SAP can be inactive, in which case it is effectively unknown to the RCLLC layer so that all data and control information addressed to it are discarded; or it can be active. An active SAP can be used to obtain either type 1 service or client network service, but not both.

The RCLLC layer maintains a number of logical client networks. A client network has a network number (within the local area network), which must be in the range 1..63, and comprises all active SAPs within RCLLC stations on the local area network whose local SAP addresses are equal to this number, and for which client network service has been requested.

For all RC local area networks client network number 1 is assigned to an IMC network, i.e. the IMC nodes in the RCLLC stations of a local area network will all access the RCLLC service using a local SAP address of 1. Similarly, client network number 2 is used for DR NET.

Associated with each client network within a local area network is a multicast address which delimits the RCLLC stations that take part in the client network from all other stations on the network; i.e. a frame which is transmitted on the local area network with this multicast address should be received by (the MAC entity within) a station if and only if the station is an RCLLC station containing a SAP belonging to the client network.

Each SAP belonging to a client network has an associated SAP mask. The SAP mask is a 16-bit word. Two SAP masks match if at least one bit position contains a one in both masks, i.e. if a logical AND-operation yields a non-zero result. The RCLLC layer will maintain connections between all pairs of SAPs belonging to the same client network whose masks match.

11.2 The Datalink Layer Service Interface

The datalink layer (the RCLLC entity) is implemented as a CCP/M-86 Resident System Process named "NETDRV". This process creates at runtime two child processes "XMIT" and "REC". The process family will in the following description be named the driver.
The concept 'a long pointer' will in the following description mean a pointer consisting of a segment and a offset value and 'octet' will be used synonymous with 'byte'.
The interaction between the driver and the client is implemented as a message/answer concept utilizing the CCP/M-86 queue interprocess communication facility. The communication between the driver and a client will have four fundamental forms:

REQUEST

The driver accepts requests written to a queue named "link_req". This queue is created by the driver. The buffer written to the queue will contain information of the request kind and request specific parameters described below. The resources (i.e. buffers) passed to the driver in a request buffer must be regarded as locked and must not be modified until release (see CONFIRM below).

CONFIRM

The driver will always respond to an issued request with a confirm event. The purpose of the confirm event is partly to signal to the client that his outstanding resource (e.i. a data buffer) has been released and partly to inform the client about the result of the issued request.
The driver will write confirm messages to a queue created by a client. This queue is made known to the driver when the client activates a SAP. The confirm queue must be created with a buffer size = 4 bytes and a number of buffers that will ensure that the driver will not be suspended in an attempt to write to the queue.

The format of the queue buffer is:

byte number

0	user buffer offset
2	user buffer segment

User buffer refers to the buffer pointer in the request buffer (see below) passed by the client to the driver in the confirmed request. The first three bytes of the buffer will have the following format:

byte number

0	depend on the confirmed request
1	depend on the confirmed request
2	result of the confirmed request

The result can have one of the following values:

result value	explanation
0	no problems
1	link down
2	protocol error - already one outstanding data request on the requested connection
4	SAP class error - the requested service requires that the SAP has been activated as a RCLLC SAP
5	SAP class error - the requested service requires that the SAP has been activated as a type 1 SAP
6	SAP occupied by another client
7	can't activate a new SAP - no resources
8	illegal SAP number
9	data buffer too big (> 1076 bytes)
10	protocol error SAP removed - reason why unsyncronized disconnect acknowledge
255	request not implemented

INDICATION

The indication event is signaled by the driver to the client to indicate an internal event which is significant to the client i.e. a data buffer has been received or a connection has been established or removed.
The driver will write indication events to a queue created by a client. This queue is made known to the driver when the client activates an SAP.
The indication queue must be created with a buffer size = 4 bytes and a number of buffers that will ensure that the driver will not be suspended in an attempt to write to the queue.

The format of the queue buffer is:

byte number

0	indication structure offset
2	indication structure segment

The content of the indication structure will be described below in the description of the individual indications.

NOTE

> The indication structure must not be modified by the client. Modifications of the indication structure can make the system behave unpredictably.

INDICATION ACKNOWLEDGE

Whenever the driver writes an indication event to the indication queue, it will pass resources (the indication structure and in most cases a data buffer) to the client. Immediately after processing the indication event (i.e. copying a possible data buffer into a local buffer), the client must return these resources to the driver with an INDICATION ACKNOWLEDGE. In assembly language it is done with a few lines of code:

```
; assumption ds:bx long pointer to the indication structure

                             ;push parameters onto the stack
          push     ds        ;segment part of the long pointer
          push     bx        ;offset part of the long pointer
          int      29h       ;the software interrupt executes
                             ;the indication acknowledge
          add      sp,4      ;clean up stack
```

11.2.1 RCLLC Services

The RCLLC services are obtained by a client through an active SAP. A SAP can be used either for type 1 service or for client network service, but not for both. The loop-back test facility is available regardless of the choice of type 1 or client network service.

SAP Activation and Deactivation

There are four primitives to request activation and deactivation of a SAP and to confirm the processing of these requests. They are described in the following subsections.

11.2.1.1 ACTIVATE.request

The primitive which requests the activation of a SAP is passed from a client to the driver by writing a request buffer to the 'link_req' queue. The driver can support two simultaneous SAPs of any type.

<u>Format of the request buffer</u>:

byte number	
0	request kind = 0 (activate.request)
1	specifies the local SAP address of the SAP to be activated (must be in the range 1 - 63).
2	specifies whether type 1 service (value = 1) or client network service (value = 0) is requested
3-4	queue ID for the indication queue
5-6	queue ID for the confirm queue
7-8	the length of the client information (max 46 bytes)
9-10	client information offset
11-12	client information segment
13-14	unused

The indication queue and the confirm queue must be created and opened by the client before any attempts to request activation of a SAP. The queue IDs must be fetched from the Queue Parameter Block (QPB see ref.2) after the queues have been opened.

Client_information is a data unit which is transmitted and passed to the remote client in the CONNECT_indication primitive whenever a connection is established between the activated SAP and a remote SAP. The format of the client information buffer is:

byte number

0-5	reserved by the driver
6-7	the SAP mask used to prevent establishment of undesired connections, cf. section 11.1
8-45	client defined information

The length of the client information includes the reserved bytes and the two SAP mask bytes.

11.2.1.2 ACTIVATE.confirm

The primitive which is issued in response to an ACTIVATE.request primitive is passed from the driver to the requesting client. This is done by writing a long pointer (pointing to the client information buffer) to the clients confirm queue.

<u>Format of the returned client information buffer</u>:

byte number

0	unused
1	confirm kind = 0 (activate.confirm)
2	confirm result indicates whether the SAP was successfully activated

When a SAP has been activated for type 1 service UDATA.request primitives may be issued requesting the transmission of data.

When a SAP has been activated for client network service, the RCLLC layer will automatically begin to establish the appropriate connections. As each connection is established, the client will be informed by means of a CONNECT.indication primitive and may subsequently request transmission of data by issuing DATA.request primitives.

In either case, once a SAP has been activated, the client may issue the TEST.request primitive to request a loop-back test.

11.2.1.3 DEACTIVATE.request

The primitive which requests the deactivation of a SAP is passed from a client to the driver by writing a request buffer to the 'link_req' queue.

<u>Format of the request buffer</u>:

byte number

0	request kind = 1 (deactivate.request)
1	specifies the local SAP address of the SAP to be deactivated
2-3	deactivate buffer offset
4-5	deactivate buffer segment
6-14	unused

The deactivate buffer must be at least 3 bytes long and it is returned to the client by the deactivate.confirm.

11.2.1.4 DEACTIVATE.confirm

The primitive which is issued in response to a DEACTIVATE.request primitive is passed from the driver to the requesting client. This is done by writing a long pointer (pointing to the deactivate buffer) to the clients confirm queue. The deactivate confirm event is a signal to the client, that <u>all</u> outstanding resources i.e. queues or databuffers can be regarded as released.

<u>Format of the returned deactivate buffer:</u>

byte number

0	unused
1	confirm kind = 1 (deactivate.confirm)
2	confirm result (always ok)

Loop-back Test Service

The loop-back test facility allows a client to request a test of the transmission path between the local RCLLC entity and one or more remote RCLLC entities without requiring the participation of any remote client(s). This is done by transmitting a TEST protocol element (command) to the specified RCLLC entity/entities to which it/each of them must respond by transmitting a TEST protocol element (response) addressed to the requesting client (SAP).

Note

No indication is given if the responding protocol element fails to arrive from any or all of the RCLLC entities addressed in the TEST.request primitive.

11.2.1.5 TEST.request

The primitive, which requests that one or more transmission paths be tested, is passed from a client to the RCLLC entity by writing a request buffer to the 'link_req' queue.

Format of the request buffer:

byte number	
0	request kind = 4 (test.request)
1	DSAP is the remote SAP address.
2	SSAP is the local SAP address.
3-8	Ethernet address. This is the MAC address of the remote RCLLC entity; a multicast or broadcast address may be used in place of a specific station address to request testing of multiple transmission paths.
9-10	length of the test buffer
11-12	test buffer offset
13-14	test buffer segment

The first three bytes in the test buffer are reserved by the driver. The length of the test buffer includes the bytes reserved by the driver.

11.2.1.6 TEST.confirm

The primitive which is issued in response to a TEST.request primitive is passed from the driver to the client. This is done by writing a long pointer (pointing to the test buffer) to the clients confirm queue.

Format of the returned test buffer:

byte number	
0	DSAP is the remote SAP address.
1	confirm kind = 4 (TEST.confirm)
2	result indicates how the transmission of test data unit went, e.g. 'no problems' or 'too many collisions'

11.2.1.7 TEST.indication

The primitive which indicates that a TEST response protocol element addressed to the local SAP has been received is passed from the driver to the client. This is done by writing a long pointer (pointing to the indication datastructure) to the clients indication queue.

Format of the indication datastructure:

byte number	
0	indication kind = 1 (TEST.indication)
1	reserved
2-3	the length of the received test buffer
4-7	reserved
8-9	received test buffer offset
10-11	received test buffer segment
12-13	reserved
14-19	source Ethernet address

The information part of the test buffer begins at the fourth byte in the test buffer. The first three bytes are included in the length of the test buffer.
Note that a TEST.request primitive issued by a client in an RCLLC station does not cause this primitive to be generated in remote RCLLC station(s), as the protocol element (TEST command) which is transmitted in this case is not addressed to a SAP, but to one or more remote RCLLC entities.

11.2.2 Type 1 Service

Type 1 service comprises unacknowledged connectionless data transfer between SAPs.

11.2.2.1 UDATA.request

The primitive which requests transmission of a data buffer is passed from a client to the driver by writing a request buffer to the 'link_req' queue.

<u>Format of the request buffer</u>:

byte number

0	request kind = 2 (UDATA.request)
1	DSAP is the remote SAP address.
2	SSAP is the local SAP address.
3-8	Ethernet address. This is the MAC address of the remote RCLLC entity; a multicast or broadcast address may be used in place of a specific station address.
9-10	length of the data buffer
11-12	data buffer offset
13-14	data buffer segment

The first three bytes in the data buffer are reserved by the driver. The length of the data buffer includes the bytes reserved by the driver.

11.2.2.2 UDATA.confirm

The primitive which is issued in response to a UDATA.request primitive is passed from the driver to the client. This is done by writing a long pointer (pointing to the data buffer) to the clients confirm queue.

<u>Format of the returned data buffer</u>:

byte number	
0	DSAP is the remote SAP address
1	confirm kind = 2 (UDATA.confirm)
2	result indicates how the transmission of the data buffer went, e.g. 'no problems'

11.2.2.3 UDATA.indication

The primitive which is used to deliver a received RCLLC service data unit is passed from the driver to the client. This is done by writing a long pointer (pointing to the indication datastructure) to the clients indication queue.

<u>Format of the indication datastructure</u>:

byte number	
0	indication kind = 2 (UDATA.indication)
1	reserved
2-3	the length of the received data buffer
4-7	reserved
8-9	received data buffer offset
10-11	received data buffer segment
12-13	reserved
14-19	source Ethernet address

The information part of the data buffer begins at the fourth byte in the data buffer. The first three bytes are included in the length of the data buffer.

11.2.3 Client Network Service

The driver automatically establishes and maintains a connection between each pair of SAPs belonging to the same client network, except when the connection is excluded because the SAP masks do not match.

When an SAP is activated, connections will be established to those remote SAPs which were already active. The (local) client will receive a CONNECT.indication primitive for each connection when it has been established. Similarly the remote clients will each receive a CONNECT.indication primitive.

When a connection has been established, both clients may request the transmission of RCLLC service data units by issuing DATA.request primitives. A received data unit is passed to the client at the destination SAP by means of a DATA.indication primitive.

The order in which RCLLC service data units are passed to the driver for transmission on a connection is preserved to the point of delivery. RCLLC service data units are delivered free of transmission errors.

When an SAP is deactivated, either because of a request or because the station in which it exists ceases to operate or is reinitialized, the driver will detect the event and remove the connections in which the SAP took part. Each of the clients at the remote end of such a connection will be notified by means of a DISCONNECT.indication primitive.

There is no guarantee that all service data units passed to the driver for transmission will have been delivered before a connection is removed.

When the driver has removed (one end-point of) a connection and passed the indication to the client it will not establish a new connection to the same remote SAP until the client has acknowledged the removal of the connection by issuing a DISCONNECT.acknowledge primitive. This procedure is significant when a connection is removed because of a temporary malfunction or the reinitialization of a station. It allows the client to gracefully terminate any activity associated with the connection before it is reestablished.

Details about the six primitives used in conjunction with client network service are given in the following subsections.

11.2.3.1 CONNECT.indication

The primitive, which indicates that a connection has been established, is passed from the driver to the client. This is done by writing a long pointer (pointing to the indication datastructure) to the clients indication queue.

Format of the indication datastructure:

byte number	
0	indication kind = 3 (CONNECT.indication)
1	connection index (the station address of the remote SAP)
2-3	the length of the received client information
4-7	reserved
8-9	received client information offset
10-11	received client information segment
12-13	reserved
14-19	source Ethernet address

The information part of the client information begins in the ninth byte in the client information buffer. The first eigth bytes are included in the length of the client information.
After receiving the primitive the client may issue DATA.request primitives on the connection, and should expect DATA.indication primitives to arrive.

11.2.3.2 DISCONNECT.indication

The primitive, which indicates that a connection has been removed, is passed from the driver to the client. This is done by writing a long pointer (pointing to the indication datastructure) to the clients indication queue.

Format of the indication datastructure:

byte number

0	indication kind = 0 (DISCONNECT.indication)
1	connection index of the disconnected connection
2-19	reserved

The client should acknowledge receipt of the primitive by issuing a DISCONNECT.acknowledge primitive. A new connection to the same remote SAP will not be established until this has been done.

11.2.3.3 DISCONNECT.acknowledge

The primitive which acknowledges the removal of a connection is passed from a client to the driver by writing a request buffer to the 'link_req' queue.

Format of the request buffer:

byte number

0	request kind = 6 (DISCONNECT.acknowledge)
1	connection index. This is the logical address of the remote MAC entity.
2	DSAP is the local SAP address, i.e. the client network number.
3	SSAP is the local SAP address, i.e. the client network number.
4-5	unused
6-7	disconnect acknowledge buffer offset
8-9	disconnect acknowledge buffer segment
10-14	unused

The disconnect acknowledge buffer must be at least three bytes long and it is returned to the client by the DISCONNECT ACKNOWLEDGE confirm. After receiving the primitive the driver may establish a new connection to the same remote SAP.

11.2.3.4 DISCONNECT_ACKNOWLEDGE.confirm

The primitive, which is issued in response to a DISCONNECT.ACKNOWLEDGE primitive, is passed from the driver to the client. This is done by writing a long pointer (pointing to the disconnect acknowledge buffer) to the clients confirm queue.

Format of the returned disconnect acknowledge buffer:

byte number

0	unused
1	confirm kind = 6 (DISCONNECT.acknowledge)
2	result

11.2.3.5 DATA.request

The primitive, which requests that a data buffer be transmitted on a connection, is passed from a client to the driver.

NOTE

The client must not request transmission on the same connection until confirmation (DATA.confirm) has been received. There are no restrictions on other connections.

Format of the request buffer:

byte number

0	request kind = 5 (DATA.request)
1	connection index. This is the logical address of the remote MAC entity
2	DSAP is the local SAP address, i.e. the client network number
3	SSAP is the local SAP address, i.e. the client network number

4-5	length of the data buffer
6-7	data buffer offset
8-9	data buffer segment
10-14	unused

The first six bytes of the data buffer are reserved by the driver. The length of the data buffer includes the bytes reserved by the driver.

11.2.3.6 DATA.confirm

The primitive, which indicates that a data buffer previously passed in a DATA.request primitive has been transmitted on a connection, is passed from the driver to the requesting client. This is done by writing a long pointer (pointing to the data buffer) to the clients confirm queue.

<u>Format of the returned data buffer</u>:

byte number

0	DSAP is the remote SAP address, i.e. the client network number.
1	confirm kind = 5 (DATA.confirm)
2	result indicates how the transmission went, e.g. 'no problems' , 'too many collisions' or 'link down'

The primitive may confirm that the data unit has been transmitted and acknowledged, but not that it has been delivered to and received by the remote client.

11.2.3.7 DATA.indication

The primitive, which is used to deliver a data buffer received on a connection, is passed from the driver to the client. This is done by writing a long pointer (pointing to the indication datastructure) to the clients indication queue.

Format of the indication datastructure:

byte number

0	indication kind = 4 (DATA.indication)
1	connection index (identify the connection on which the data has been received)
2-3	the length of the received data buffer
4-7	reserved
8-9	received data buffer offset
10-11	received data buffer segment
12-13	reserved
14-19	source Ethernet address

The information part of the data buffer begins at the seventh byte in the data buffer. The first six bytes are included in the length of the data buffer.

11.3 MAC Services

The function performed by the MAC layer is to accept from an RCLLC entity a MAC service data unit, to transmit it to one, several (multicast), or all stations in the network, and in the receiving station(s) to deliver the unit to the destination RCLLC entity(ies).

In the PICCOLINE (CSMA/CD) type network the MAC service includes retransmission following a detected collision.

There is no guarantee that a MAC service data unit which is transmitted from one station on the network is received at the destination station(s).

Each MAC entity is sensitive to its station address and possibly one or more multicast addresses, i.e. addresses of groups of stations to which the station belongs. Only MAC service data units transmitted with one of these addresses or the broadcast address will be received by the MAC entity.

Padding of frames containing MAC service data units in order to reach the minimum size (46 bytes) is performed by the MAC layer. The padding is removed again before delivery.

The maximum size (1076 bytes) for MAC service data units is also enforced by the MAC layer, i.e. data units exceeding the maximum size will not be transmitted, and the receiver part of a MAC entity will discard all incoming frames that would yield a data unit longer than the maximum size.

The RCLLC layer uses the MAC service by transmitting each RCLLC protocol element as a MAC service data unit.

11.3.1 Controller Specific Information.

This subsection describes the PICCOLINE specific programming of the INTEL 82586 Ethernet controller. For general information about programming the controller we refer to ref.10.

Interrupt vector.

The offset part of the pointer to the net controller interrupt rutine must be placed in 0:94H. The segment part of the pointer to the net controller interrupt routine must be placed in 0:96H.

Setting up interrupt vector:

```
; assumption cs:ax  long pointer to net controller
; interrupt routine
; es segment register = 0 (interrupt table starts in 0:0)


        cli             ;disable all interrupts
        mov  di,94H     ;94H = offset part of net interrupt
                        ;routine
        stosw           ;
        mov  ax,cs      ;get segment pointer
        stosw           ;96H = segment part of net
                        ;interrupt
        sti             ;enable interrupts
```

After setting of the interrupt vector the interrupt source must be enabled. It is default disable after system initialization.

Enabling interrupts from the net controller:

```
        mov  dx,2          ;8259 interrupt controller I/O
                           ;address
        and  al,10111111B  ;enable net interrupt
        out  dx,al         ;execute the open
```

The communication with the net controller is performed by information exchange in common memory (the SCB and related control structures). When the user will force the controller to look in the common memory, he executes a channel attention. When the controller will force the user to look in the common memory, it executes an interrupt.

A channel attention is performed:

```
        mov  dx,100H    ;net controller channel
                        ;attention I/O address
        in   al,dx      ;note overwrites the contents of al
                        ;reg with non significant
                        ;information
```

Due to an Intel based inconsistency between the CRT controller's and the net controller's interpretation of the SYSBUS bit, the initialization of the net controller differs a little bit from the description given in ref.11. Ref.10 prescribes that the System Configuration Pointer (SCP) begins at location 0FFFF:6 (PICCOLINE Prom address room). In PICCOLINE the SCP is placed in the RAM address room. The SCP segment is 3000H and the SCP offset is 0FFF6H. The SYSBUS byte in the SCP must be 0 to indicate 16 bits bus word mode.

11.4 RCLLC Datalink Layer Protocol.

The description of RCLLC procedures falls in two parts:

1) the type 1 procedures which are in conformance with (ref.6)

2) the procedures for client network service which con stitute a functional extension to the type 1 procedures.

In general, an RCLLC protocol element may be a command protocol element or a response protocol element. As a protocol element is transmitted using the services of the MAC layer it may be addressed to one or several stations, using an individual, multicast, or broadcast station address. Within each addressed station an RCLLC protocol element is addressed either to the RCLLC entity as such or to a specific SAP.

11.4.1 Type 1 Procedures

This section contains a general description of the type 1 procedures. Details not covered in the general description are given in conjunction with the individual protocol elements in section 11.4.3.

11.4.1.1 Unacknowledged Data Transfer

This subsection applies to data transfer between active SAPs for which type 1 service has been requested.

Unacknowledged connectionless data transfer as requested by the UDATA.request primitive is accomplished by transmission of a UI protocol element containing the service data unit passed as a parameter of the primitive. This may occur at any time while the source SAP is active.

When a UI protocol element is correctly received, the service data unit which it contains is passed to the client by means of a UDATA.indication primitive. There is no associated acknowledgement or sequence checking. Notice that a UI protocol element which is found to be in error by the receiving MAC or RCLLC entity is simply discarded. Buffer shortage in the receiving RCLLC entity may also cause a UI protocol element to be discarded.

11.4.1.2 Loop-back Test Procedure

An RCLLC entity will initiate the loop-back test procedure upon receipt of a TEST.request primitive from a client. It does so by transmitting a TEST command protocol element with the poll bit set to 1 and addressed as specified in the request. The information field of the TEST command will contain the specified test data unit. Notice that multi- or broadcasting may be used to test several transmission paths using one command protocol element.

For each TEST response protocol element which is subsequently received correctly, with or without an information field, the client is informed by means of a TEST.indication primitive.

An RCLLC entity will not transmit a TEST command protocol element, except when directed by a TEST.request primitive.

When an RCLLC entity correctly receives a TEST command protocol element addressed to itself or to an active SAP, with the poll bit set to 1, it will respond by transmitting a TEST response protocol element addressed to the source RCLLC entity or SAP. The received information is copied to the response protocol element. If the information field could not be held in the receive buffer(s) of the RCLLC entity due to overlength, the response protocol element will contain an empty information field. The receiving of the TEST command protocol element will not effect the receiving RCLLC entity's clients.

A TEST command protocol element received with poll bit set to 0 is discarded.

11.4.1.3 Station Identification Exchange

Type 1 station identification exchange is not supported as part of the RCLLC service interface, and an RCLLC entity will not, therefore, transmit XID command protocol elements. It will, however, answer politely when an XID protocol element addressed to itself or to an active SAP is correctly received. Observe that the source station in this case will not be an RCLLC station.

11.4.2 Procedures for Client Network Service

This section contains a general description of the procedures for client network service. Details not covered in the general description are given in conjunction with the individual protocol elements in section 11.4.3.

Client networks are supervised by the RCLLC layer. The protocol element ACTIVE_SAP plays a central role in this respect. Whenever a SAP belonging to a client network is active the RCLLC entity serving the SAP will regularly transmit this protocol element to its peer entities using the multicast address for the client network. An RCLLC receiving the ACTIVE_SAP protocol element will discard it unless the included SAP mask matches the mask of the local SAP belonging to the same client network.

This procedure serves to make an SAP known throughout the client network so all desired connections to the SAP may be established. Notice that the SAP remains unknown to all stations where its mask does not match the local SAP mask.

Moreover, the procedure allows RCLLC entities to supervise that all existing connections are alive. When the ACTIVE_SAP protocol element fails to arrive from an SAP to which a connection exists, for a sufficiently long period of time, this will be taken to indicate that the SAP is no longer active, and the RCLLC entity will therefore remove its end of the connection.

All protocol elements other than ACTIVE_SAP are transmitted using individual station address.

The rigorous description of the procedures for establishment and supervision of connections which is given in the following is based on a state, two timers and a retransmission counter maintained by an RCLLC entity for each connection in which it takes part, i.e. for each remote SAP it knows. The following connection states exist: UNKNOWN, RESETTING, DATA, DISCONNECTING. The timers are:

- the acknowledgement timer which runs when an acknowledgement, i.e. a RACK or ACK protocol element, is expected

- the SAP alive timer which runs whenever the remote SAP is known and is restarted each time an ACTIVE_SAP protocol element is received.

In addition to the state, timers, and retransmission counter an RCLLC entity maintains for each connection two sequence counters for data units, N(S): the number of the data unit to transmit, and N(R): the number of the next data unit to be received.

The following events may cause the state of a connection to change:

PE_new_SAP	An ACTIVE_SAP protocol element is received from the remote SAP indicating it has become active, possibly by reinitialization, see section 11.4.3.4
PE_rack	A RACK or RESET protocol element is received from the remote SAP when RACK is expected.
PE_reset	A RESET protocol element is received from the remote SAP, except when RACK is expected.
give_up	The RCLLC entity gives up the connection when the retransmission counter is exhausted, or when the SAP alive timer runs out.

SP_dack An expected DISCONNECT.acknowledge service primitive is received from the client.

An overview of the state changes caused by events and the associated actions, i.e. protocol elements and service primitives that are generated, is given in figure 1. Note that the figure and the description which follows apply to a single connection, in fact to each end-point separately.

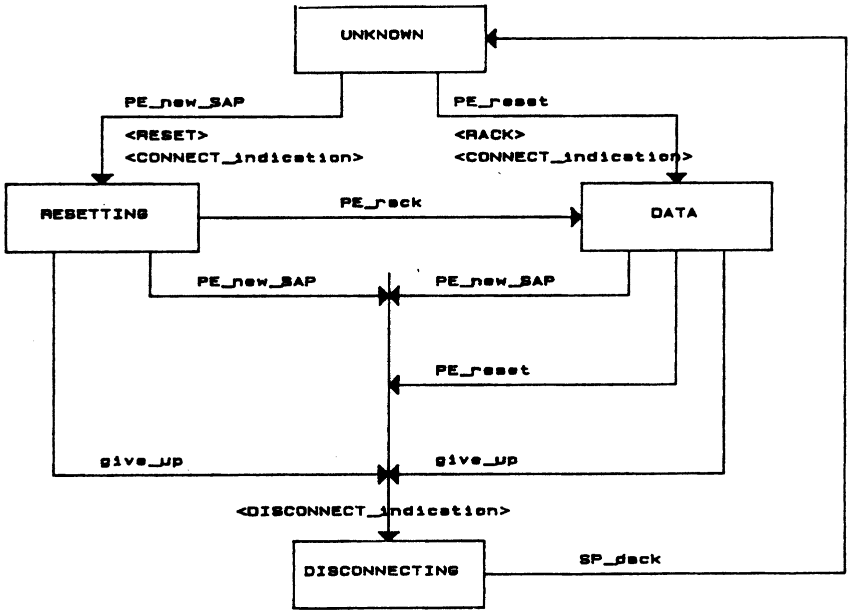


Figure 11.1: State graph for a connection.

A general procedure applies to the transmission of protocol elements for which an acknowledgement is required in the form of a protocol element transmitted in the opposite direction, viz. RESET and DATA which are acknowledged by RACK and ACK, respectively. Initiating the transmission of one of these elements means: initializing the retransmission counter, starting the acknowledgement timer, and actually transmitting the protocol element. When the acknowledging protocol element arrives, the transmission is considered successfully completed, and the timer is stopped. If, on the other hand, the acknowledgement timer expires, the retransmission counter is decremented, and if it was exhausted, i.e. became zero, the connection is given up (give_up event). Otherwise, the timer is restarted and the protocol element retransmitted.

There is never more than one outstanding protocol element requiring acknowledgement, i.e. transmission of a RESET or DATA protocol element is not initiated until transmission of the previous element is completed. For this reason a DATA.request primitive containing an RCLLC service data unit for transmission on a connection may be accepted while an unacknowledged protocol element is outstanding, but it will then be queued (by the RCLLC entity) for transmission rather than processed immediately.

The remaining part of this section contains a discussion of the meaning of each state of a connection (end-point) and the procedures followed by an RCLLC entity in each state.

UNKNOWN

The RCLLC entity has no knowledge of the remote SAP, but is ready to establish a connection. No service primitives are accepted and all protocol elements except ACTIVE_SAP and RESET are discarded.

A received ACTIVE_SAP protocol element (with matching SAP mask) constitutes a PE_new_SAP event. It causes the RCLLC entity to establish a connection to the remote SAP by starting the SAP alive timer, initiating the transmission of a RESET protocol element, resetting the sequence counters, passing a CONNECT.indication primitive to the client, and changing the connection state to RESETTING.

A received RESET protocol element constitutes a PE_reset event indicating that the local SAP has become known to the remote RCLLC entity and caused it to establish a connection. The local RCLLC entity will establish its end of the connection by starting the SAP alive timer, transmitting a RACK protocol element to acknowledge RESET, resetting the sequence counters, passing a CONNECT.indication primitive to the client, and changing the connection state to DATA.

RESETTING

The RCLLC entity has established the connection by initiating the transmission of a RESET protocol element. The state is used to wait for the acknowledging RACK protocol element after which data may be transmitted in both directions.

DATA.request primitives are accepted (queued). DISCONNECT.-acknowledge primitives are discarded.

A received RESET or RACK protocol element constitutes a PE_rack event and causes the RCLLC entity to change the connection state to DATA. RESET, which may occur if RESET protocol elements are transmitted in both directions simultaneously, is answered with RACK.

Received DATA or ACK protocol elements are discarded.

If a PE_new_SAP event occurs (see section 11.4.3.4), or if the connection is given up, either because the SAP alive timer expires or because the RESET protocol element is retransmitted to exhaustion, the RCLLC entity will pass a DISCONNECT.indication primitive to the client and change the connection state to DISCONNECTING.

DATA

The connection has been completely established through the exchange of RESET and RACK protocol elements. In this state RCLLC service data units are transferred between the two SAPs through the exchange of DATA and ACK protocol elements between the RCLLC entities.

Each DATA.request primitive received from the client causes initiation of the transmission of a DATA protocol element containing the service data unit passed as a parameter of the primitive. The sequence number of the protocol element is set equal to the value of N(S), and subsequently N(S) is incremented modulo 2. Initiation of the transmission of the protocol element takes place: either when an ACK or RACK protocol element is received marking the successful completion of a previous transmission provided a non-empty queue of service data units are awaiting transmission; or immediately upon receipt of the DATA.request primitive if there is no outstanding protocol element awaiting acknowledgement.

When a DATA protocol element is received, its sequence number is compared to the value of N(R). If they are equal the received service data unit is passed to the client by means of a DATA.indication primitive, and N(R) is incremented modulo 2. Otherwise, the service data unit is discarded. In both cases an ACK protocol element with sequence number equal to that of the DATA protocol element is transmitted to the remote SAP in order to acknowledge receipt.

If a RACK protocol element or a DISCONNECT.acknowledge service primitive is received, it is discarded.

If a PE_new_SAP event occurs (see section 11.4.3.4), if a RESET protocol element is received, or if the connection is given up, either because the SAP alive timer expires or because a DATA protocol element is retransmitted to exhaustion, the RCLLC entity will pass a DISCONNECT.indication primitive to the client and change the connection state to DISCONNECTING.

DISCONNECTING

The connection has been disconnected as seen from the point of view of the RCLLC layer. This state allows the client to decide when it will accept the connection to be reestablished.

All received protocol elements and service primitives are discarded except the DISCONNECT.acknowledge primitive. When this primitive is received the connection state is changed to UNKNOWN.

11.4.3 RCLLC Protocol Elements

All RCLLC protocol elements conform to the syntax for LLC type 1 protocol elements ("protocol data units"). This is achieved by defining the formats of all the protocol elements used in the procedures oriented toward client network service to be instances of type 1 UI (Unnumbered Information) commands.

The following conventions apply to the figures in this section: the octets of a protocol element are shown in the order they are transmitted downward on the page, and the bits within an byte similarly from right to left. The least significant bit position within an byte, the contents of which are transmitted first, is numbered 0, and so forth.

The general format for type 1 protocol elements consists of a three-octet link control header followed by an information field:

bit no.	7	6	5	4	3	2	1	0
byte no. 0	DSAP						0	0
1	SSAP						0	C/R
2	Control							
3	Type 1 Information							

The DSAP field contains the local SAP address of the destination SAP and the SSAP field the local SAP address of the source SAP.

If the DSAP field contains 0 (all bits 0) the protocol element is interpreted as addressed to the destination RCLLC (or other LLC type 1) entity rather than to a client.

If the DSAP field does not contain all 0 bits, its contents taken as a binary number in the range 1..63 are interpreted as the address of an individual SAP.

A C/R bit with value 0 indicates a command protocol element, and one with value 1 a response protocol element. The UI protocol element, and thus all protocol elements for client network service, can only be transmitted as commands, i.e. with the C/R bit set to 0.

Bit 0 of byte 0 and bit 1 of byte 1 must always be 0.

If the SSAP field contains 0 (all bits 0), the protocol element is interpreted as originating from the source RCLLC (or other LLC type 1) entity rather than from a client. Otherwise, the contents of the SSAP field are interpreted as a binary number in the range 1..63.

Bit 4 of byte 2 (the Control field) is the Poll/Final bit. When this bit is set to 1 (Poll) in a command, a response is requested. The response should contain the same coding of the Control field; i.e., bit 4 (Final) should also be set in the response. The Poll bit must not be set in a UI protocol element; this bit is 0 in all protocol elements for client network service.

The SSAP field of a response protocol element always contains the same value as the DSAP field of the command protocol element to which it corresponds, and vice versa.

The remaining bits of the Control field specify the type of protocol element in question, viz.:

00000011	UI, Unnumbered Information
101X1111	XID, eXchange IDentification
111X0011	TEST

The use of the Type 1 Information field depends on the type of protocol element, and is described for each element type in the section Type 1 Protocol Elements page 141.

The protocol elements for client network service are UI commands addressed to an individual SAP with three extra bytes of RCLLC header in addition to the LLC type 1 header. Source
and destination SAPs have the same local address which is equal to the client network number, Netno. The format is as shown below:

bit no.	7	6	5	4	3	2	1	0		
byte no. 0	Netno						0	0		
1	Netno						0	0	type 1	
2	0	0	0	0	0	0	1	1	header	RCLLC
3	Function									header
4	Param 0									
5	Param 1									
6	Information									

The value in the Function field specifies the type of protocol element, viz.:

00000000 (binary 0)	ACTIVE_SAP
00000001 (binary 1)	RESET
00000010 (binary 2)	RACK
00000011 (binary 3)	DATA
00000100 (binary 4)	ACK

The use of the Param 0 and 1 fields and of the Information field depends on the type of protocol element, and is described for each element type in the section Protocol Elements for Client Network Service page 141.

Type 1 Protocol Elements

This section specifies the encoding of the Type 1 Information field of protocol elements used in conjunction with type 1 procedures.

11.4.3.1 UI (Unnumbered Information)

The UI protocol element may only be transmitted as a command, i.e. the C/R bit must be 1.
When the protocol element is used for type 1 service the Type 1 Information field is used to hold an RCLLC service data unit.

11.4.3.2 XID (eXchange IDentification)

The Type 1 Information field in a received XID command is ignored. In an XID response protocol element transmitted by an RCLLC entity three octets, numbers 6 through 8, are encoded as follows:

bit no.	7	6	5	4	3	2	1	0
byte no. 6	1	0	0	0	0	0	0	1
7	0	0	0	0	0	0	0	1
8	0	0	0	0	0	0	0	0

11.4.3.3 TEST

The Type 1 Information field is used to hold a test data unit. The associated procedure is described in subsection 11.4.1.2.

Protocol Elements for Client Network Service

In order to facilitate speedy access to status information associated with connections, each RCLLC entity will assign to each connection an index in the range 0..255. When a connection is established the assigned indices are exchanged between the two RCLLC entities. Subsequent DATA and ACK protocol elements each contains the index assigned to the connection by the receiver of the element.

11.4.3.4 ACTIVE_SAP

An RCLLC entity transmits this protocol element periodically for each active SAP it serves which belongs to a client network. It is transmitted using the multicast address for the client network in question so that all relevant RCLLC entities will receive it. The frequency with which the protocol element is transmitted depends on the implementation.

The first word of the information field contains the SAP mask of the active SAP. Unless the mask matches that of the local SAP at the receiving RCLLC entity the protocol element is discarded.

The Param 1 field contains a sequence number in the range 0..254. The first 255 ACTIVE_SAP protocol elements transmitted after activation of a SAP will have sequence numbers 0, 1, 2,.. 254. In all subsequent ACTIVE_SAP protocol elements the sequence number will also be 254. This procedure allows the receiving RCLLC entity to detect when an SAP is deactivated and swiftly reactivated, possibly because of station reinitialization.

When an ACTIVE_SAP protocol element is received from a previously unknown SAP a PE_new_SAP event is generated (cf. section 11.4.2). The same is the case if the sequence number is less than the sequence number found in the last received ACTIVE_SAP or RESET protocol element from the same SAP. However, when the sequence number are equal or ascending, the protocol element is only taken to indicate that the SAP is still active. In the latter case the SAP alive timer is restarted.

The Information field from the third byte contains the client_info passed from the client when the SAP was activated.

11.4.3.5 RESET

This protocol element is transmitted in conjunction with establishment of a connection.

The Param 0 field contains the index assigned to the connection by the sending RCLLC entity.

The Param 1 field contains the sequence number to be included in the next ACTIVE_SAP protocol element to be transmitted from the sender.

The Information field contains the client_info passed from the client when the SAP was activated.

11.4.3.6 RACK

This protocol element is transmitted to acknowledge receipt of a RESET protocol element in conjunction with establishment of a connection.

The Param 0 field contains the index assigned to the connection by the sending RCLLC entity.

The Param 1 field contains the index assigned to the connection by the receiver as indicated in the RESET protocol element being acknowledged.

The Information field is empty.

11.4.3.7 DATA

This protocol element is transmitted to carry an RCLLC service data unit from the source SAP to the destination SAP.

The Param 0 field contains the sequence number of the element, cf. section 11.4.2. The sequence number, which can only be 0 or 1, is placed in bit 0. The remaining bits are all 0.

The Param 1 field contains the index assigned to the connection at the destination RCLLC entity.

The Information field contains the RCLLC service data unit.

11.4.3.8 ACK

This protocol element is transmitted to acknowledge receipt of a DATA protocol element on a connection.

The Param 0 field contains the sequence number of the element being acknowledged.

The Param 1 field contains the index assigned to the connection at the destination RCLLC entity, i.e. the sender of the DATA element.

The Information field is empty.

12. iSBX Bus Specification

The iSBX bus is a unique interface facilitating on-board expansion with iSBX Multimodule boards. The iSBX bus is derived directly from the on-board CPU bus and, as such, an iSBX Multimodule board plugged into the iSBX bus becomes an integral element of the PICCOLINE computer. The physical interface between the single board computer and the iSBX Multimodule board is a unique connector designed specifically for the iSBX bus. The iSBX bus is brought out to a female iSBX bus connector on the computer and mates with its male equivalent resident on the iSBX Multimodule board (fig.12.1) page 146.

The iSBX Multimodule board concept offers a unique design approach to board level users. The iSBX Multimodule boards bring a new concept to expansion, providing a product family of smaller modules that can be plugged directly onto the single board computer. In short, the user may now tailor his application directly onboard the single board computer at a minimal cost. In addition, the iSBX Multimodule boards offer maximum performance because they are tightly coupled to the microprocessor through the iSBX bus.

This chapter has been prepared for those users who intend to evaluate or design custom iSBX Multimodule board products that will be compatible with RC759 base board. The chapter defines the logical, electrical, and mechanical aspects of the iSBX Multimodule boards. The iSBX Multimodule board specifications are defined in a similar way an I/O component would be.

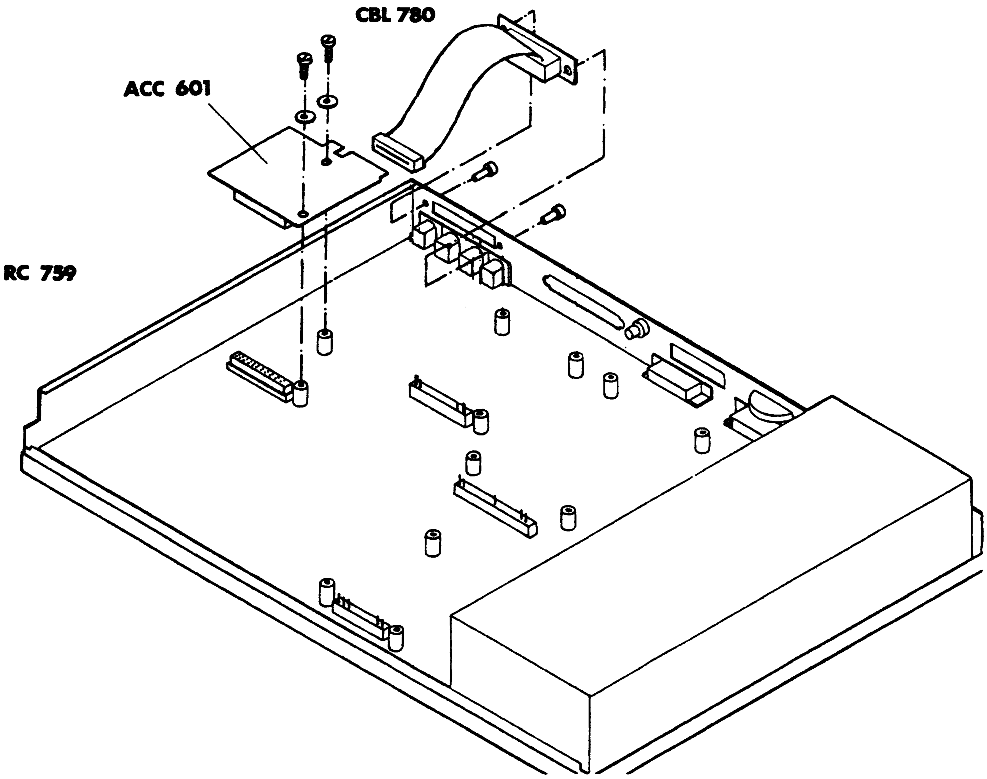


Fig.12.1 iSBX Multimodule Board Concept

12.1 Functional Description

This section will give the reader an overall understanding of how the iSBX Multimodule board functions. It describes the basic elements of an iSBX Multimodule board, defines the iSBX Multimodule interface signals and describes the basic communication operations.

In this section, as well as throughout the specification, a clear and consistent notation for signals has been used. The I/O Read (IORD) signal will be used to explain this notation. The terms one, zero, true, and false can be ambiguous, so their use will be avoided. In their place, the terms electrical High and Low (H and L) will be used. A slash following a signal name (IORD/) indicates that the signal is active low as shown:

IORD/ = IORD = $\overline{\text{IORD}}$- = Asserted at 0 volts

The signal (IORD/), driven by a three state driver will be pulled up to VCC when not asserted. Fig.12.2 is used to further explain the notation used in this specification.

Signal Name	Electrical	Definition	
		Logical	State
IORD	H L	1 True 0 False	Active, Asserted
IORD/	L H	1 True 0 False	Active, Asserted

Fig.12.2 Notational Summary

12.1.1 iSBX Multimodule System Elements

This section will describe the two basic elements in an iSBX Multimodule system: base boards and iSBX Multimodule boards (see fig.12.1).

12.1.1.1 Base Boards

The base board provides an electrical and mechanical interface for the iSBX Multimodule boards. The electrical interface provides the communication link between the two elements. The base board is the master of this link, in that it controls the address and command signals. The base board also provides the mounting for the iSBX Multimodule board. With the aid of screws, spacers, nuts, and the iSBX connector, the iSBX Multimodule board is mounted to the base board.

There are two classes of base boards: those with Direct Memory Access (DMA) support and without.

Base boards with DMA support are boards with DMA controllers on them. These boards, in conjunction with an iSBX Multimodule board (with DMA capability), can perform direct I/O to memory or memory to I/O operations. Base boards without DMA support use a subset of the iSBX bus and simply do not use that aspect of the iSBX Multimodule board.

12.1.1.2 iSBX Multimodule Boards

The iSBX Multimodule boards are small, specialized, I/O mapped boards which plug into base boards. The iSBX boards connect to the iSBX bus connector and convert the iSBX bus signals to a defined I/O interface.

12.1.2 iSBX Bus Interface

The iSBX bus interface can be grouped into six functional classes:

Control Lines
Address and Chip Select Lines
Data Lines
Interrupt Lines
Option Lines
Power Lines

12.1.2.1 Control Lines

The following signals are classified as control lines:

COMMANDS:
- IORD/ (I/O Read)
- IOWRT/ (I/O Write)

DMA:
- MDRQT (DMA Request)
- MDACK/ (DMA Acknowledge)
- TDMA (Terminate DMA)

INITIALIZE:
- RESET

CLOCK:
- MCLK (iSBX Multimodule Clock)

SYSTEM CONTROL:
- MWAIT/
- MPST/ (iSBX Multimodule Board Present)

Command Lines (IORD/, IOWRT/)

The command lines are active low signals which provide the communication link between the base board and the iSBX Multimodule board. An active command line, conditioned by chip select, indicates to the iSBX Multimodule board that the address lines are valid and the iSBX Multimodule board should perform the specified operation.

DMA Lines (MDRQT, MDACK/, TDMA)

The DMA lines are the communication link between the DMA controller device on the base board and the iSBX Multimodule board. MDRQT is an active high output signal from the iSBX Multimodule board to the base board's DMA device requesting a DMA cycle. MDACK/ is an active low input signal to the iSBX Multimodule board from the base board DMA device acknowledging that the requested DMA cycle has been granted. TMDA is used by the iSBX Multimodule board to terminate DMA activity. The use of the DMA lines is optional as not all base boards will provide DMA channels and not all iSBX Multimodule boards will be capable of supporting a DMA channel.

Initialize Lines (Reset)

This input line to the iSBX Multimodule board is generated by the base board to put the iSBX Multimodule board into a known internal state.

Clock Lines (MCL)

This input to the iSBX Multimodule board is a timing signal. The clock frequency is 10 MHZ. This clock is asynchronous from all other iSBX bus signals.

System Control Lines (MWAIT/, MPST/)

These output signals from the iSBX Multimodule board control the state of the system.

Active MWAIT/ (Active Low) will put the CPU on the board into a wait state providing additional time for the iSBX Multimodule board to perform the requested operation. MWAIT/ must be generated from address (address plus chip select) information only. If MWAIT/ is driven active due to a glitch on the CS line during address transitions, MWAIT/ must be driven inactive in less than 75 ns.

The iSBX Multimodule board present (MPST/) is an active low signal (tied to signal ground) that informs the base board I/O decode logic that an iSBX Multimodule board has been installed.

12.1.2.2 Address and Chip Select Lines

The address and chip select lines are made up of two groups of signals.

Address Lines:	MA0-MA2
Chip Select Lines:	MCS0/-MCS1/

The base board decodes I/O addresses and generates the chip selects for the iSBX Multimodule boards. The base board decodes all but the lower order three addresses in generating the iSBX Multimodule board chip selects. Thus, a base board would normally reserve two blocks of 8 I/O ports for each iSBX socket it provides.

Address Lines (MA0-MA2)

These positive true input lines to the iSBX Multimodule boards are generally the least three significant bits of the I/O address. In conjunction with the commmand and chip select lines, they establish the I/O port address being accessed.

Chip Select Lines (MCS0/-MCS1/)

These input lines to the iSBX Multimodule board are the result of the base board I/O decode logic. MCS/ is an active low signal which conditions the I/O command signals and enables communication with the iSBX Multimodule boards.

NOTE

> If MCS/ glitches, the MWAIT/ line may also glitch. MWAIT/ must be in its proper state in less than t_{cw} (75 ns) after MCS/ is in its proper state.

12.1.2.3 Data Lines (MD0-MD7)

Eight bidirectional data lines (active high) are used to transmit or receive information to or from the iSBX Multimodule ports. MD0 is the least significant bit.

12.1.2.4 Interrupt Lines (MINTR0-MINTR1)

These active high output lines from the iSBX Multimodule board are used to make interrupt requests to the base board.

12.1.2.5 Option Lines (OPT0, OPT1)

These two signals are two reserved lines that are connected to wire wrap posts on both the base board and iSBX Multimodule board. They are for unique requirements where a user needs a base board signal on the iSBX Multimodule board and is willing to put a potentially long wire on the base board to connect it.

12.1.2.6 Power Lines

All base boards will provide +5 and +12 /-12 volts to the iSBX Multimodule boards.

12.1.3 iSBX Multimodule Command Operations

The command lines are driven from the base board by tri-state drivers with pull-up resistors or standard TTL totem pole drivers. These lines indicate to the iSBX Multimodule board what action is being requested.

12.1.3.1 I/O READ

There are two I/O READ operations that a base board can perform. The iSBX Multimodule board determines which type of I/O READ is performed. The first type is a full speed I/O READ (fig.12.3). The base board generates a valid I/O address and a valid chip select for the iSBX Multimodule board. After the set up timings are met, the base board activates the IORD line. The iSBX Multimodule board must generate valid data from the addressed I/O port in less than 250 ns. The base board then reads the data and removes the read command, address, and chip selects shown in the timing diagram.

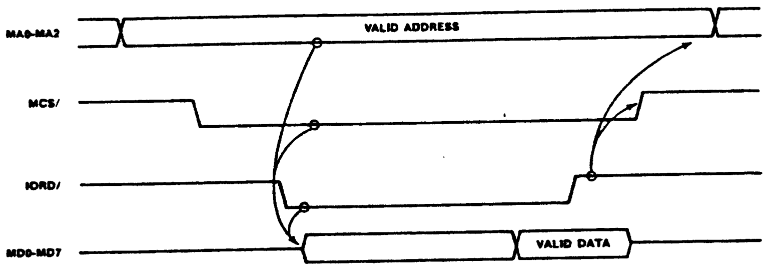


Fig.12.3 iSBX Multimodule Board Read, Full Speed

The second type of I/O READ is an extended read (fig.12.4). This type of read is used by iSBX Multimodule boards that cannot perform a READ operation under the full speed specifications. The base board generates a valid address and chip select, just as in a full speed read. The iSBX Multimodule board then activates the MWAIT/ signal which in turn deactivates the ready input to the CPU (putting it into a WAIT state). The iSBX Multimodule board will remove the MWAIT/ signal when valid READ data is on the iSBX Multimodule data bus. The base board then reads the data and deactivates the command, address, and chip select.

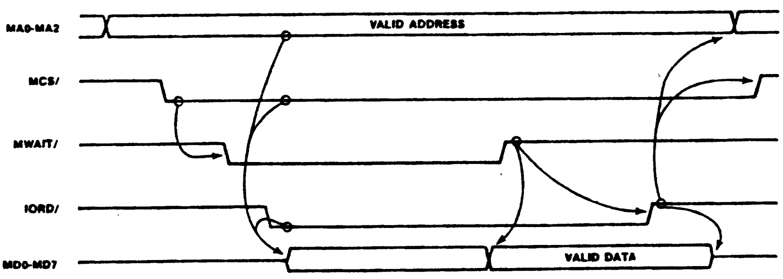


Fig.12.4 iSBX Multimodule Board Extended Read

12.1.3.2 I/O WRITE

There are two I/O WRITE operations that a base board can perform. The iSBX Multimodule board determines which type of I/O WRITE is performed.

The first type of write is a full speed I/O WRITE (fig.12.5). The base board generates a valid I/O address and chip select. The base board activates the IOWRT line after the set up times are met. The IOWRT/ line will remain active for 300 ns and the data will be valid for 250 ns before the IOWRT/ command is removed. The base board will then remove the data address and chip select after it meets the hold times as shown in fig.12.5.

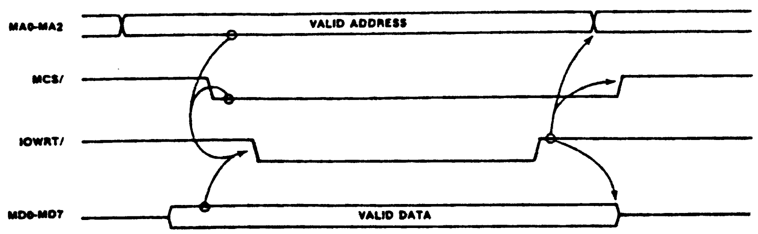


Fig.12.5 iSBX Multimodule Board Write, Full Speed

The second type of I/O WRITE is an extended write (fig.12.6). This write is used by iSBX Multimodule boards that cannot write into an I/O port with the full speed specifications. The base board again generates valid address and chip selects. The iSBX Multimodule board will activate the MWAIT/ signal based on address information (chip select + MA0-1). This will remove the ready from the CPU causing it to go into a wait state after the WRITE command has been activated and valid data provided. The iSBX Multimodule board will remove the MWAIT/ signal (allowing the CPU to leave its wait state) when it has satisfied its write pulse width requirement. The base board will then remove the WRITE command, then the data, address, and chip select after the hold times are met.

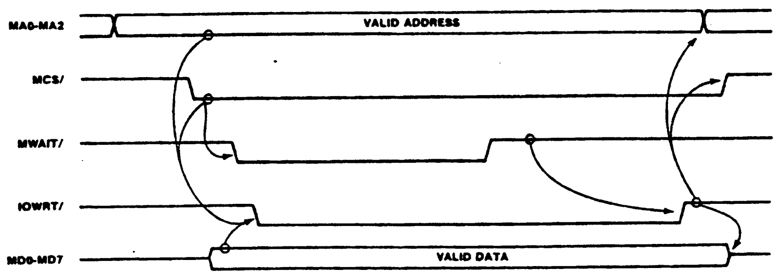


Fig.12.6 iSBX Multimodule Board Extended Write

12.1.3.3 Direct Memory Access (DMA)

An iSBX Multimodule system can support DMA when the base board has a DMA controller and the iSBX Multimodule board can support DMA mode. The following example is for a base board using an 8257 DMA controller. Because of the similarity between DMA reads and DMA writes, only the DMA write is given in the following example. A DMA cycle is initiated when the iSBX Multimodule board activates MDRQT, which goes to the DMA controller on the base board (fig.12.7). Once the DMA controller gains control of the base board bus, it acknowledges back to the iSBX Multimodule board with MDACK/. The DMA controller then activates a memory write or I/O write respectively. The delay may be zero, if the memory is a trailing edge type (data is written when the write pin changes from active to inactive state). The MDACK/ signal must act as a chip select and address to the iSBX Multimodule board (the MCS and MA0-MA1 signals are undetermined as they are driven by the memory address). The iSBX Multimodule board will remove the DMA request during the cycle to stop the DMA cycle. Once the write operation is complete (MWAIT inactive and memory acknowledge active), the DMA controller deactivates the write command and the read command providing a data hold time. If the DMA request signal was removed, the controller will release the base board bus back to the CPU and remove MDACK/. If the request is not removed, the DMA controller will proceed to do another DMA cycle (burst mode).

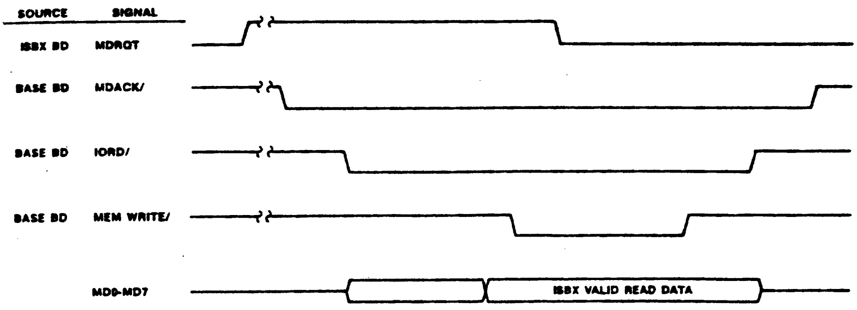


Fig.12.7 iSBX Multimodule Board DMA Cycle

(iSBX Multimodule to Base Board Memory)

12.1.4 RC759 Interface

This section gives information in details about the RC759 address decoder and interrupt circuit.

12.1.4.1 Address Decoder

On fig.12.9 is shown the relationship between the RC759 I/O addresses and the signals in the iSBX connector.

Remark that DMA acknowledge is generated by an OUTput instruction to a special device number.

12.1.4.2 Status Signals

The state of MPST/ OPT0 and OPT1 can be sensed by the RC759 programmer by an INput instruction to device 70H. This INput returns the following information:

```
bit     7    6    5    4    3    2    1    0
      ┌────┬────┬────┬────┬────┬────┬────┬────┐
      │ x  │ x  │ x  │ x  │    │    │    │ x  │
      └────┴────┴────┴────┴─┬──┴─┬──┴─┬──┴────┘
                            │    │    │
                            │    │    └──> MPST/
                            │    └──────> OPT0
                            └───────────> OPT1
```

Fig.12.8 Status Signals

MPSTS/ = 0 if an iSBX module is present. The state of OPT1-2 are iSBX dependent.

12.1.4.3 Interrupt Signals

MINTR0 and MINTR1 are connected to the INT1 and INT3 interrupt inputs to 80186 interrupt controller.

MINTR0 has vector type 13
MINTR1 has vector type 15

Both interrupt sources must deliver edge triggered interrupts.

RC759 I/O device number (HEX)	iSBX signal MCSX/	MA2	MA1	MA0	OPERATION
IOBASE + 300	MCS0/ =0	0	0	0	iSBX module dependent
302		0	0	1	
304		0	1	0	
306		0	1	1	
308		1	0	0	
30A		1	0	1	
30C		1	1	0	
30E		1	1	1	
IOBASE + 310	MCS1/ =0	0	0	0	iSBX module dependent
312		0	0	1	
314		0	1	0	
316		0	1	1	
318		1	0	0	
31A		1	0	1	
31C		1	1	0	
31E		1	1	1	
IOBASE + 320		x	x	x	Out to this device number generates MDACK/

IOBASE = 0 for RC759

Fig.12.9 RC759 Device Decoder

12.2 Electrical Specifications

This section will define all electrical specifications for an iSBX Multimodule board. First the ac timing is specified and then the dc specifications are described.

12.2.1 General Bus Considerations

Fig.12.10 shows the relationship between logical and electrical states.

12.2.2 Power Supply Specifications

All power supply voltages are + 5%.

Minimum (volts)	Nominal (volts)	Maximum (volts)	Maximum (current)
+4.75	+5.0	+5.25	1.0 A
+11.4	+12	+12.6	0.3 A
−12.6	−12	−11.4	0.3 A
—	GND	—	2.0 A

12.2.3 Environmental

All bus specifications should be met while the environment is within the following ranges:

Signal Name	Logical State	Electrical Signal Level	At Receiver	At Driver
IORD/	0	H = TTL High State	5.25 ≥ H ≥ 2.0V	5.25 ≥ H ≥ 2.4V
IORD/	1	L = TTL Low State	0.8 ≥ L ≥ −0.5V	0.5 ≥ L ≥ 0V
IORD	0	L = TTL Low State	0.8 ≥ L ≥ −0.5V	0.5 ≥ L ≥ 0V
IORD	1	H = TTL High State	5.25 ≥ H ≥ 2.0V	5.25 ≥ H ≥ 2.4V
Vcc = 5 volts ±5% referenced to logical ground. V = volts.				

Fig.12.10 Logical and Electrical States

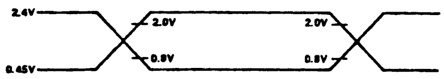
Temperature:	0-55 C (32-131 F) Free moving air across the base board and iSBX Multimodule board.
Humidity:	90% max relative (no condensation).
Shock:	30 g's of force for an 11 msec duration 3 times in 3 planes both sides (total of 18 drops).
Vibration:	Sweeping from 10 Hz to 55 Hz and back to 10 Hz at a distance of 0.010 inches peak-to-peak lasting 15 minutes in each of three planes.

12.2.4 Timing

Fig.12.11 summarizes all the ac timing specifications. The timing diagrams are shown in fig.12.12 through 12.15.

NOTE

The input waveforms for the ac timing specifications are as follows:



Symbol	Parameter	Min (ns)	Max (ns)	Figure Reference
t_1	Address stable before read	50	—	13
t_2	Address stable after read	30	—	13
t_3	Read pulse width	300	—	13
$t_4{}^2$	Data valid from read	0	250	13
$t_5{}^2$	Data float after read	0	150	13
t_6	Time between RD and/or WRT	—	Note 3	
t_7	CS stable before CMD	25	—	13
t_8	CS stable after CMD	30	—	13
t_9	Power up reset pulse width	50 Msec	—	15
t_{10}	Address stable before WRT	50	—	12
t_{11}	Address stable after WRT	30	—	12
$t_{12}{}^2$	Write pulse width	300	—	12
$t_{13}{}^2$	Data valid to write	250	—	12
t_{14}	Data valid after write	30	—	12
t_{15}	MCLK cycle	100	110	15
t_{16}	MCLK width	35	65	15
$t_{17}{}^1$	MWAIT/ pulse width	0	4 msec	12, 13
t_{18}	Reset pulse width	10 Msec	—	15
t_{19}	MCS/ to MWAIT/ valid	0	75	12, 13
t_{20}	DACK set up to I/O CMD	100	—	14
t_{21}	DACK hold	30	—	14
t_{22}	CMD to DMA RQT removed to end of DMA cycle	—	200	14
t_{23}	TDMA pulse width	500	—	14
$t_{24}{}^1$	MWAIT/ to valid read data	—	0	13
$t_{25}{}^1$	MWAIT/ to WRT CMD	0	—	12

NOTES:
1. Required only if WAIT is activated.
2. If MWAIT/ not activated.
3. To be specified by each iSBX Multimodule board.

Fig.12.11 iSBX Multimodule Board I/O AC Specifications

12.4.5 DC Specifications

The dc specifications for the iSBX bus are summarized in fig.12.16. The figure is divided into two sections, output specifications and input specifications. The output specifications are the requirements on the output drivers of the iSBX Multimodule board (i.e., the data bus output drivers must guarantee at least 1.6 mA @ 0.5 volts). The output specifications in fig.12.16 are the minimum drive requirements. The input specifications are the requirements of the receivers on the iSBX Multimodule board (e.g., the loading of the address lines (MA0-MA2) can be no greater than 0.5 mA @ 0.8 volts). Fig.12.16 also summarizes the maximum loading permitted on an iSBX Multimodule interface at any one time.

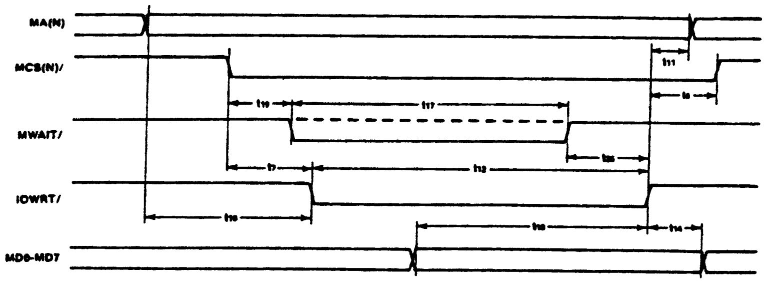


Fig.12.12 iSBX Multimodule Board I/O Write Timing

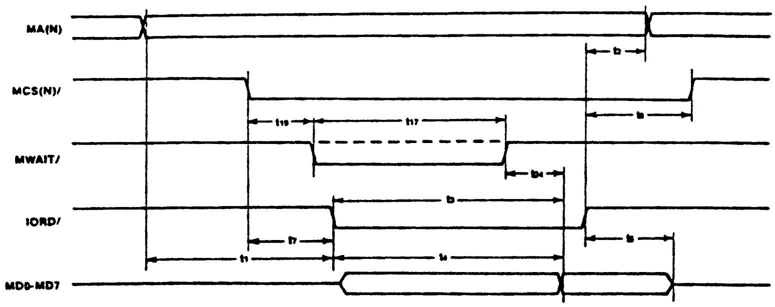


Fig.12.13 iSBX Multimodule Board I/O Read Timing

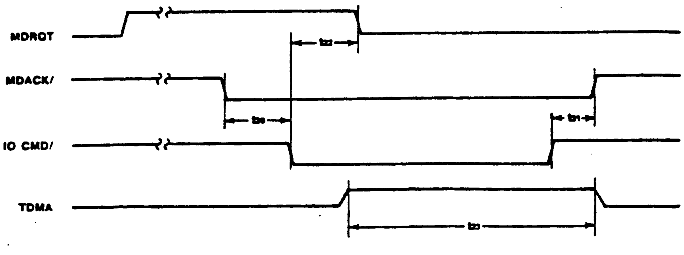


Fig.12.14 iSBX Multimodule Board I/O DMA Timing

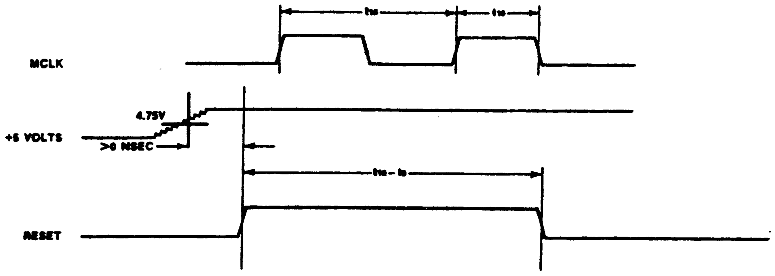


Fig.12.15 iSBX Multimodule Board I/O Reset Timing

Output

Bus Signal Name	Type[2] Drive	I_{OL} Max −Min (mA)	@ Volts (V_{OL} Max)	I_{OH} Max −Min (μA)	@ Volts (V_{OH} Min)	Co (Min) (pf)
MD0-MD7	TRI	1.6	0.5	−200	2.4	130
MINTR0-1	TTL	2.0	0.5	−100	2.4	40
MDRQT	TTL	1.6	0.5	− 50	2.4	40
MWAIT/	TTL	1.6	0.5	− 50	2.4	40
OPT1-2	TTL	1.6	0.5	− 50	2.4	40
MPST/	TTL	Note 3				

Input

Bus Signal Name	Type[2] Receiver	I_{IL} Max (mA)	@ V_{IN} Max (volts)	I_{IH} Max (μA)	@ V_{IN} Max (volts)	Ci Max (pf)
MD0-MD7	TRI	−0.5	0.8	70	2.0	40
MA0-MA2	TTL	−0.5	0.8	70	2.0	40
MCS0/-MCS1/	TTL	−4.0	0.8	100	2.0	40
MRESET	TTL	−2.1	0.8	100	2.0	40
MDACK/	TTL	−1.0	0.8	100	2.0	40
IORD/ IOWRT/	TTL	−1.0	0.8	100	2.0	40
MCLK	TTL	−2.4	0.8	100	2.0	40
OPT1-OPT2	TTL	−2.0	0.8	100	2.0	40

NOTES:

2. TTL = standard totem pole output. TRI = Three-state.
3. iSBX Multimodule board must connect this signal to ground.

Fig.12.16 iSBX Multimodule Board I/O DC Specifications

12.3 Mechanical Specifications

This sections describes all the physical attributes of an iSBX Multimodule board.

12.3.1 iSBX Connector

The male iSBX connector is attached to the iSBX Multimodule board and the female iSBX connector is attached to the base board. Fig.12.17 is an outline drawing of the iSBX connector and also shows the pin numbering. Fig.12.19 lists the signal pin assignments.

12.3.2 iSBX Multimodule Board Height Requirement

Fig.12.18 shows the iSBX Multimodule board height requirements. The total board height minus the iSBX connector is:

Maximum component height (0.400 Max)	0.400
P.C. board thickness (0.62 + 0.005)	0.067
Component lead length (0.093 Max)	0.093
	0.560 in.

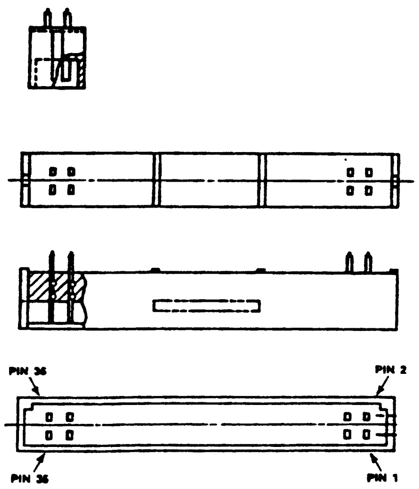


Fig.12.17 iSBX Connector

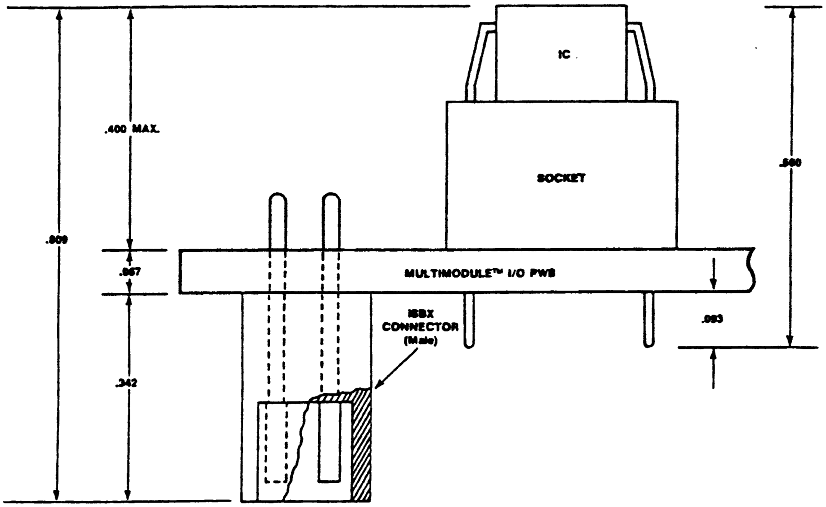


Fig.12.18 iSBX Multimodule Board Height

Pin	Mnemonic	Description	Pin	Mnemonic	Description
35	GND	Signal Ground	36	+5V	+5 Volts
33	MD0	MDATA Bit 0	34	MDRQT	M DMA Request
31	MD1	MDATA Bit 1	32	MDACK/	M DMA Acknowledge
29	MD2	MDATA Bit 2	30	OPT0	Option 0
27	MD3	MDATA Bit 3	28	OPT1	Option 1
25	MD4	MDATA Bit 4	26	TDMA	Terminate DMA
23	MD5	MDATA Bit 5	24		Reserved
21	MD6	MDATA Bit 6	22	MCS0/	M Chip Select 0
19	MD7	MDATA Bit 7	20	MCS1/	M Chip Select 1
17	GND	Signal Gnd	18	+5V	+5 Volts
15	IORD/	I/O Read Cmd	16	MWAIT/	M Wait
13	IOWRT/	I/O Write Cmd	14	MINTR0	M Interrupt 0
11	MA0	M Address 0	12	MINTR1	M Interrupt 1
9	MA1	M Address 1	10		Reserved
7	MA2	M Address 2	8	MPST/	iSBX Multimodule Board Present
5	RESET	Reset	6	MCLK	M Clock
3	GND	Signal Gnd	4	+5V	+5 Volts
1	+12V	+12 Volts	2	−12V	−12 Volts
All undefined pins are reserved for future use.					

Fig.12.19 iSBX Signal Pin Assignments

12.3.3 iSBX Multimodule Board Outline

The iSBX Multimodule boards will have two standard board outlines. Fig.12.20 and 12.21 show the iSBX Multimodule board outlines.

12.3.4 iSBX Multimodule Board User I/O Connector Outlines

The top of the iSBX Multimodule board can be defined by the user. Fig.12.22 through 12.24 show the dimensions of suggested top edge connectors for the most common designs.

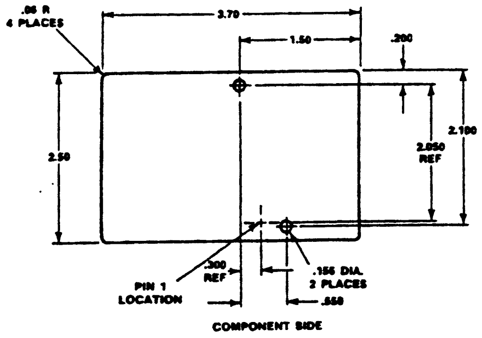


Fig.12.20 iSBX Multimodule Board Outline

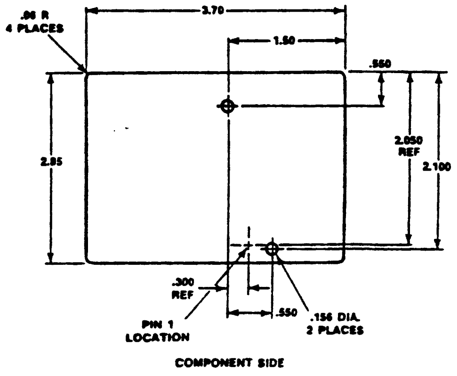


Fig.12.21 iSBX Multimodule Board Outline

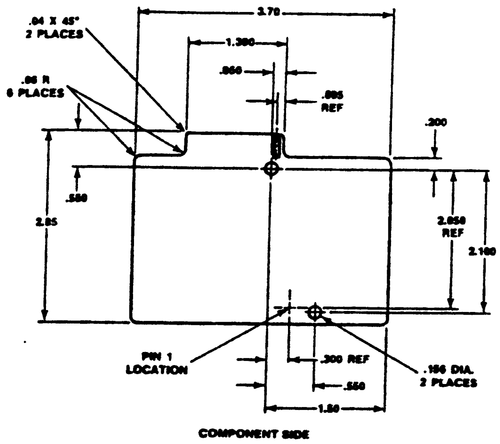


Fig.12.22 13/26 Pin Connector

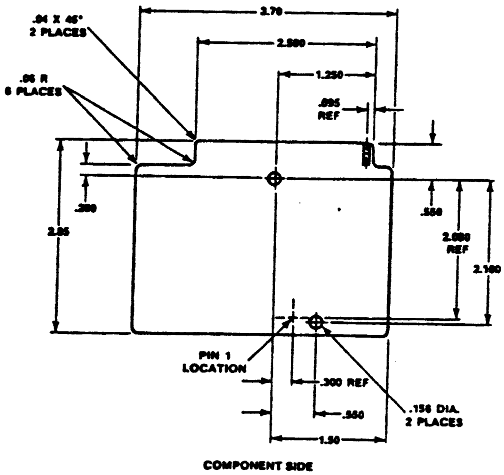


Fig.12.23 20/50 Pin Connector

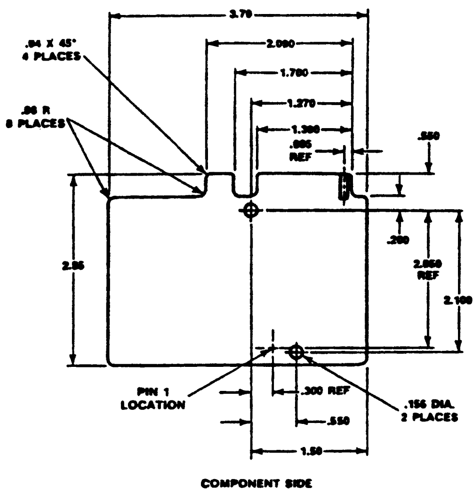


Fig.12.24 13/26 and 20/40 Pin Connector

12.4 Design Example

This section provides a functional description of a design example. The design example that will be used is an Serial Multimodule Board. The functional description includes details on the RS232C and RS422/449 communications interface signals, the interface signals between the iSBX Multimodule board and the host microcomputer, and the clock generation hardware on the iSBX Multimodule board. Fig.12.25 shows a block diagram of the Communication Multimodule board.

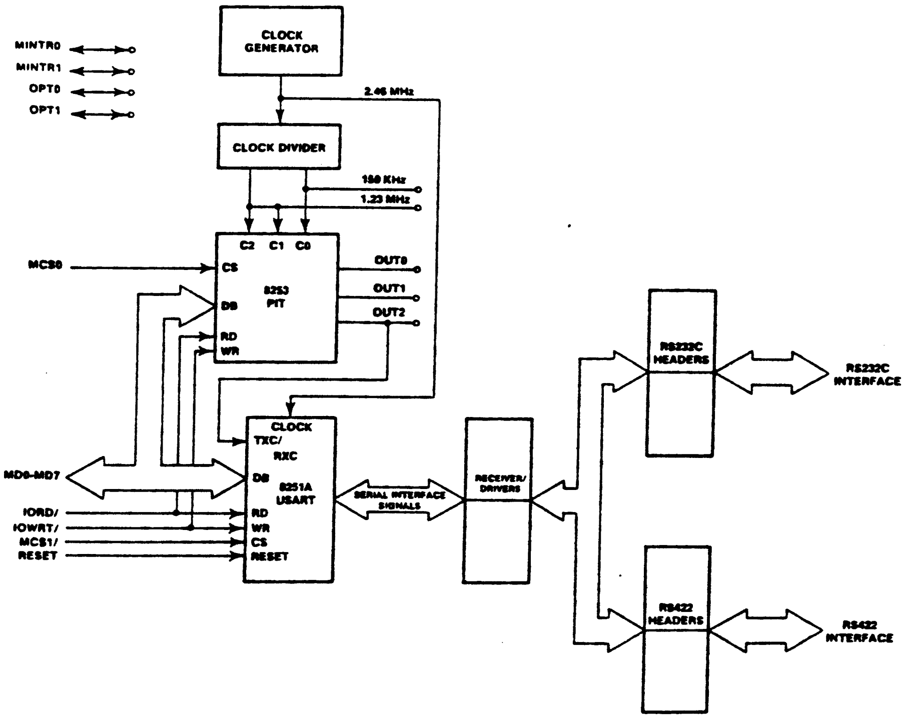


Fig.12.25 iSBX Board Block Diagram

12.4.1 Serial I/O Communications Channel Interface

The communications interface on the iSBX Multimodule board may be configured for either RS232C or RS422/449 operation via jumper modifications. Default wiring of the iSBX Multimodule board is for RS232C operation. To convert to RS422/449 operation, move the two 8-circuit shorting plugs from sockets XU6 and XU7 to XU4 and XU5.

The serial interface provides RS232C or RS422 buffers for eight lines. These lines are the Data In, Data Out, Request to Send, Clear to Send, Data Set Ready, Data Terminal Ready, Receive Clock, and DTE Transmit Clock. All necessary driver and receiver chips are supplied with the board.

12.4.2 CPU Interface

The interface between the host microcomputer and the iSBX Multimodule board consists of several signals that are defined in the following paragraphs. The DC characteristics for these signals are given in fig.12.26.

RESET (Reset). This active high input signal to the 8251A **USART** places the **USART** chip into the IDLE mode until a new set of control words is written to the chip.

MA0 (Address bit 0). This active high input to the 8251A USART and to the 8253 is used in conjunction with IORD/ and IOWRT/ signals to define which register on the 8251A or 8253 is addressed.

MA1 (Address bit 1). This active high input signal to the 8253 isused in conjunction with MA0 to select one of the counters to be operated on in 8253 and to address the control word register for mode selection.

IORD/ (I/O Read). This active low input signal to the iSBX Multimodule board performs one of two functions depending on the chip selected. When low, IORD/ informs the 8251A that the host iSBC microcomputer is reading data or status from the 8251A, and it informs the 8253 that the host iSBC microcomputer is reading the value of a counter.

IOWRT/ (I/O Write). This active low input to the iSBX Multimodule board may perform one of two functions dependent on chip select. When low, IOWRT/ informs the 8251A that the host microcomputer is writing data or control words to the 8251A. IOWRT/ also informs the 8253 that the host microcomputer is outputting mode information or loading counters.

MCS0/ (Chip Select). This active low input signal to the 8251A USART enables it to perform read and write operations. When MCS0/ is high, the USART bus is held in a float state and the IORD/ and IOWRT/ signals do not effect the USART.

MCS1/ (Chip Select). This active low input signal to the 8253 PIT enables it to perform read and write operations. However, MCS1/ has no effect on the operation of the internal counters in the 8253.

MD0-MD7 (Bidirectional Data Bus). These active high I/O lines are the iSBX Multimodule boards' tie-in to the host iSBC microcomputer data bus. MD0 through MD7 transfer data, commands, and status between the iSBX Multimodule board and the host iSBC microcomputer.

MINTR0, MINTR1 (Interrupt Request Lines). These active high output lines may be jumpered to OUT 0, or OUT 1 on the 8253, or to TXRDY on the 8251A.

OPT0, OPT1 (Option Lines). These active high I/O lines are included to give the iSBX Multimodule board greater functional flexibility. These lines may be user-configured for special functions.

Output

Bus Signal Name	Type Drive	I_{OL} Max (mA)	V_{OL} Max I_{OL} = Max	I_{OH} Max (μA)	V_{OH} Min I_{OH} = Max	Co (Min) (pf)
MD0-MD7	TRI	2.2	0.45	−390	2.4	130
MINTR0-1	TTL	2.2	0.45	−200	2.4	40
OPT0-1	TTL	2.2	0.45	−200	2.4	40

Input

Bus Signal Name	Type Receiver	I_{IL} Max (mA) V_{IL} = 0.45V	V_{IL} Max	I_{IH} Max (mA) V_{IH} = 2.4V	V_{IH} Min	Ci (Max) (pf)
MD0-MD7	TRI	−0.02	0.8	0.02	2.2	40
MA0-1	TTL	−0.02	0.8	0.02	2.2	20
MCS0/-1/	TTL	−0.01	0.8	0.01	2.2	20
RESET	TTL	−0.01	0.8	0.02	2.0	20
IORD/, IOWRT/	TTL	0.02	0.8	0.02	2.2	40
OPT0-1	TTL	−1.6	0.8	0.02	2.2	40

TTL = Standard totem pole output.
TRI = Three state output.

Fig.12.26 DC Characteristics

12.4.3 Interface Buffering

Interface buffering is provided by three receiver/driver logic elements U1, U2, and U3. U1 is an input buffer that may be used with either RS232C or RS442 configuration, depending on the position of the mode selection header blocks. U2 provides RS422 output buffering, and U3 provides RS232C output buffering.

12.4.4 Clock Generation Circuitry

The Communication 351 board includes an 8224 Clock Generator chip that creates a 2.46 MHz output from a 22.1148 MHz crystal input. The output is then passed through a synchronous four-bit counter which generates a 1.23 MHz clock and a 153.6 KHz clock to drive the 8253 PIT. The clock output frequency labeled OUT 2, which is produced by the 8253 PIT, will vary according to the configuration and programming of the PIT chip.

The two remaining clock frequencies output from the 8253 PIT are jumper selectable to generate interrupts for the iSBX Multimodule board.

12.4.5 AC Specifications

The ac specifications for the Communication Multimodule Board are listed in fig.12.27. Fig.12.28 and 12.29 define the timing parameters for the board.

Symbol	Parameter	Min (ns)	Max (ns)
t_1	Address stable before IORD/	50	—
t_2	Address stable after IORD/	30	—
t_3	READ pulse width	300	—
t_4	Data valid from IORD/	—	250
t_5	Data float after IORD/	0	100
t_6(1)	Time between commands	1000	—
t_7	CS stable before CMD	25	—
t_8	CS stable after CMD	30	—
t_9	Address stable before IOWRT/	50	—
t_{10}	Address stable after IOWRT/	30	—
t_{11}	WRITE pulse width	300	—
t_{12}	Data valid to IOWRT/	250	—
t_{13}	Data valid after IOWRT/	30	—
t_{14}	Reset pulse width	2.9 msec	—

NOTES:
1. During initialization, all writes to the control port: t_6 = 1.92 μs. After initialization in asynchronous mode all writes to the control port: t_6 = 2.56 μs. After initialization in synchronous mode all writes to the control port: t_6 = 5.12 μs. All writes to the data port: Depends upon the baud rate since TXRDY must be true.

Fig.12.27 AC Specifications

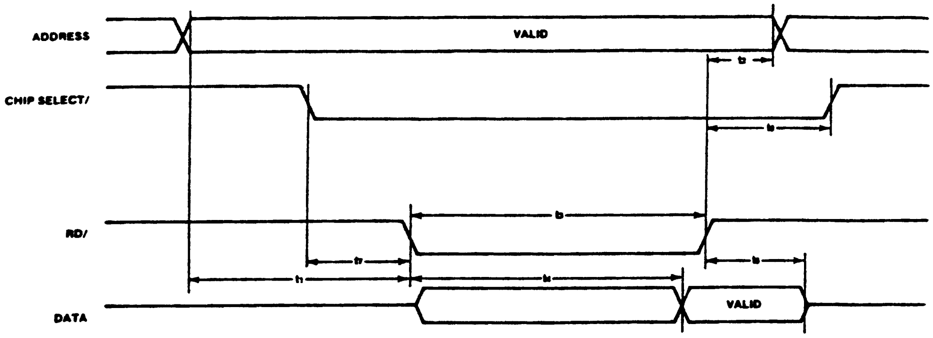


Fig.12.28 READ Timing

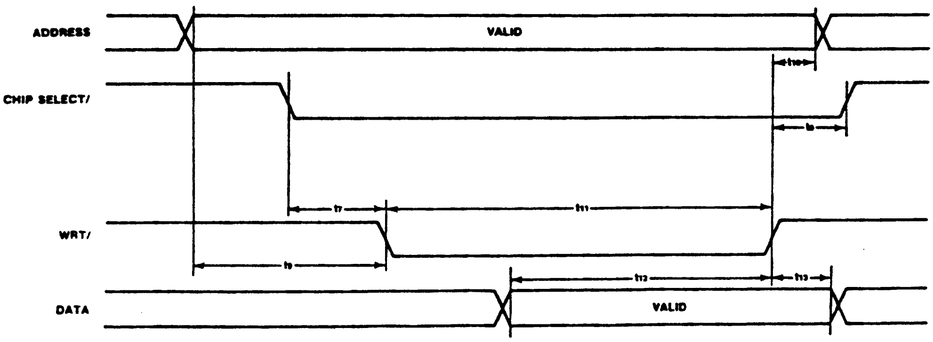


Fig.12.29 WRITE Timing

13. Inter Computer File Transfer

This chapter provides technical information concerning the FILEX file transfer program which in its standard form can be used to transfer files between a PICCOLINE and one of the following computers:

1) Another PICCOLINE (RC759)
2) An RC750 Partner
3) An RC702 Piccolo
4) An RC703 Piccolo
5) An RC855 Workstation

Together with the FILEX source program included on the PICCOLINE distribution disk, this chapter contains the necessary information for an experienced user to modify FILEX or implement a FILEX type file transfer program on another computer with serial communication support (e.g. an IBM PC with SYNC/ASYNC controller option installed).

13.1 Requirements

Since the FILEX file transfer program is based on serial communication the PICCOLINE system has to be enhanced with an iSBX351 serial interface (V24-interface).

The two computers on which FILEX is to run must be connected by means of an appropiate cable.

To connect two computers, arbitrarily chosen among the RC702, RC703, RC855, RC750 and RC759, one of the following cables should be used:

1)	CBL912	(5 metres)
2)	CBL913	(12 metres)
3)	CBL914	(25 metres)

Furthermore, the user should configurate the two selected computers to ensure:

1) that the two computers use the same baudrate on the channel used,
2) that the line character format is set to 7 bits per character.

13.2 How FILEX Works

FILEX type file transfers take place as follows.

The local computer sends a number of transactions to the remote computer. Each time the remote computer receives a transaction, it carries out the appropriate file operation and sends an answer back to the local computer. The transactions sent depend upon whether the file is to be transferred to or from the local computer (see the FILEX program listing for details).

The entire set of transactions and the transmission protocol are described in the following.

13.2.1 FILEX Transactions

The effect of the file operations below is as described in ref.2.

OPEN

Request	Field	Answer
1	opcode	1
0	unused	0
0	result	result
file name	name 16 byte	

MAKE

Request	Field	Answer
2	opcode	2
0	unused	0
0	result	result
file name	name 16 byte	

READ

Request	Field	Answer
3	opcode	3
0	unused	0
0	result	result
	area 128 byte	area

WRITE

Request	Field	Answer
4	opcode	4
0	unused	0
0	result	result
area	area 128 byte	

CLOSE

Request	Field	Answer
5	opcode	5
0	unused	0
0	result	result

END

Request	Field	Answer
6	opcode	6
0	unused	0
0	result	result

13.2.2 Transmission protocol

The transactions described in 13.2.1 are sent by means of the blocked tranmission protocol described below.

A block consists of the following elements:

1) start character:
ASCII value 35

2) Block size:
The size defines the number of characters (N) in the string to be sent, not the number of characters necessary to send the string (2*N+8, explained below). The block size is a 16-bit integer (0..65535) split into four 4-bit digits. Each digit is interpreted as an integer to which 64 has been added, so that the resulting value lies between 64 and 79. These values are transmitted as characters, the most significant part first, the least significant part last.

3) Data section:
Each character in the string to be sent is split into two 4-bit digits, to which 64 is added, as above. These two integers are transmitted as ASCII values, the most significant part first.

4) Checksum:
An 8-bit number which is transmitted as two ASCII values as explained above. The checksum is calculated so that the following condition is satisfied:

((the sum of the values of the characters in the original string) + checksum) modulo 256 = 0.

5) Stop character:
ASCII value 13.

If the number of characters in the string to be transmitted is N, then the actual number of characters transmitted are:

```
1   (start character)
+
4   (block size)
+
2*N (data section)
+
2   (checksum)
+
1   (stop character)

= 2*N + 8 characters.
```

A. Int-28h Function Interface

Function 0

Changes the console mode to graphics mode.

Registers on entry:

AL	0
AH	1=high resolution/2=medium resolution
DX	Address segment of graphics control block.
CX	Address offset of graphics control block.

Registers on return:

Undefined

See 4.5.1.

Function 1

Changes the console mode to character mode.

Registers on entry:

AL	1

Registers on return:

Undefined

See 4.5.2.

Function 2

Reserved

Function 3

Returns the address of a copy of the nonvolatile memory contents.

Registers on entry:

AL	3

Registers on return:

ES	Address segment
SI	Address offset

See 3.2.

Function 4

Returns the address of a configuration description.
Registers on entry:

AL	4

Registers on return:

ES	Address segment
SI	Address offset

See 3.1.

Function 5

Recalibrate floppy disk drive.

Registers on entry:

AL	5

Stack on entry:

+10	Drive (0/1)
+ 8	Head (0/1)
+ 6	Cylinder
+ 4	Bytecount
+ 2	DMA segment
+ 0	DMA offset

Registers on return:

AL	Floppy disk controller status register

Function 6-7

Reserved.

Function 8

Step floppy drive head one track in.

Registers on entry:

AL	8

Stack on entry:

+10	Drive (0/1)
+ 8	Head (0/1)
+ 6	Cylinder
+ 4	Bytecount
+ 2	DMA segment
+ 0	DMA offset

Registers on return:

AL	Floppy disk controller status register

Function 9

Step floppy drive head one track out.

Registers on entry:

AL	9

Stack on entry:

+10	Drive (0/1)
+ 8	Head (0/1)
+ 6	Cylinder
+ 4	Bytecount
+ 2	DMA segment
+ 0	DMA offset

Registers on return:

AL	Floppy disk controller status register

Function 10

Write a track to floppy disk.

Registers on entry:

AL	10

Stack on entry:

+10	Drive (0/1)
+ 8	Head (0/1)
+ 6	Cylinder
+ 4	Bytecount
+ 2	DMA segment
+ 0	DMA offset

Registers on return:

AL	Floppy disk controller status register

Function 11

Read a track from floppy disk.

Registers on entry:

AL	11

Stack on entry:

+10	Drive (0/1)
+ 8	Head (0/1)
+ 6	Cylinder
+ 4	Bytecount
+ 2	DMA segment
+ 0	DMA offset

Registers on return:

AL	Floppy disk controller status register

Function 12

Write a byte to the sound generator.

Registers on entry:

AL	12
DL	byte

Registers on return:

Undefined

Function 13

Get address of disk driver statistics

Registers on entry:

AL	13

Registers on return:

ES	Address segment
BX	Address offset

The disk driver statistics has the following layout:

```
Read_Count      RW 16      ; Each word contain number of
                           ; read operations on the
                           ; corresponding drive (word 0 is
                           ; count for drive A etc.)

Write_Count     RW 16      ; Each word contain number of
                           ; write operations on the
                           ; corresponding drive.

Hard_Err_Read   RW 16      ; Each word contain number of
                           ; non recoverable errors occured
                           ; during read operations on the
                           ; corresponding drive.

Hard_Err_Write  RW 16      ; Each word contain number of
                           ; non recoverable errors occured
                           ; during write operations on the
                           ; corresponding drive.

Soft_Err_Read   RW 16      ; Each word contain number of
                           ; recoverable errors occured
                           ; during read operations on the
                           ; corresponding drive.

Soft_Err_Write  RW 16      ; Each word contain number of
                           ; recoverable errors occured
                           ; during write operations on the
                           ; corresponding drive.

; Floppy controller status bit statistics. First word in
; each field is count for drive A, second field is count
; for drive B. See WD1797 controller manual for details.

Fl_Error_Read   DW 0,0     ; Bit 0 - BUSY
                DW 0,0     ; Bit 1 - DRQ
                DW 0,0     ; Bit 2 - LOST DATA
                DW 0,0     ; Bit 3 - CRC ERROR
                DW 0,0     ; Bit 4 - RECORD NOT FOUND
                DW 0,0     ; Bit 5 - DELETED DATA
                DW 0,0     ; Bit 6 - NOT USED
                DW 0,0     ; Bit 7 - READY
```

```
Fl_Error_Write DW 0,0    ; Bit 0 - BUSY
               DW 0,0    ; Bit 1 - DRQ
               DW 0,0    ; Bit 2 - LOST DATA
               DW 0,0    ; Bit 3 - CRC ERROR
               DW 0,0    ; Bit 4 - RECORD NOT FOUND
               DW 0,0    ; Bit 5 - DELETED DATA
               DW 0,0    ; Bit 6 - NOT USED
               DW 0,0    ; Bit 7 - READY
```

Function 14-18

Reserved

Function 19

Returns 16 mS counter.

To offer a better time resolution than the one second from the real time clock, the XIOS maintains a 32 bit wide second count field and a tick (16 millisecond) count field which together make it possible to make relative time measurements with a 16 millisecond resolution.

Both the second and the tick count field are initialized to zero at boot time and it is not possible to adjust them later (the counters are intended for relative time measurements only).

Registers on entry:

AL	19

Registers on return:

DX	Second count high
AX	Second count low
CX	Elapsed 16 mS periods of next second.

Function 20

Defines a character in the alternative character set.

Registers on entry:

AL	20
CL	Character number (0-255)
DS	Address segment of character definition block
DX	Address offset of character definition block

Registers on return:

Undefined

See 4.3.2.

Function 21

Returns a pointer to a console display list.

Registers on entry:

AL	21

Registers on return:

ES	Address segment display list table
BX	Address offset display list table
DX	Display buffer segment
SI	Intel 82730 command block

See 4.2.3.

Function 22

Returns the current cursor position.

Registers on entry:

AL	22

Registers on return:

BH	Row
BL	Column

See 4.2.4.

Function 23

Returns status of iSBX351 controller (if installed).

Registers on entry:

AL	23

Registers on return:

AX	Status

See 10.3

Function 24

Initializes the iSBX351 controller (if installed).

Registers on entry:

AL	24

Stack on entry:

+2	Parameter block segment
+0	Parameter block offset

Registers on return:

Undefined

See 10.3

Function 25

Reserved.

Function 26

Read file header record from cassette tape.

Registers on entry:

AL	26
CX	Max number of bytes to read
DX	Input buffer offset

Stack on entry:

+0	Input buffer segment

Registers on return:

Al	Function result = 0 ok = 1 CRC error = 2 no data on tape = 3 no leader found = 4 wrong record number = 5 end of file
AH, BX	Undefined
CX	Number of bytes read
DX	Offset of next byte in input buffer

Stack on return:

Unchanged

See 7.2.5

Function 27

Write file header on cassette tape.

Registers on entry:

AL	27
CX	Number of bytes to write
DX	Output buffer offset

Stack on entry:

+0	Output buffer segment

Registers on return:

AX, CX	0
BX	Undefined
DX	Offset of next byte in output buffer

Stack on return:

Unchanged

See 7.2.2

Function 28

Read next data record from cassette tape.

Registers on entry:

AL	28
CX	Max number of bytes to read
DX	Input buffer offset

Stack on entry:

+0	Input buffer segment

Registers on return:

```
Al          Function result
            = 0  ok
            = 1  CRC error
            = 2  no data on tape
            = 3  no leader found
            = 4  wrong record number
            = 5  end of file

AH, BX      Undefined
CX          Number of bytes read
DX          Offset of next byte in input buffer
```

Stack on return:

Unchanged

See 7.2.6

Function 29

Write next data record on cassette tape.

Registers on entry:

AL	29
CX	Number of bytes to write
DX	Output buffer offset

Stack on entry:

+0	Output buffer segment

Registers on return:

AX, CX	0
BX	Undefined
DX	Offset of next byte in output buffer

Stack on return:

Unchanged

See 7.2.3

Function 30

Subfunction 1:

Initializes mouse.

Registers on entry:

AL	30
CL	1

Registers on return:

Undefined

Subfunction 2:

Deinitializes mouse.

Registers on entry:

AL	30
CL	2

Registers on return:

Undefined

Subfunction 3:

Returns the current status of the mouse device.

Registers on entry:

AL	30
CL	3

Registers on return:

a) Nothing happened

AL	0

b) Button pressed

AL	1
AH	Character information.

c) Mouse moved

AL	2
BX	Delta x
CX	Delta y

See 4.7.

Function 31

Defines palette contents.

Registers on entry:

AL	31
DS	Address segment of palette definition
DX	Address offset of palette definition

Registers on return:

Undefined

See 4.1.3.

Function 32-34

Reserved

Function 35

Write a string direct to the console buffer.

Registers on entry:

AL	35
DL	Column
DH	Row
CX	Count
DS	Address segment of string
SI	Address offset of string

Registers on return:

Undefined

See 4.2.2.

Function 36

Set cursor position.

Registers on entry:

BH	Row
BL	Column

Registers on return:

Undefined

See 4.2.4.

Function 37

Returns current attributes.

Registers on entry:

AL	37

Registers on return:

AH	Current attributes

See 4.2.5.

Function 38

Set attributes.

Registers on entry:

AL	38
AH	Attributes

Registers on return:

Undefined

See 4.2.5.

Function 39

Update physical screen.

Registers on entry:

AL	39

Registers on return:

Undefined

See 4.2.3

Function 40

Write an end of file record on cassette tape.

Registers on entry:

AL	40

Registers on return:

AX	0

Function 41

Subfunction 1:

Reserve DPC parallel interface.

Registers on entry:

AL	41
AH	1

Registers on return:

Undefined

See 9.3.1

Subfunction 2:

Release DPC parallel interface.

Registers on entry:

AL	41
AH	2

Registers on return:

Undefined

See 9.3.2

Function 42

Subfunction 1:

Reserve shared disk.

Registers on entry:

AL	42
AH	1

Registers on return:

Undefined

See 8.4.1

Subfunction 2:

Release shared disk.

Registers on entry:

AL	42
AH	2

Registers on return:

Undefined

See 8.4.2

Function 43-49

Reserved

Function 50

Reset iSBX351.

Registers on entry:

AL	50

Registers on return:

Undefined

see 10.3

Function 51

Get font.

Returns a character from the character set.

Registers on entry:

AL	51
CX	Character number (0-1023)
DS	Address segment of character definition block
DX	Address offset of character definition block

Registers on return:

Undefined

See 4.3.4.

Function 52

Define font.

Defines a character in the character set.

Registers on entry:

AL	52
CX	Character number (0-1023)
DS	Address segment of character definition block
DX	Address offset of character definition block

Registers on return:

Undefined

See 4.3.3.

Function 53

Get XIOS version

Registers on entry:

AL	53

Registers on return:

AH	Year (BCD)
AL	Version number (BCD)
BH	Month (BCD)
BL	Day (BCD)

B. Peripheral Device I/O Addresses

Address	Peripheral	Direction	Interrupt	DMA request
0000H	I8259 Int. Crt.			
0020H	Keyboard	I	1	
0056H-005CH	Sound	O		
005CH	RTC		3	
0060H	CRT Control	O	4	3
0070H	PPI Port A	I		
0072H	PPI Port B	I		
0074H	PPI Port C	O		
0076H	Control 70H-74H	O		
0080H-00FEH	NVM	I/O		
0100H	Net Ch. Attent			
0180H-01BEH	Palette	O		
0230H	Reset Int. CRT			
0240H	Ch. Attent.			
0250H	Local Prin. Data	I/O	6	
0260H	Local Prin. Contr.			
0280H	Floppy Control	I/O	0	0
0282H	Track Reg.	I/O		
0284H	Sector Reg.	I/O		
0286H	Data Reg.	I/O		
0288H	Floppy Control	O		
028AH	DPC Prin. Data	I/O		
028CH	DPC Prin. Contr.	I/O	2	
028EH	Test Floppy	I		
028EH	Reserve Floppy	O		
0290H	Release Floppy	O		
0292H	Test Printer	I		
0292H	Reserve Printer	O		
0294H	Release Printer	O		
0300H-030EH	iSBX	I/O	INT1	0
0310H-031EH	iSBX	I/O	INT3	0
0320H	DMA ACK to iSBX	O		
0330H	TC to iSBX	O		

C. Interrupt Vector Assignment

Cause	Type	Vector
Divide error exception	Internal 0	0000:0000H
Single step interrupt	Internal 1	0000:0004H
Non maskable interrupt	Internal 2	0000:0008H
Breakpoint interrupt	Internal 3	0000:000CH
INT 0 detected	Internal 4	0000:0010H
Array bounds exception	Internal 5	0000:0014H
Unused opcode exception	Internal 6	0000:0018H
ESC opcode exception	Internal 7	0000:001CH
Timer 0 interrupt	Internal 8	0000:0020H
Timer 1 interrupt	Internal 18	0000:0048H
Timer 2 interrupt	Internal 19	0000:004CH
DMA 0 interrupt	Internal 10	0000:0028H
DMA 1 interrupt	Internal 11	0000:002CH
INT 0 interrupt	Internal 12	0000:0030H
INT 1 interrupt	Internal 13	0000:0034H
INT 2 interrupt	Internal 14	0000:0038H
INT 3 interrupt	Internal 15	0000:003CH
Floppy controller	External 0	0000:0080H
Keyboard interface	External 1	0000:0084H
DPC interface	External 2	0000:0088H
Real time clock	External 3	0000:008CH
CRT	External 4	0000:0090H
Net controller	External 5	0000:0094H
Parallel interface	External 6	0000:0098H
Not used	External 7	0000:009CH
Int-28h functions	Int 28H	0000:00A0H
Net driver	Int 29H	0000:00A4H
IMC	Int 30H	0000:00A8H

D. Character Set and Keystrokes

VALUE			
DEC	HEX	KEYSTROKES	
0	00	CTRL @	
1	01	CTRL A	É
2	02	CTRL B	Ä
3	03	CTRL C	ö
4	04	CTRL D	é
5	05	CTRL E	ä
6	06	CTRL F	ö
7	07	CTRL G	
8	08	⬅	ß
9	09	⇥	£
10	0A	CTRL J	
11	0B	CTRL K	Æ
12	0C	⇤	Ø
13	0D	↵	
14	0E	CTRL N	Å
15	0F	CTRL O	Ü

VALUE			
DEC	HEX	KEYSTROKES	
16	10	CTRL P	§
17	11	CTRL Q	#
18	12	CTRL R	[
19	13	CTRL S	\
20	14	CTRL T	]
21	15	CTRL U	^
22	16	CTRL V	`
23	17	CTRL W	{
24	18	CTRL X	¦
25	19	CTRL Y	}
26	1A	CTRL Z	~
27	1B	ESC	@
28	1C	CTRL Ø	æ
29	1D	CTRL Å	ø
30	1E	CTRL Ü	å
31	1F	CTRL _	ü

VALUE			
DEC	HEX	KEYSTROKES	
32	20	space bar	
33	21	SHIFT !	!
34	22	SHIFT "	"
35	23	SHIFT §	§
36	24	SHIFT $	$
37	25	SHIFT %	%
38	26	SHIFT &	&
39	27	SHIFT '	'
40	28	SHIFT (	(
41	29	SHIFT)	)
42	2A	SHIFT *	*
43	2B	SHIFT +	+
44	2C	,	,
45	2D	-	-
46	2E	.	.
47	2F	/	/

VALUE			
DEC	HEX	KEYSTROKES	
48	30	0	0
49	31	1	1
50	32	2	2
51	33	3	3
52	34	4	4
53	35	5	5
54	36	6	6
55	37	7	7
56	38	8	8
57	39	9	9
58	3A	:	:
59	3B	;	;
60	3C	SHIFT <	<
61	3D	SHIFT =	=
62	3E	SHIFT >	>
63	3F	SHIFT ?	?

VALUE			
DEC	HEX	KEYSTROKES	
64	40	SHIFT @	@
65	41	SHIFT A	A
66	42	SHIFT B	B
67	43	SHIFT C	C
68	44	SHIFT D	D
69	45	SHIFT E	E
70	46	SHIFT F	F
71	47	SHIFT G	G
72	48	SHIFT H	H
73	49	SHIFT I	I
74	4A	SHIFT J	J
75	4B	SHIFT K	K
76	4C	SHIFT L	L
77	4D	SHIFT M	M
78	4E	SHIFT N	N
79	4F	SHIFT O	O

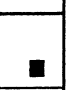
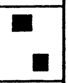
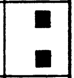
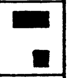
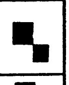
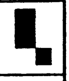
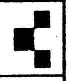
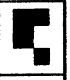
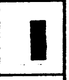
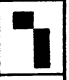
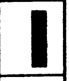
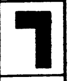
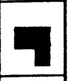
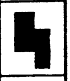
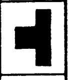
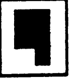
VALUE			
DEC	HEX	KEYSTROKES	
80	50	SHIFT P	P
81	51	SHIFT Q	Q
82	52	SHIFT R	R
83	53	SHIFT S	S
84	54	SHIFT T	T
85	55	SHIFT U	U
86	56	SHIFT V	V
87	57	SHIFT W	W
88	58	SHIFT X	X
89	59	SHIFT Y	Y
90	5A	SHIFT Z	Z
91	5B	SHIFT Æ	Æ
92	5C	SHIFT Ø	Ø
93	5D	SHIFT Å	Å
94	5E	SHIFT Ü	Ü
95	5F	SHIFT _	_

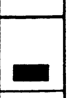
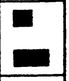
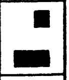
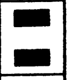
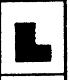
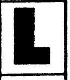
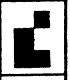
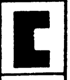
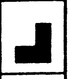
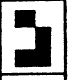
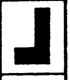
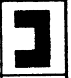
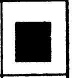
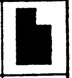
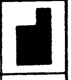
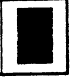
VALUE			
DEC	HEX	KEYSTROKES	
96	60	`	`
97	61	A	a
98	62	B	b
99	63	C	c
100	64	D	d
101	65	E	e
102	66	F	f
103	67	G	g
104	68	H	h
105	69	I	i
106	6A	J	j
107	6B	K	k
108	6C	L	l
109	6D	M	m
110	6E	N	n
111	6F	O	o
112	70	P	p
113	71	Q	q
114	72	R	r
115	73	S	s
116	74	T	t
117	75	U	u
118	76	V	v
119	77	W	w
120	78	X	x
121	79	Y	y
122	7A	Z	z
123	7B	Æ	æ
124	7C	Ø	ø
125	7D	Å	å
126	7E	Ü	ü
127	7F	CTRL ←	

VALUE			
DEC	HEX	KEYSTROKES	
128	80	CTRL ALT @	┏
129	81	CTRL ALT A	┓
130	82	CTRL ALT B	┗
131	83	CTRL ALT C	┛
132	84	CTRL ALT D	┳
133	85	CTRL ALT E	┫
134	86	CTRL ALT F	┣
135	87	CTRL ALT G	┻
136	88	CTRL ALT H	━
137	89	CTRL ALT I	┃
138	8A	CTRL ALT J	╋
139	8B	CTRL ALT K	╭
140	8C	CTRL ALT L	╮
141	8D	CTRL ALT M	╰
142	8E	CTRL ALT N	╯
143	8F	CTRL ALT O	▪

VALUE			
DEC	HEX	KEYSTROKES	
144	90	CTRL ALT P	←
145	91	CTRL ALT Q	→
146	92	CTRL ALT R	↑
147	93	CTRL ALT S	↓
148	94	CTRL ALT T	↖
149	95	CTRL ALT U	±
150	96	CTRL ALT V	≠
151	97	CTRL ALT W	≥
152	98	CTRL ALT X	≤
153	99	CTRL ALT Y	≈
154	9A	CTRL ALT Z	√
155	9B	CTRL ALT Æ	∞
156	9C	CTRL ALT Ø	⌠
157	9D	CTRL ALT Å	⌡
158	9E	CTRL ALT Ü	²
159	9F	CTRL ALT _	³

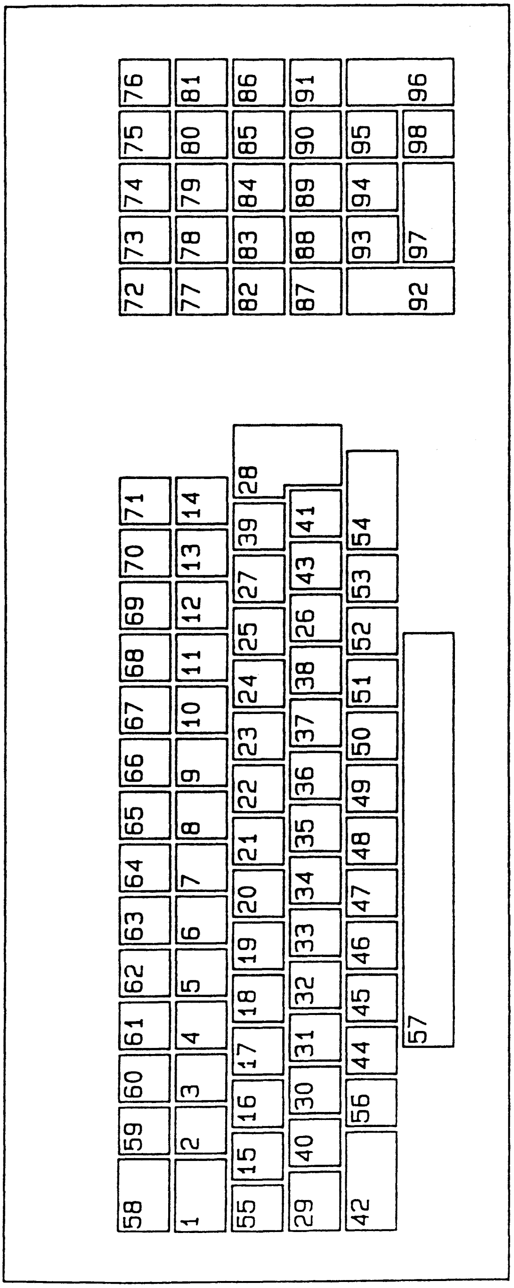
VALUE				VALUE			
DEC	HEX	KEYSTROKES		DEC	HEX	KEYSTROKES	
160	A0	ALT SPACE		176	B0	ALT 0	🬏
161	A1	SHIFT ALT !	🬀	177	B1	ALT 1	🬐
162	A2	SHIFT ALT "	🬁	178	B2	ALT 2	🬑
163	A3	SHIFT ALT @	🬂	179	B3	ALT 3	🬒
164	A4	SHIFT ALT $	🬃	180	B4	ALT 4	🬓
165	A5	SHIFT ALT %	🬄	181	B5	ALT 5	▌
166	A6	SHIFT ALT &	🬅	182	B6	ALT 6	🬔
167	A7	SHIFT ALT '	🬆	183	B7	ALT 7	🬕
168	A8	SHIFT ALT (	🬇	184	B8	ALT 8	🬖
169	A9	SHIFT ALT)	🬈	185	B9	ALT 9	🬗
170	AA	SHIFT ALT *	🬉	186	BA	ALT :	🬘
171	AB	SHIFT ALT +	🬊	187	BB	ALT ;	🬙
172	AC	ALT ,	🬋	188	BC	SHIFT ALT <	🬚
173	AD	ALT -	🬌	189	BD	SHIFT ALT =	🬛
174	AE	ALT .	🬍	190	BE	SHIFT ALT >	🬜
175	AF	ALT /	🬎	191	BF	SHIFT ALT ?	🬝

VALUE			
DEC	HEX	KEYSTROKES	
192	C0	SHIFT ALT @	
193	C1	SHIFT ALT A	
194	C2	SHIFT ALT B	
195	C3	SHIFT ALT C	
196	C4	SHIFT ALT D	
197	C5	SHIFT ALT E	
198	C6	SHIFT ALT F	
199	C7	SHIFT ALT G	
200	C8	SHIFT ALT H	
201	C9	SHIFT ALT I	
202	CA	SHIFT ALT J	
203	CB	SHIFT ALT K	
204	CC	SHIFT ALT L	
205	CD	SHIFT ALT M	
206	CE	SHIFT ALT N	
207	CF	SHIFT ALT O	

VALUE			
DEC	HEX	KEYSTROKES	
208	D0	SHIFT ALT P	
209	D1	SHIFT ALT Q	
210	D2	SHIFT ALT R	
211	D3	SHIFT ALT S	
212	D4	SHIFT ALT T	
213	D5	SHIFT ALT U	
214	D6	SHIFT ALT V	
215	D7	SHIFT ALT W	
216	D8	SHIFT ALT X	
217	D9	SHIFT ALT Y	
218	DA	SHIFT ALT Z	
219	DB	SHIFT ALT Æ	
220	DC	SHIFT ALT Ø	
221	DD	SHIFT ALT Å	
222	DE	SHIFT ALT Ü	
223	DF	SHIFT ALT _	

VALUE				VALUE			
DEC	HEX	KEYSTROKES		DEC	HEX	KEYSTROKES	
224	E0	ALT `	α	240	F0	ALT P	ρ
225	E1	ALT A	β	241	F1	ALT Q	σ
226	E2	ALT B	γ	242	F2	ALT R	τ
227	E3	ALT C	δ	243	F3	ALT S	υ
228	E4	ALT D	ε	244	F4	ALT T	φ
229	E5	ALT E	ζ	245	F5	ALT U	χ
230	E6	ALT F	η	246	F6	ALT V	ψ
231	E7	ALT G	θ	247	F7	ALT W	ω
232	E8	ALT H	ι	248	F8	ALT X	Δ
233	E9	ALT I	κ	249	F9	ALT Y	Γ
234	EA	ALT J	λ	250	FA	ALT Z	Σ
235	EB	ALT K	μ	251	FB	ALT Æ	Λ
236	EC	ALT L	ν	252	FC	ALT Ø	Ω
237	ED	ALT M	ξ	253	FD	ALT Å	#
238	EE	ALT N	ο	254	FE	ALT Ü	¨
239	EF	ALT O	π	255	FF	impossible	RC

E. Keyboard Position Codes



F. Console Escape Sequences

Sequence	Function
ESC A	Cursor Up
ESC B	Cursor Down
ESC C	Cursor Forward
ESC D	Cursor Backward
ESC E	Clear Screen, Cursor Home
ESC H	Cursor Home
ESC I	Reverse Index
ESC J	Erase to End of Screen
ESC K	Erase to end of line
ESC L	Insert Line
ESC M	Delete Line
ESC N	Delete Character
ESC O	Insert Character
ESC P	Select Alternative Character Set
ESC Q	Select Standard Character Set
ESC Y x x	Position Cursor
ESC a	Ignored
ESC b x	Set Foreground Colour
ESC c x	Set Background Colour
ESC d	Erase Beginning of Screen
ESC e	Enable Cursor
ESC f	Disable Cursor
ESC g	Enter Underline Mode
ESC h	Exit Underline Mode
ESC i	Enter Non-Displayed Mode
ESC j	Save Cursor Position
ESC k	Restore Cursor Position
ESC l	Erase Line
ESC m	Enable Cursor
ESC n	Disable Cursor
ESC o	Erase Beginning of Line
ESC p	Enter Reverse Video Mode
ESC q	Exit Reverse Video Mode
ESC r	Enter Intensify Mode
ESC s	Enter Blink Mode
ESC t	Exit Blink Mode
ESC u	Exit Intensify Mode
ESC v	Wrap at End of Line
ESC w	Discard at End of Line
ESC x	Exit Non-Displayed Mode
ESC z	Reset Attributes

Sequence	Function
ESC 0	Status Line Off (25 Line Mode)
ESC 1	Status Line On (24 Line Mode)
ESC 2	Save Current Attributes
ESC 3	Restore Attributes
ESC 6	Function Key Expansion Off
ESC 7	Function Key Expansion On
ESC : x c...c NUL	Program Function Keys
ESC < x x	Scroll Window Up
ESC > x x	Scroll Window Down
ESC <241>	Set Blinking Cursor
ESC <242>	Set Non-Blinking Cursor
ESC <243> x	Set Cursor Representation
ESC <244>	Set Soft Scroll
ESC <245>	Set Line Scroll
ESC <246>	Disable Underline Attribute
ESC <247>	Enable Underline Attibute
ESC <253>	Save Function Keys
ESC <254>	Restore Function Keys

G. References

1. Concurrent CP/M-86, User's Guide.
   Digital Research

2. Concurrent CP/M-86, Programmer's Reference Guide.
   Digital Research

3. Concurrent CP/M-86, System Guide.
   Digital Research

4. DrNet, Network operating system, System guide.
   Digital Research

5. PICCOLINE brugervejledning.
   SW1401D
   RC COMPUTER

6. Local Area Networks - Logical Link Control - Draft E
   ISO/DP 8802/2 (TC 97/SC 6 N2925)

7. Data Processing - Open Systems Interconnection
   Basic Reference Model
   Feb. 4, 1982
   ISO/DIS 7498

8. Distributed System Architecture, Report
   RCSL No. 42-i1982
   RC Computer

9. DSA Inter Module Communication,
   Functional Description
   RCSL No. 42-i1983
   RC Computer

10. Intel 82586 Reference Manual
    order number 210891-001
    Intel Corporation, 1983

11. RC 759 Techical Hardwaredocumentation
    SW1493D
    RC Computer, 1985

Catchword Index:

RETURN LETTER

Title: PICCOLINE Programmer's Guide Version 2.0 **RCSL No.:** 99 0 00864

A/S Regnecentralen af 1979/RC Computer A/S maintains a continual effort to improve the quality and usefulness of its publications. To do this effectively we need user feedback, your critical evaluation of this manual.

Please comment on this manual's completeness, accuracy, organization, usability, and readability:

____

____

____

____

Do you find errors in this manual? If so, specify by page.

____

____

____

____

How can this manual be improved?

____

____

____

____

Other comments?

____

____

____

____

____

Name:____________________ Title:____________________

Company:____________________________________

Address:____________________________________

Date:________

Thank you

PN: 99200176

.... Fold here ....

.... Do not tear - Fold here and staple ....

Affix
postage
here

REGNECENTRALEN
af 1979

Information Department
Lautrupbjerg 1
DK-2750 Ballerup
Denmark