Partner

Programmer's

Guide

Partner

Programmer's Guide

Partner Information	Category System Information	Product SW1695	
Ident PN 798 50 178	Replaces		Page 1/1
Subject Programmer's toolkit, release 2.0 - package description			

1. Package Name

SW1695 Programmer's toolkit, release 2.0

2. Package contents

1 Disk labelled SW1695 release 2.0
1 Partner programmer's guide, version 4.0
1 Concurrent DOS 86, Programmer's Guide, Digital Research
1 Concurrent DOS 86, System Guide, Digital Research
1 Programmer's utilities guide, Digital Research

Programmer's toolkit release 2.0 covers Partner Concurrent DOS release 5.0.

3. Disk Contents

The disk contains the following files:

ASM86.CMD	8086 Assembler
GENCMD.CMD	Generates .CMD-file from ASM86 output
DDT86.CMD	8086 Dynamic Debug Tool
SYSTAT.CMD	Displays current system status

NOTE: The disk is identical to the disk in SW1695 rel 1.0.

DIGITAL RESEARCH
LANGUAGES END USER LICENSE AGREEMENT

Use and possession of this software package is governed by the following terms.

1. DEFINITIONS - These definitions shall govern:

A. "DRI" means DIGITAL RESEARCH INC., P.O. Box 579, Pacific Grove, California 93950, the author and owner of the copyright on this SOFTWARE.

B. "CUSTOMER" means the individual purchaser and the company CUSTOMER works for, if the company paid for this SOFTWARE.

C. "COMPUTER" is the single microcomputer on which CUSTOMER uses this program. Multiple CPU systems may require supplementary licenses.

D. "SOFTWARE" is the set of computer programs in this package, regardless of the form in which CUSTOMER may subsequently use it, and regardless of any modification which CUSTOMER may make to it.

E. "LICENSE" means this Agreement and the rights and obligations which it creates under the United States Copyright Law and California laws.

F. "RUNTIME LIBRARY" is the set of copyrighted DRI language subroutines, provided with each language compiler, a portion of which must be linked to and become part of a Customer program for that program to run on the COMPUTER.

2. LICENSE

DRI grants CUSTOMER the right to use this serialized copy of the SOFTWARE on a single COMPUTER at a single location so long as CUSTOMER complies with the terms of the LICENSE, and either destroys or returns the SOFTWARE when CUSTOMER no longer has this right. CUSTOMER may not transfer the program electronically from one computer to another over a network. DRI shall have the right to terminate this license if CUSTOMER violates any of its provisions. CUSTOMER owns the diskette(s) purchased, but under the Copyright Law DRI continues to own the SOFTWARE recorded on it and all copies of it. CUSTOMER agrees to make no more than five (5) copies of the SOFTWARE for backup purposes and to place a label on the outside of each backup diskette showing the serial number, program name, version number and the DRI copyright and trademark notices in the same form as the original copy. CUSTOMER agrees to pay for licenses for additional user copies of the SOFTWARE if CUSTOMER intends to or does use it on more than one COMPUTER. If the microcomputer on which CUSTOMER uses the SOFTWARE is a multi-user microcomputer system, then the license covers all users on that single system, without further license payments, only if the SOFTWARE is used only on that microcomputer. This is NOT a license to use the SOFTWARE on mainframes or emulators.

3. TRANSFER OR REPRODUCTION

CUSTOMER understands that unauthorized reproduction of copies of the SOFTWARE and/or unauthorized transfer of any copy may be a serious crime, as well as subjecting CUSTOMER to damages and attorney fees. CUSTOMER may not transfer any copy of the SOFTWARE to another person unless CUSTOMER transfers all copies, including the original, and advises DRI of the name and address of that person, who must sign a copy of the registration card, pay the then current transfer fee, and agree to the terms of this LICENSE in order to use the SOFTWARE. DRI will provide additional copies of the card and LICENSE upon request. DRI has the right to terminate the LICENSE, to trace serial numbers, and to take legal action if these conditions are violated.

4. COMPOSITE PROGRAMS

As an exception to Paragraph 3, CUSTOMER is granted the right to include portions of the DRI RUNTIME LIBRARY in CUSTOMER developed programs, called COMPOSITE PROGRAMS, and to use, distribute and license such COMPOSITE PROGRAMS to third parties without payment of any further license fee. CUSTOMER shall, however, include in such COMPOSITE PROGRAM, and on the exterior label of every diskette, a copyright notice in this form: "Portions of this program, ©1982 DIGITAL RESEARCH INC." In cases where such COMPOSITE PROGRAM is contained in READ-ONLY-MEMORY (ROM) chips, a copyright notice in the form listed above, must be displayed on the exterior of the chip and internally in the chip (in ASCII literal form). As an express condition to the use of the RUNTIME LIBRARY, CUSTOMER agrees to indemnify and hold DRI harmless from all claims by CUSTOMER and third parties arising out of the use of COMPOSITE PROGRAMS.

5. LIMITED WARRANTY

The only warranty DRI makes is that the diskette(s) on which the SOFTWARE is recorded will be replaced without charge, if DRI in good faith determines that the media was defective and not subject to misuse, and if returned to DRI or the dealer from whom it was purchased, with a copy of the original registration card, within ten days of purchase. Customer will receive support from the Vendor from whom customer has purchased the software. In addition, support is available from DRI directly, for qualified, registered customers under DRI's then current support policies. DRI reserves the right to change the specifications and operating characteristics of the SOFTWARE it produces, over a period of time, without notice.

6. DRI MAKES NO OTHER WARRANTIES, EITHER EXPRESSED OR IMPLIED, AND DRI SHALL NOT BE LIABLE FOR WARRANTIES OF FITNESS OF PURPOSE OR MERCHANTABILITY, NOR FOR INDIRECT, SPECIAL OR CONSEQUENTIAL DAMAGES SUCH AS LOSS OF PROFITS OR INABILITY TO USE THE SOFTWARE. SOME STATES MAY NOT ALLOW THIS DISCLAIMER SO THIS LANGUAGE MAY NOT APPLY TO CUSTOMER. IN SUCH CASE, OUR LIABILITY SHALL BE LIMITED TO REFUND OF THE DRI LIST PRICE. CUSTOMER MAY HAVE OTHER RIGHTS WHICH VARY FROM STATE TO STATE. CUSTOMER and DRI agree that this product is not intended as "Consumer Goods" under state or federal warranty laws.

7. MISCELLANEOUS

This is the only agreement between CUSTOMER and DRI and it cannot and shall not be modified by purchase orders, advertising or other representations of anyone, unless a written amendment has been signed by one of our company officers. When CUSTOMER opens the SOFTWARE package or uses the SOFTWARE, this act shall be considered as mutual agreement to the terms of this LICENSE. This LICENSE shall be governed by California law, except as to copyright matters which are covered by Federal laws, and is deemed entered into at Pacific Grove, Monterey County, CA by both parties.

DIGITAL RESEARCH™

by Gary A. Kildall
President

SAVE THIS LICENSE FOR FUTURE REFERENCE

Form E-2 (10/22/82)

Partner Mikrodatamatsystem

Partner Programmer's Guide

Version 3.0

Regnecentralen a/s PN: 991-10560

Partner, operating system, Concurrent DOS, software, hardware.

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

Throughout this manual it is assumed that the reader is familiar with the ASM86 or the RASM86 assembler, with the Concurrent DOS operating system and with peripheral device interfacing (interrupts e.t.c.).

The manual is intended for use in connection with SW1500 release 4.0.

Changes from version 2.0 of this manual includes:

- Re-allocation of interrupt vectors
- Descriptions of MF140, MF141, MF142, MF143 and MF144 adapters
- Descriptions of new int-28h functions
- Programmer relevant details about the DOS emulation.

June 1986.

Table of Contents

1. Introduction 7
   1.1 XIOS Overview 9

2. CPU 17
   2.1 Interrupt System 18
   2.2 Direct Memory Access 19
      2.2.1 DMA Channel Reservation 19
      2.2.2 DMA Request Line Setup 20
      2.2.3 DMA Channel Priority 21
      2.2.4 DMA Interrupt Handling 21
   2.3 Timers 22

3. Configuration 23
   3.1 Auto Configuration 23
   3.2 Non Volatile Memory 25

4. Console Module 33
   4.1 CRT controller 33
      4.1.1 82730 Command Block 33
      4.1.2 Character Format 34
      4.1.3 Palette 35
      4.1.4 Graphics Mode 38
   4.2 Direct Console access 38
      4.2.1 XIOS conout 39
      4.2.2 Direct console buffer output 40
      4.2.3 Display Buffer Manipulation 41
      4.2.4 Get/Set Cursor Position 42
      4.2.5 Get/Set Attribute 43
   4.3 Character Sets 43
      4.3.1 Altering the Character Set 44
      4.3.2 Define Character Font (Alternative) 45
      4.3.3 Define Character Font 46
      4.3.4 Get Character Font Definition 46
   4.4 Console control characters 48
      4.4.1 Console Escape Sequences 48
   4.5 Graphics Mode 62
      4.5.1 Init Graphics 63
      4.5.2 Exit Graphics 64
      4.5.3 Exclusive Access to Pixel Memory 64
      4.5.4 Pixel Address Calculation 67
   4.6 Window Handling 68
      4.6.1 Return Pointers 69
      4.6.2 Set Window Manager State 70
      4.6.3 Create a New Window 70
      4.6.4 Set Cursor Tracking Mode and Viewpoint 70

4.6.5 Set Wrap Around Column 71
4.6.6 Switch Between Full Screen and Window 71
4.7 Keyboard Interface 71
4.7.1 Keyboard driver 71
4.8 PC-mode 72
4.8.1 PC-mode keyboard 72
4.8.2 8-bit PC-mode 73
4.9 Mouse Interface 73

5. Real time clock 75
5.1 Real time clock controller 75
5.2 Reading and writing real time clock registers ... 75

6. Sound 79
6.1 Programming tones 80
6.2 Programming noise 83

7. Serial interface 87
7.1 Standard serial communication support 87
7.1.1 V24 Handshake scheme 88
7.2 Serial communication controller 89
7.2.1 Asynchronous communication using ch. A. ... 90
7.2.2 Synchronous communication using ch. A 92
7.2.3 Asynchronous communication using ch. B 93
7.3 Initializing the INTEL 8274 95
7.4 Sample asynchronous communication program 99

8. Disk system 107
8.1 Floppy disk characteristics 107
8.2 Floppy disk controller 111
8.3 Floppy disk driver 113
8.4 SCSI interface 113
8.5 SCSI interface driver 114

9. Parallel (printer) interface 119
9.1 Parallel interface description 119
9.2 Sample printer driver routines 122

10. Local Area Network 127
10.1 Fundamental concepts 128
10.2 The datalink layer service interface 129
10.2.1 RCLLC Services 133
10.2.1.1 ACTIVATE.request 133
10.2.1.2 ACTIVATE.confirm 134
10.2.1.3 DEACTIVATE.request 135
10.2.1.4 DEACTIVATE.confirm 136
10.2.1.5 TEST.request 137
10.2.1.6 TEST.confirm 138
10.2.1.7 TEST.indication 138

10.2.2 Type 1 Service 139
10.2.2.1 UDATA.request 139
10.2.2.2 UDATA.confirm 140
10.2.2.3 UDATA.indication 141
10.2.3 Client Network Service 141
10.2.3.1 CONNECT.indication 142
10.2.3.2 DISCONNECT.indication 143
10.2.3.3 DISCONNECT.acknowledge 144
10.2.3.4 DISCONNECT_ACKNOWLEDGE.confirm 145
10.2.3.5 DATA.request 145
10.2.3.6 DATA.confirm 146
10.2.3.7 DATA.indication 147
10.3 MAC Services 148
10.3.1 Controller specific information 148
10.4 RCLLC datalink layer protocol 150
10.4.1 Type 1 Procedures 150
10.4.1.1 Unacknowledged Data Transfer . 151
10.4.1.2 Loop-back Test Procedure 151
10.4.1.3 Station Identification Exchange 152
10.4.2 Procedures for Client Network Service . 152
10.4.3 RCLLC protocol elements 157
10.4.3.1 UI (Unnumbered Information) .. 160
10.4.3.2 XID (eXchange IDentification) 160
10.4.3.3 TEST 160
10.4.3.4 ACTIVE_SAP 160
10.4.3.5 RESET 161
10.4.3.6 RACK 162
10.4.3.7 DATA 162
10.4.3.8 ACK 162

11. Adapters 163
11.1 MF140 Dual Satellite Adapter 163
11.1.1 MF140 Driver 163
11.1.2 MF140 Hardware Description 164
11.2 MF141 Parallel/Printer Adapter 166
11.2.1 MF141 Driver 166
11.2.2 MF141 Hardware Description 166
11.3 MF142 NRZI Adapter 166
11.3.1 MF142 Driver 167
11.3.2 MF142 Hardware Description 167
11.4 MF143 IEEE488 Adapter 171
11.4.1 MF143 Driver 171
11.4.2 MF143 Hardware Description 171
11.5 MF144 Modem 174
11.5.1 MF144 Driver 174
11.5.2 MF144 Hardware Description 179

12. File Transfer 187
 12.1 Requirements 187
 12.2 How FILEX works 188
  12.2.1 FILEX transactions 188
  12.2.2 Transmission protocol 190

Appendices 193
 A. Int-28h function interface 193
 B. Peripheral device I/O addresses 213
 C. Interrupt vector assignment 215
 D. Character Set and Keystrokes 219
 E. Keyboard Position Codes 237
 F. Console Escape Sequences 239
 G. Adapter Board Dimensions 241
 H. MF140 Schematics 243
 I. MF140 Test Program 251
 J. Adapter Connector 265
 Z. References 267

Index 269

1. Introduction

The intention of this manual is to enable programmers to use the Partner 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 Partner 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 80730

Floppy disk controller based on WD1797

Serial communication controller based on Intel 8274

SCSI bus interface

Keyboard interface

Real time clock

Sound device

Non volatile memory (NVM)

The interconnection of these peripherals is shown on figure 1-1.

Besides these standard peripherals the partner system may be enhanced with a local area network controller based on the Intel 82586 ethernet controller, with an arithmetic co-processor based on an 8 Mhz Intel 8087 and with other controllers connected to the I/O expansion connector.

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

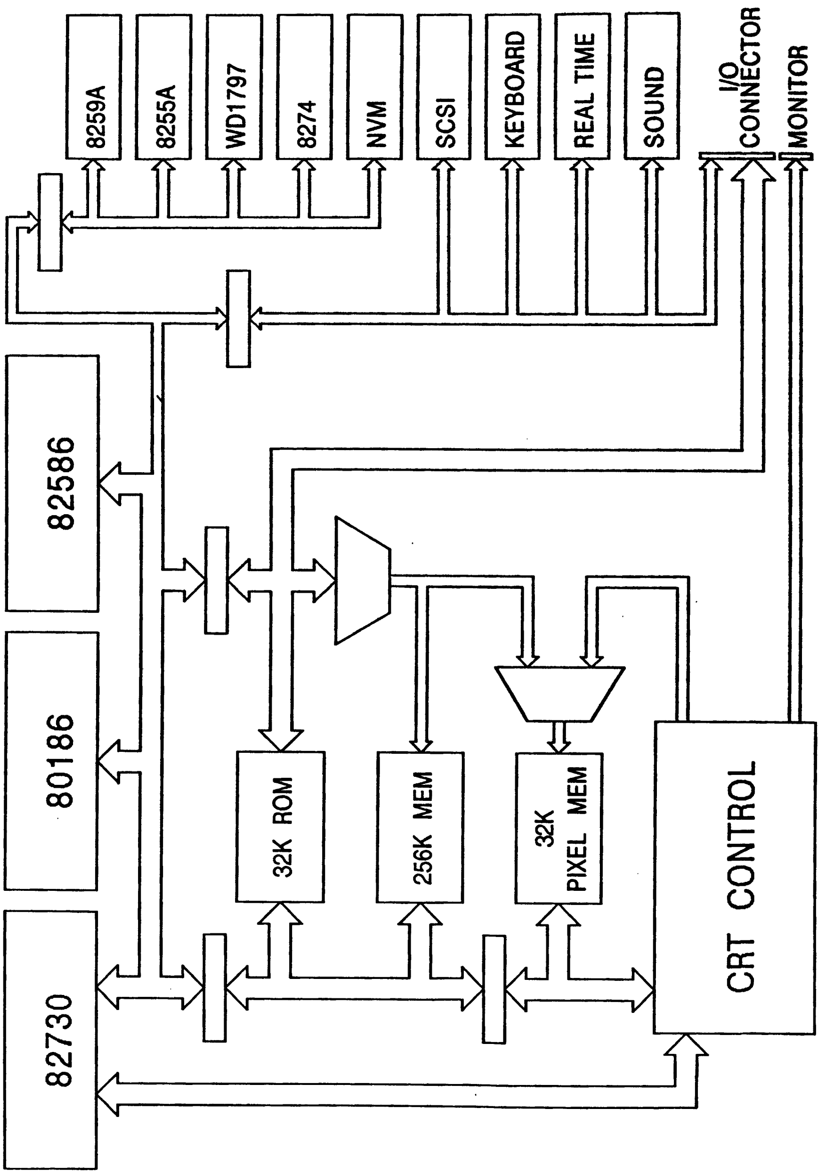


Figure 1-1: Block Diagram.

1.1 XIOS Overview

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

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

An 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.

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

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
PC_VIDEO	31
PC_KBD	32
PC_SHIFTS	33
Not used	34
Not used	35
Not used	36
IO_AUXINST	37
IO_AUXOUTST	38

Table 1-1: XIOS routines.

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 Concurrent DOS operating system call, Concurrent DOS 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 Concurrent DOS flag mechanism (see ref.2 and 3.). Table 1-2 shows how the these flags are assigned on the Partner.

Flag number	**Use**
0	Reserved by Concurrent DOS
1	Tick
2	Second
3	Minut
4	Scroll synchronization
5	Key available flag
6	SCSI
7	Winchester disk
8	Floppy disk
9	Scroll synchronization
10	Scroll synchronization
11	Floppy motor
12	Parallel interface
13	SIO channel A (receive)
14	SIO channel A (transmit)
15	SIO channel B (receive)
16	SIO channel B (transmit)
17	SIO channel A (Xon Xoff)
18	SIO channel B (Xon Xoff)
19	Error key flag
20	SIO channel A (Xon Xoff)
21	SIO channel B (Xon Xoff)
22	Net transmitter
23	Net receiver
24	Window manager
25	MF140 channel A (receive)
26	MF140 channel A (transmit)
27	MF140 channel B (receive)
28	MF140 channel B (transmit)
29	MF140 channel A (Xon-Xoff)
30	MF140 channel B (Xon-Xoff)
31	MF140 channel A (reservation)
32	MF140 channel B (reservation)
33	1.Satellite Statusline error flag
34	2.Satellite Statusline error flag
35	MF141 parallel interface
36-63	Reserved for future use
64-127	**Free**
128-255	Reserved for 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	132	See below
Tmp1	1	132	See below
Tmp2	1	132	See below
Tmp3	1	132	See below
VINQ0	64	2	Virtual Console 0 input
VINQ1	64	2	- - 1 -
VINQ2	64	2	- - 2 -
VINQ3	64	2	- - 3 -
MXalt	1	0	Alt. charset reservation
NTWKQ000	32	4	DR NET
NETSYNC	1	8	DR NET
nios_ind	6	4	NIOS (Net system only)
nios_con	6	4	NIOS (Net system only)
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

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 succeds 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 by means of the 'read console buffer' function (i.e. the same syntax as for command lines is valid, including multible commands separated with the sequence '//'). The format of the queue buffer is:

Byte 0:	Length of Command Line
Byte 1:	Buffer Length (132)
Byte 2-131:	Command Line

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 Partner 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 OFFOOH.

A block diagram of the 80186 is shown below:

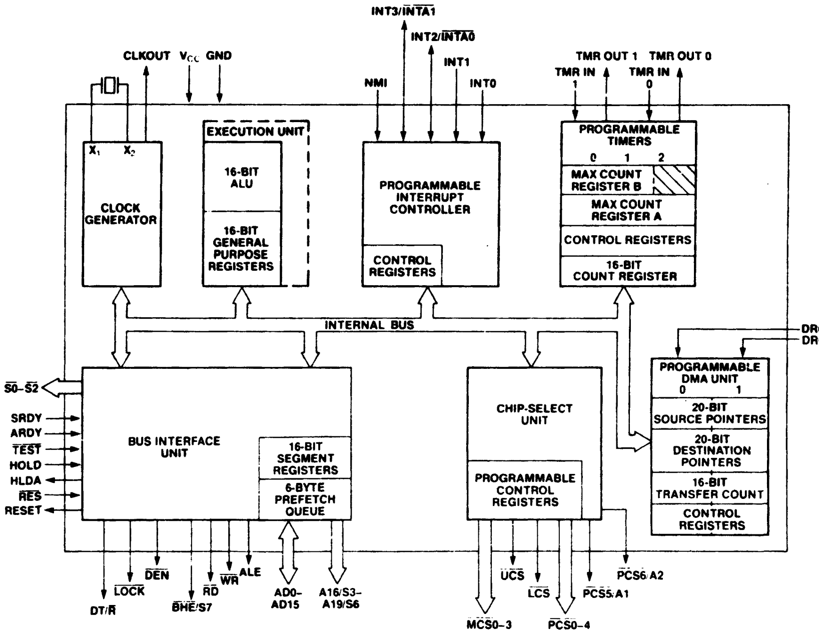


Fig. 2-1: Block diagram of the 80186.

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

2.1 Interrupt system

The peripherals which are able to interrupt the CPU (except the Intel 8274) 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
IR1:	keyboard interface
IR2:	SCSI interface
IR3:	Real Time Clock
IR4:	CRT controller
IR5:	NET controller
IR6:	Parallel (printer) interface
IR7:	I/O expansion connector

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

The Intel 8274 contains its own interrupt control logic and it is connected to the INT1 and INTA1 terminals of the CPU.

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 0:120H:

```
    Mov  DX,0
    Mov  AL,19H         ; level triggered, not single
    Out  DX,AL
    Mov  DX,2
    Mov  AL,48H         ; first vector 0:120H
    Out  DX,AL
    Mov  AL,0
    Out  DX,AL
    Mov  AL,1DH          ; buffer mode, master,
    Out  DX,AL           ; specific EOI, fully nested.
```

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 1 is reserved exclusively for use by user programs while DMA channel 0 is shared amongst user programs and the XIOS (floppy disk and winchester disk driver).

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 amongst 6 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. To avoid bus contention during soft scrolling, DMA channel 0 is implicitly reserved by the process that caused the scrolling. The reservation is done on a per character basis. Due to this, programs that use DMA channel 0 should not use soft scrolling, as this may result in a deadlock situation.

2.2.2 DMA request line setup

Each of the two DMA channels can handle DMA requests from 8 different sources:

Request	Source
0	SCSI bus controller
1	Serial communication controller channel A, transmitter.
2	Serial communication controller channel A, receiver.
3	Not used
4	Not used
5	The floppy disk controller.
6	Reserved for expansion boards.
7	Reserved for expansion boards.

Table 2-1: DMA request sources.

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 70H. The format of the control byte is as follows:

Bit 0-2	DMA request source for DMA channel 1
Bit 3-5	DMA request source for DMA channel 0
Bit 6-7	must have the binary value 11.

Only the bits concerning the reserved channel must be changed.

2.2.3 DMA channel priority

The DMA channels share the access to the system bus with the CPU, the CRT controller and possibly with a local area network controller.

To let the most time critical controllers get the fastest access to the system bus, the controllers are assigned different priorities. There are two possible priority assignments which are controlled in the following way:

Outputting the value 0AH to the I/O address 76H will result in the following assignment (priorities in decreasing order):

DMA channel 0
Local area network controller
CRT controller
DMA channel 1

Outputting the value 0BH to the I/O address 76H will result in the following assignment (priorities in decreasing order):

Local area network controller
CRT controller
DMA channel 1
DMA channel 0

The only difference in the two assignments is in the priority of DMA channel 0.

It is only legal to change the priority assignment when DMA channel 0 has been reserved.

2.2.4 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 the following purposes:

Timer 0 is used for baudrate generation for the Intel 8274 channel B.

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 Partner 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, 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).

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.
14-17	These bytes contain the identification of an attached I/O expansion board. Each byte correspond to a bit in the expansion board identification bit mask. A value of 0ffh correspond to a bit value of 1 and a value of 0 correspond to a bit value of 0.
18	This byte is 03H if a colour monitor is used and 02H if a monochrome monitor is used.
19	This byte hold the number of floppy drives connected to the system.
20	This byte contains a SCSI bit vector. Each bit in the eight bit vector corresponds to a SCSI device address i.e. address 0 to address 7. If a bit is set a controller is attached to the corresponding SCSI address. The 8 SCSI addresses has been allocated to different controllers to enable programs to distinguish between these. The relationship between the controller type and the SCSI address is:

Address	Controller type
0	DTC510B or OMTI20L,C
1	XEBEC S1410 or WD1002-SHD
2	DTC510B or OMTI20L,C
3	XEBEC S1410 or WD1002-SHD
4	DTC510B or OMTI20L,C
5	XEBEC S1410 or WD1002-SHD
6	Reserved
7	Reserved

21	Reserved
22	This byte contains the value of the nationality code switch of the keyboard (range 0-15).

Table 3-1: Configuration Map format.

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 6 and bit 7 in the I/O port at address 70H. 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	Baudrates for the COMM/V24 serial communication channel. High nibble is receive baudrate, low nibble is transmit baudrate. The nibble encoding is: 0: 50 baud 1: 75 - 2: 110 -

3:	150	-
4:	300	-
5:	600	-
6:	1200	-
7:	2400	-
8:	4800	-
9:	9600	-

8	Reserved.
9	Intel 8274 write register 4 content (COMM/V24 channel).
10	Intel 8274 write register 5 content (COMM/V24 channel).
11	Intel 8274 write register 1 content (COMM/V24 channel).
12	Intel 8274 write register 3 content (COMM/V24 channel).
13	Baud rate and mode for the RS232C/V24 communication channel. High nibble is baudrate with the same encoding as in byte 7(receive baudrate=transmit baudrate). Low nibble designates channel usage. 0: virtual console 1: printer
14	Intel 8274 write register 4 contents (RS232C/V24 channel).
15	Intel 8274 write register 5 content (RS232C/V24 channel).
16	Intel 8274 write register 1 content (RS232C/V24 channel).
17	Intel 8274 write register 3 content (RS232C/V24 channel).
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 14 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).
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 a disc drive letter between 'A' and 'D' or the letter 'N' which means load via the local area network.
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	Auto-logon 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	Reserved.
52	System disk number (0-15).
53	Hardware identification (0: Partner)
54	MF140 channel A mode instruction
55	MF140 channel A command instruction
56	MF140 channel A baudrate
57	MF140 channel A konfiguration
58	MF140 channel B mode instruction
59	MF140 channel B command instruction
60	MF140 channel B baudrate
61	MF140 channel B konfiguration
62	MF140 channel A buffer
63	MF140 channel B buffer
64	SIO channel A buffers
65	SIO channel B buffers
66	SIO channel A and B protocol
67	Satellite 1 configuration Bit 0:3 cursor height Bit 4:4 cursor blink Bit 5:5 scroll Bit 6:6 monochrome/colour

68	Satellite 1 colour definition Bit 0:3 foreground colour Bit 4:7 background colour
69	Satellite 2 configuration Bit 0:3 cursor height Bit 4:4 cursor blink Bit 5:5 scroll Bit 6:6 monochrome/colour
70	Satellite 2 colour definition Bit 0:3 foreground colour Bit 4:7 background colour
71	MF144 mode instruction
72	MF144 command instruction
73	MF144 baudrate
74	MF144 configuration
75	MF144 buffer
76-127	Reserved.

Table 3-2: NVM format.

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 (AA) are maintained
;
;destroyed: none
```

```
write_nvm:
    push dx                 ; save registers
    push bx                 ;
    push cx                 ;
    push ax                 ;
    call nvm_read           ; read the old value
    mov  bl,al              ; save old value in bl
    mov  ah,0               ;
    mov  al,0               ;
    call nvm_read           ; 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,80H             ;
    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 signif. nibble
    out  dx,al              ;
    pop  cx                 ;
    mov  al,cl              ;
    and  al,0FH             ;
    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,0               ;
    call address_block      ; address the checksum block
    mov  dx,80H             ;
    mov  al,bh              ;
    mov  cl,4               ;
    shr  al,cl              ;
    out  dx,al              ;
    mov  al,bh              ;
    and  al,0FH             ;
    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
nvm_read:
    push dx             ;save registers
    push cx             ;
    push ax             ;
    call address_block  ;select block
    mov  dx,80H         ;
    pop  ax             ;
    shl  al,1           ;convert byte to nibble offset
    shl  al,1           ;
    xor  ah,ah          ;
    add  dx,ax          ;
    in   al,dx          ;
    add  dx,2           ;
    xchg ah,al          ;
    in   al,dx          ;
    mov  cl,4           ;
    shl  ah,cl          ;
    and  al,0FH         ;
    or   al,ah          ;transform nibbles to bytes
    pop  cx             ;
    pop  dx             ;
    ret                 ;


address_block:          ; select block number(ah)
    mov  dx,70H         ;
    in   al,dx          ;
    and  al,03FH        ;
    mov  cl,6           ;
    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 breaf 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 Partner 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:

```
AL = 21
```

At return the ES:SI register pair contains the segment and offset 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 F000:0000 (hex).

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 heigt of one character row is 14 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 720 by 350 pixels in alphanumeric and high resolution graphics mode and 360 by 350 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 bits 2 and 3 are used and the colours are represented by levels of intensity.

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	
180h	1	0
182h	3	2
184h	5	4
.		
.		
1BEh	63	62

Table 4-1: I/O address vs. palette cells.

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

Table 4-2: Palette cell selection - alphanumeric mode.

High resolution graphics:

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

Table 4-3: Palette cell selection - high resolution graphics.

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

Table 4-4: Palette cell selection -
medium resolution graphics.

Examples:

Alphanumeric mode:

character = 0100100010000000B

bit 0-9	character number = 128 address F000:2000 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 F000:0000 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 = 0111000000010000B

    bit 0-9   pixel block number = 16
              address F000:0200 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 + 20.

    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 Concurrent DOS 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,10H[bx]                ; 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:[bx]                ; 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 Partner" 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 Partner'
```

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 screen 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:

```
    AX = 21
```

At return the ES:BX register pair contains the segment and offset of the table and DX contains 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 (0-23).
; 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:[bx]     ; 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:

    AX = 22

At return BX contains the cursor position in the following encoding:

    DH = line (0-23)
    DL = coloumn (0-79)

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

    AL = 36
    DH = line (0-23)
    DL = 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 (hex notation) F000:0000.

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. This gives a total of 1024 character definitions.

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 9 bit wide is defined by the following bits:

```
xxxxxxxxx0111111B
```

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 9 pixels. The height of one character row is 14 videolines.

When the display operates in graphics mode all of the 32 Kbytes 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 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   0000000000111111B   ; Character definition.
         dw   0000000000111111B   ;
         dw   1101111110111111B   ; The character is 9 bit
         dw   1111111110111111B   ; wide (As the tail is
         dw   1110000000111111B   ; 0111111).
         dw   1100111110111111B
         dw   1101111110111111B
         dw   1101100000111111B
         dw   1101100000111111B
         dw   1101111110111111B
         dw   1100111110111111B
         dw   0000000000111111B
         dw   0000000000111111B
         dw   0000000000111111B
```

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)

Table 4-5: Console Control Characters.

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 (hex 1B) 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 explained below (- a summary may be found in Appendix F).

ESC A - Cursor Up

Moves the cursor up one line. If the cursor is already on the top line, the 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 Partner 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 20H (decimal 32) 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:

	Significance	
Bit	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: 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 Concurrent DOS.

ESC n - Disable Cursor

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

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 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 statusline 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. <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 4-6 on the next page:

Identifier	Function Key	Identifier	Function Key
;	F1	e	alt-F5
<	F2	f	alt-F6
=	F3	g	alt-F7
>	F4	h	alt-F8
?	F5	i	alt-F9
@	F6	j	alt-F10
A	F7	k	shift-F1
B	F8	l	shift-F2
C	F9	m	shift-F3
D	F10	n	shift-F4
E	F11	o	shift-F5
F	F12	p	shift-F6
G	Home	q	shift-F7
H	Up Arrow	r	shift-F8
I	A1	s	shift-F9
J	A2	t	shift-F10
K	Left Arrow	u	ctrl-F1
L	Return (keypad)	v	ctrl-F2
M	Right Arrow	w	ctrl-F3
N	A3	x	ctrl-F4
O	A4	y	ctrl-F5
P	Down Arrow	z	ctrl-F6
Q	Tab (keypad)	æ	ctrl-F7
R	Insert	ø	ctrl-F8
S	Delete	å	ctrl-F9
T	Print	ü	ctrl-F10
U	shift-A1	0	0
V	shift-A2	1	1
W	shift-A3	2	2
X	shift-A4	3	3
Y	alt-F11	4	4
Z	alt-F12	5	5
Æ	shift-F11	6	6
Ø	shift-F12	7	7
Å	ctrl-F11	8	8
Ü	ctrl-F12	9	9
a	alt-F1	+	+
b	alt-F2	-	-
c	alt-F3	,	,
d	alt-F4	.	.

Table 4-6: Function Key identifiers.

Example:

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

```
ESC  :  <  Partner  NULL

Hex: 1B  3A  3C  50  61  72  74  6E  65  72  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 20H (decimal 32) is added to the row numbers.

Example:

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

```
ESC  <  $  +        (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 20H (decimal 32) is added to the row numbers.

Example:

The following sequence scrolls row 0 to row 16 one line down:

```
ESC  >  $  +      (hex: 1B  3E  20  30)
```

ESC <238> - Change function key program terminator

<238> denotes one character with the decimal value 241.

The format of the sequence is:

```
ESC <238> terminator
```

The default value of terminator is 0. Possible values are 0 to 255.

ESC <239> - Set Tranparent mouse mode

After this sequence all characters received from the mouse are treated as if they came from the keyboard. The most significant byte of the character is set to 254 (use raw i/o to force the operating system to return 16 bit characters). Transparent mouse mode is used to connect another device than a mouse to the keyboard mouse interface.

ESC <240> - Set normal mouse mode

After this sequence data received from the mouse interface are treated in the console driver. Data about the mouse movement are obtained by use of int-28h function 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 most significant bits specifiy the start videoline and the four least significant bit specifiy the end videoline.

The videolines of a row are numbered 0-13. 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 0-13):

```
dec: 27  243  224          hex: 1B  F3  E0
```

The following sequence selects a double underline cursor (occupying videolines 12-13):

```
dec: 27  243  236          hex: 1B  F3  EC
```

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.

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 and at the same time 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.

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.

When the gcb_mx field is set to ff(hex), 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_grahics:
    mov  gcb_seg,es      ; initialize the buffer segment
    mov  al,0            ; function code for init graphics
    mov  ah,1            ; high resolution
    mov  cx,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
```

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 1:

```
; this routine sets the no-switch bit in the console
; control block. The keep flag is set in the procss
; descriptor so we cannot be terminated before the
; no-switch bit has been cleared.
;
os              equ       224
sys_ccb         equ       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_flag[bx],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_state[bx],cf_noswitch
    ret

unlock_console:
    mov  es,sysdat
    mov  bx,ccb_addr     ; clear noswitch bit
    and  es:ccb_state[bx],not cf_noswitch
    mov  bx,pd_addr      ; and turn off keep flag
    and  es:p_flag[bx],not pf_keep
    ret

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

Example 2:

```
; 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,400h        ; 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*21 + Y
;    pixel address = X mod 16
;
y_max    equ  349
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
    mov  cl,bl             ; save pixel address
    and  bx,0fff0h         ; mask out bit number
    mov  dx,bx             ;
    shl  bx,1              ; BX * 2
    shl  bx,1              ; BX * 4
    add  bx,dx             ; BX * 5
    shl  bx,1              ; BX * 10
    add  bx,dx             ; BX * 11
    shl  bx,1              ; BX * 22
    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_masks[bx]
    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 DOS 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.

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 = OFFH
    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 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 RC750 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
Ctrl+A1	enter setup mode
Ctrl+A2	no action
Ctrl+A3	wake up window manager
Ctrl+A4	full screen key

4.8 PC-mode

When a virtual console is in PC-mode (set by XIOS function 32) several functions behave different:

1. The status-line is disabled, allowing MS-DOS programs to use 25 lines.
2. Scan codes are returned by XIOS function IO-CONIN.
3. Function keys are not expanded (extended codes are returned).

4.8.1 PC-mode keyboard

In PC-mode a PC keyboard is emulated. The differences are explained in the following.

1. The scan codes for printable characters can not be expected to be correct.
2. NUMLOCK not supported. The numeric keys in the keypad always return their ASCII values.
3. Ctrl-Break not supported.
4. The following keys do not exist, but are emulated:

END	emulated by	A2
Page Up	-	A3
Page Down	-	A4

4.8.2 8-bit PC-mode

In 8-bit PC-mode (see App. A: function 71) the keyboard conversion is changed, so that the values returned for national characters is in accordance with the IBM 8-bit character set (see App. D).

As a consequense characters output to a printer not supporting 8-bit IBM character set must be converted. For this purpose an Int-28h function (see App. A: function 73) is offered. This function checks whether the console is in 8-bit PC-mode and if it is, converts the character supplied as parameter.

The CHAR8 program may be used to put the console in 8-bit PC-mode. It also initializes the alternative character font with an IBM PC compatible character set. Note that the alternative character set cannot be used for other purposes when the CHAR8 program has been run.

4.9 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
```

Four functions are provided:

```
    CL = 0 - Set Mouse Vector
    CL = 1 - Initialize mouse
    CL = 2 - Deinitialize mouse
    CL = 3 - Return mouse status
```

Function 0 is used by an application to supply its own interrupt vector. In this case all mouse handling must be done by the application. The interrupt routine is called with the byte received from the mouse in AL.

Function 0 is called with the following parameters:

```
    SI = Offset of Interrupt Routine
    DX = Segment of Interrupt Routine
```

Function 1 is used to apply a default interrupt routine to do the mouse handling. In this case the mouse status is examined by calling mouse function 3.

When function 3 (return mouse status) is called, register AL contains the mouse status at return:

- nothing happened: AL = 0

- button press: AL = 1

  register AH contains a button code:

left button:	20h
middle button:	21h
right button:	22h

- coordinate information: AL = 2

  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 Partner 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 Partner. 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 following example shows two routines which can be used for this purpose.

__Example:__

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

    ; read register address setup
    Mov  DX,5CH
    Or   AL,80H
    Out  DX,AL

    ; generate read pulse
    Or   AL,0A0H
    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,9FH
    Out  DX,AL
    Xchg AH,AL
    Ret

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

    ; write register address setup
    Mov  DX,5CH
    And  AL,1FH
    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,40H
Out  DX,AL

; wait at least 1 micro sec
Nop
Nop
Nop
Nop

; remove write pulse
And  AL,1FH
Out  DX,AL
Ret
```

6. Sound

The sound device produces sound via the loudspeaker located in the CRT display unit.

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

Table 6-1: Control registers.

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 frequency 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 Concurrent DOS 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  0224                     ;
    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      ; queue 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 shown in the following table:

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

Table 6-2: Attenuation control.

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, as shown in table 6-3. The first 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

Table 6-3: Shift rate select. Clock rate is 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  eloopl           ;
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. Serial Interface

This chapter describes the serial communication support on the Partner.

The Partner standard configuration includes an INTEL 8274 serial communication controller with two independent serial communication channels.

7.1 describes how Concurrent DOS supports the two communication channels.

7.2 describes how the INTEL 8274 serial communication chip is used in the Partner.

7.3 describes how to initialize the INTEL 8274 to asynchronous operation.

7.4 is an example of a program that use the INTEL 8274 for asynchronous communication.

7.1 Standard serial communication support

On Partner the Concurrent DOS operating system supports the following two communication channels:

Channel A:

Concurent DOS supports channel A as an extra console device (with console number 4), as a list device (with printer number 5) or as an auxiliary device (with aux number 0).

Channel B:

Concurrent DOS supports channel B as an extra console device (with console number 5), as a list device (with printer number 1) or as an auxiliary device (with aux number 1).

Both channels are operated in asynchronous mode as standard. By supplying ones own drivers, it is possible to run channel A in synchronous mode and to support X.21 signals (see 7.2).

When a channel is operated as a console device, access to this extra console is gained in the same way as access to the normal virtual consoles, i.e. using Concurrent DOS console input/output functions (ref.2). When a channel is operated as a virtual console a program cannot access the channel simultaneous with access to the normal virtual console. If this is wanted the channel should be operated as an auxiliary device.

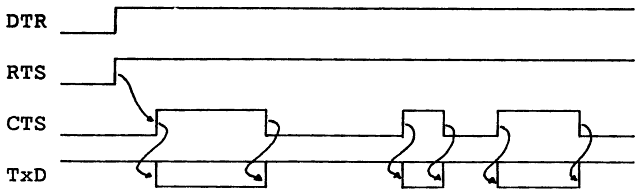
When a channel is operated as a list device, it is accessed through Concurrent DOS's list device functions (ref. 2.).

When a channel is operated as an auxiliary device, it is accessed through Concurrent DOS's auxiliary 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 (ref.5.).

7.1.1 V24 Handshake scheme

When operating the communication channels in the standard asynchronous mode the connected devices must adhere to the handshake scheme, based on the INTEL 8274 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.

7.2 Serial communication controller

Channel A:

Channel A (labelled 'V24/COM') is capable of operating in both synchronous and asynchronous mode and has support for X21 signals.

To take advantage of these capabilities, you must supply a driver fitted for your special needs. This is necessary because Concurrent DOS beyond disk handling supports character input/output only.

Channel A also has the capability to use DMA (direct memory access). Two DMA request inputs are assigned for this purpose:

DRQ 1:	Channel A transmitter
DRQ 2:	Channel A receiver

The DMA request input must be selected and the DMA channel reserved before the channel can utilize the DMA function (see 2.2 for further information about DMA usage).

Channel B:

Channel B (labelled 'RS232-C/V24') can only be used for asynchronous communication.

Both channels:

Both channel A and Channel B are supplied with two extra modem signals, namely DSR (Data Set Ready) and CI (Calling Indicator) the value of which may be read from the I/O address 210H. The resulting byte value has the following encoding:

bit 2:	Calling Indicator for channel A.
bit 3:	Data Set Ready for channel A.
bit 4:	Calling Indicator for channel B.
bit 5:	Data Set Ready for channel B.

The INTEL8274 serial communication controller has a built-in interrupt controller which is connected to the INTEL80186 INT1 and INTA1 pins. The INTEL8274 has been assigned interrupt level 40H-47H, which corresponds to the following interrupt vector addresses:

Channel B transmitter interrupt	0:100H
- - status -	0:104H
- - receiver -	0:108H
- - special receive -	0:10CH
Channel A transmitter interrupt	0:110H
- - status -	0:114H
- - receiver -	0:118H
- - special receive -	0:11CH

The INTEL 8274 uses the following I/O addresses:

Channel A command	30H
Channel A data	32H
Channel B command	34H
Channel B data	36H

7.2.1 Asynchronous communication using channel A

The baud rate clock to channel A may be supplied from one of two sources:

1) From an INTEL 8254 programmable interval timer (asynchronous operation). Counter output 0 is used for receive clock generation and Counter output 1 for transmit clock generation. Input to this timer is a 4 Mhz clock.

2) From the transmitter and receiver clock pins on the port terminals (synchronous operation).

To select INTEL 8254 as clock source, output the value 8 to I/O address 76H.

The INTEL 8254 uses the following I/O addresses:

Counter 0:	40H
Counter 1:	42H
Control word:	46H

The two counters are programmed to operate in mode 3 (square wave mode) with 16 bit binary count by outputting the values 36H (counter 0) and 76H (counter 1) to the control word address.

The receive/transmit baudrate is set by outputting one of the values in the following table to the appropriate I/O address (40H or 42H). The value is output as two bytes with the least significant byte first.

Baudrate	**Value**
50	5000
75	3333
110	2273
150	1667
300	833
600	416
1200	208
2400	104
4800	52
9600	26

Example:

Initialize INTEL 8254 to generate a 1200 baud clock for channel A receiver and a 300 baud clock for channel A transmitter.

```
    Mov  DX,46H           ;
    Mov  AL,36H           ; Initialize counter 0 to:
    Out  DX,AL            ; mode 3, 16 bit binary count
    Mov  AL,76H           ; Initialize counter 1 to:
    Out  DX,AL            ; mode 3, 16 bit binary count
    Mov  AX,208           ; Initialize counter 0 count
    Mov  DX,40H           ; value to obtain 1200 baud:
    Out  DX,AL            ; least significant byte of value
    Xchg AH,AL            ;
    Out  DX,AL            ; most significant byte of value
    Mov  AX,833           ; Initialize counter 1 count
    Mov  DX,42H           ; value to obtain 300 baud:
    Out  DX,AL            ; least significant byte of value
    Xchg AH,AL            ;
    Out  DX,AL            ; most significant byte of value
```

Further information on how to program the INTEL 8254 programmable interval timer may be found in the Intel reference documentation.

7.2.2 Synchronous communication using channel A

To select the transmitter and receiver clock pins on the port terminals as clock source, output the value 9 to I/O address 76H.

If channel A is used to connect the Partner to an X.21 network the following items should be considered:

1) The modem cable must have a connection between pin 11 and pin 7 (ground) before the signal control logic in the Partner will recognize the X21 signals R(eceive), I(ndication), S(ignal element timing), T(ransmit) and C(ontrol).

2) The X21 input signals are obtained from the INTEL8274 signals in the following way:

   R is the INTEL8274 RxD signal.
   I is the INTEL8274 CTS signal.

   When the R and the I signal have been low (off) for a period of 16 contiguous bit intervals the INTEL8274 signal DCD will be set to indicate either a DCE clear indication or a DCE clear confirmation

3) The X21 output signals are obtained from the INTEL8274 signals in the following way:

   T is the INTEL8274 TxD signal.
   C is the INTEL8274 DTR signal.

   T is gated with the INTEL8274 RTS signal to make it possible to let the INTEL8274 send 'all zeroes' in the idle state (RTS=0 means that an 'all zeroes' bit pattern will be sent, RTS=1 means that the TxD signal will be sent).

7.2.3 Asynchronous communication using channel B

The baudrate clock to channel B is supplied from timer 0 on the the INTEL 80186 chip. Input to this timer is derived from the 6 Mhz system clock. This timer uses the following addresses:

Count:	FF50H
Max count A:	FF52H
Max count B:	FF54H
Control:	FF56H

The count register is reset by outputting the value 0 to the count address.

Max count A and max count B are set to one of the values in the following table.

Baudrate	Max count A	Max count B
50	936	936
75	624	624
110	426	426
150	312	312
300	156	156
600	78	78
1200	39	39
2400	19	20
4800	10	10
9600	5	5

The control word is set to alternating, continuous count without interrupt by outputting the value C003H to the control word address.

Example:

Initialize the INTEL 80186 timer 0 to generate a 2400 baud-rate clock for channel B (This channel will always use the same baudrate on both transmitter and receiver).

```
    Mov  DX,0FF50        ; count register address
    Xor  AX,AX           ;
    Out  DX,AX           ; reset count register
    Add  DX,2            ; max count A register address
    Mov  AX,19           ;
    Out  DX,AX           ; set max count A
    Add  DX,2            ; max count B register address
    Mov  AX,20           ;
    Out  DX,AX           ; set max count B
    Add  DX,2            ; timer 0 control word address
    Mov  AX,0C003H       ;
    Out  DX,AX           ;set alt.,cont.,no interrupt
```

Further information about the INTEL 80186 may be found in the relevant INTEL documentation.

7.3 Initializing the INTEL 8274

Three Int-28h functions are available for initializing the INTEL 8274 to operate in asynchronous mode:

Int-28h function 24 is available for compatibility reasons only. New application should use function 54 (see below). Function 24 is used to initialize a channel according to a parameter block with the following format.

+ 0	Channel number (0: Comm; 1: V24)
+ 1	Mode (0: Console; 1: Printer)
+ 2	Protocol (0: None; 1: Xon-Xoff)
+ 3	Receiver baud rate (0: 50; 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 INTEL 8274 write register specification (see relevant INTEL doc.
+ 6	Register no.
+ 7	Register contents
+ 8	Register no.
+ 9	Register contents

-
-
-
-
etc.

When channel number is 1, the transmitter and receiver baud rates must be equal (this channel has only one baud rate generator).

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

Example:

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

    ; execute function 24
    Mov  AX,24
    Int  28H

    ; clean-up stack
    Add  Sp,4
    Ret

ParamBlock:
    Db   0     ; Channel 0 (comm line)
    Db   0     ; Console mode
    Db   0     ; No protocol
    Db   6     ; Receive 1200 baud
    Db   8     ; Transmit 4800 baud
    Db   4     ; 4 registers to program
    Db   1,17H; write register 1:
               ;   Interrupt on all characters
               ;   Vectored interrupt
               ;   Transmit and external status int. enable
    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 54 is used to initialize either the built-in 8274, the 8251A's on a MF140 adapter or the 8251A on the MF144 adapter. The format is as follows:

+ 0	Virtual Console No. (4-8) 4: COMM/V24 5: V24/RS232 6: Channel A on MF140 7: Channel B on MF140 8: Channel A on MF144
+ 1	Mode 0: Console, 1: Printer, 2: Satellite
+ 2	Protocol 0: None, 1: Xon-Xoff, 2: Satellite
+ 3	Receiver baud rate 0: 50, 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	Bit per Character for Receiver 5-8
+ 6	Bit per Character for Transmitter 5-8
+ 7	Stop Bit 0: 1, 1: 1.5, 2: 2
+ 8	Parity 0: Odd, 1: Even, 2: No
+ 9	DTR and RTS 00H: DTR ON and RTS OFF 01H: DTR ON and RTS ON 10H: DTR OFF and RTS OFF 11H: DTR OFF and RTS ON

When channel number is 1, the transmitter and receiver baud rates must be equal (this channel has only one baud rate generator).

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

Example:

This example initialize the comm/v24 port in the same way as the previous example.

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

    ; execute function 54
    Mov  AX,54
    Int  28H

    ; clean-up stack
    Add  Sp,4
    Ret

ParamBlock:
    Db   0    ; Channel 0 (comm line)
    Db   0    ; Console mode
    Db   0    ; No protocol
    Db   6    ; Receive 1200 baud
    Db   8    ; Transmit 4800 baud
    Db   7    ; Receive character length
    Db   7    ; Transmit character length
    Db   0    ; 1 stop bit
    Db   1    ; Even parity
    Db   01h  ; DTR and RTS on

    END
```

Int-28h function 50 is used to initialize the communication controller according to the preconfiguration (NVM). The only parameter to this function is the virtual console number which must be in register DL when the function is entered. Channel numbers are assigned as follows:

4: COMM/V24
5: V24/RS232
6: First 8251A on a MF140 adapter
7: Second 8251A on a MF140 adapter
8: 8251A on MF144 adapter

If the channel has been used for synchronous communication, the baud rate clock selection bit must be reestablished before this function is entered (channel 4 only).

Example:

```
ResetSIO:
    Mov  AX,50
    Mov  DL,4
    Int  28H
    Ret
```

7.4 Sample asynchronous communication program

All examples in this chapter use the following declarations:

```
P_Flagset      Equ   133; Concurrent DOS flagset function
P_Flagwait     Equ   132; -             -  flagwait -

ReceiveFlag    Equ    13; Flag allocated to ch.A receive
TransmitFlag   Equ    14; Flag allocated to ch.A xmit

DataPort       Equ    32H; I8274 ch.A data port
CommandPort    Equ    30H; i8274 ch.A 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 INTEL 8274 controller and initialize all driver variables).

Example:

```
Initialize:
    ; get sysdat segment
    Mov  CL,154
    Int  224
    Mov  sysdat,ES

    ; get dispatcher address
    Mov  AX,ES:.38H
    Mov  dispatcher,AX
    Mov  AX,ES:.3AH
    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,110H; first vector for ch. A
    Mov  AX,Offset TransmitInterrupt
    Stos AX
    Mov  AX,CS
    Stos AX
    Mov  AX,Offset StatusInterrupt
    Stos AX
    Mov  AX,CS
    Stos AX
    Mov  AX,Offset ReceiveInterrupt
    Stos AX
    Mov  AX,CS
    Stos AX
    Mov  AX,Offset SpecialReceiveInterrupt
    Stos AX
    Mov  AX,CS
    Stos AX
    Sti

    ; Initialize INTEL8274 Controller
    ; See 7.3
    Call SetSIO

    Ret

sysdat         rw        1
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 initializes the controller to send 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
```

The INTEL 8274 serial communication controller has a built-in interrupt controller which can be programmed to work in different modes (see the Intel reference documentation for details).

The XIOS driver initializes the controller to work with vectored interrupt. In this mode the controller needs 4 interrupt routines for each channel:

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 8274 controller and 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.

3) Special receive interrupt routine. This routine will gain control when an erroneous character (with parity error) is received. The action of this routine will depend on the actual application.

4) External status interrupt routine. This routine will gain control when the status of the modem signals on the INTEL 8274 controller changes. The action of this routine will depend on the actual application.

Example:

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

    ; reset transmit buffer empty
    Mov  DX,CommandPort
    Mov  AL,28H
    Out  DX,AL

    ; execute non specific end of interrupt
    Call Sio_EOI

    Mov  DX,TransmitFlag
    Call Flagset
    Jmp  DispatchReturn ;
```

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

    ; reset external status interrupt
    Mov  DX,CommandPort
    Mov  AL,10h
    Out  DX,AL                 ;

    ; execute non specific end of interrupt
    Call Sio_EOI               ;

    Jmp  NoDispatchReturn      ;


ReceiveInterrupt:
    ; save context
    Push DS
    Push ES
    Pusha

    ; 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


SpecialReceiveInterrupt:
    ; save context
    Push DS
    Push ES
    Pusha

    ; execute non specific end of interrupt
    Call Sio_EOI               ;
    ; execute error reset command
    Mov  DX,CommandPort
    Mov  AL,30H
    Out  DX,AL

    Jmp  NoDispatchReturn


Sio_EOI:
    Mov  DX,0FF22H                  ; non specific end of
    Mov  AX,8000H                   ; interrupt to internal
    Out  DX,AX                      ; interrupt controller.
    Mov  AL,38H                     ;
    Out  CommandPort,AL             ;  end  of  interrupt  to
sio                                 ; NOTE: end of interrupt
                                    ; commands are always
                                    ; issued on channel A,
                                    ; even when channel B is
                                    ; used
    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
    Mov       DS,CS:sysdat
    Callf     CS:Dword Ptr supervisor
    Pop       DX
    Test      AX,AX
    Jz        FlagSet_ret

    ; if error code='flag abandoned'
    ; then try again
    Cmp  CL,2AH
    Jz   FlagSet

FlagSet_ret:
    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
```

8. Disk System

The disk configuration of a Partner consists of:

0,1 or 2 flexible disk drives
0,1,2 or 3 winchester disk drives.

Floppy disk drives are always housed inside the Partner unit while the winchester disk drives could be either internal or in a separate unit (external).

Partner disk drives are always assigned drive letters in sequential order (floppy disk drives first).

8.1 Floppy disk characteristics

The Partner disk format uses a sector-to-sector skew factor of 1, and a track-to-track skew factor of 0, i.e. no track skewing at all.

The Partner 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 (physical):

Capacity	:	1604 Kbytes unformatted
Recording density	:	9646 BPI
Track density	:	96 TPI
Cylinders	:	80
Tracks	:	160
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

Precompensation (write)

All cylinders	: 125 nsec

Floppy disk format (CP/M):

Capacity	: 1232K formatted
Cylinders	: 77
Tracks	: 154
Sectors/track	: 8
Sector length	: 1024

Track format:

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

Sector format:

No. of bytes	Value (hex)
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)

CP/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)
4	reserved tracks
616	blocks per disk
384	directory entries (FCB's) per disk
128	directory entries (SFCB's) per disk
1232	K bytes total disk capacity

Floppy disk format (DOS):

Capacity	:	1200K formatted
Cylinders	:	80
Tracks	:	160
Sectors/track	:	15
Sector length	:	512

Track format:

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

Sector format:

No. of bytes	Value (hex)
12	* 00 (gap1/gap3)
3	* F5 (writes A1)
1	* FE (ID addres mark)
1	* track no.
1	* sector no.
1	* 02 (sector length)
1	* F7 (2 CRC written)
22	* 4E (gap 2)
12	* 00
3	* F5 (write A1)
1	* FB (data addressmark)
512	* E5 (data)
1	* F7 (2 CRC written)
54	* 4E (gap 3)

DOS drive characteristics:

80	cylinders per disk
2	track per cylinder
15	sectors per track
512	bytes per sector
2	sectors per block (1 K bytes block size)
0	reserved tracks
1193	blocks per disk
224	directory entries (FCB's) per disk
1200	K bytes total disk capacity

8.2 Floppy disk controller

The floppy disk controller is based on the WD1797 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
0200H	I	Read FDC status register
	O	Write control command
0202H	I	Read FDC TRACK register
	O	Write FDC TRACK register
0204H	I	Read FDC SECTOR register
	O	Write FDC SECTOR register
0206	I	Read FDC DATA register
	O	Write FDC DATA register
0210	O	Write FCR register
	I	Not defined

Further information on programming the registers on address 0200H-0206H 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	Precompensation	0 125 nsec.
		1 125 nsec.
5	Not used	must be 0
6	Not used	must be 1
7	Ready control	0 Ready from drive
		1 Ready always set

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 (function 5-11).

8.4 SCSI interface

The Partner SCSI (Small Computer System Interface, ref. ANSI X3T9.2/82-2 Rev. 3) is used for attachment of winchester disks.

The optional busarbitration is not implemented.

The interface consists of three elements:

1) A data port accessible via I/O address 10H.

2) A bus signal input port accessible via I/O address 72H with the following layout:

7	6	5	4	3	2	1	0
undefined			C/D	BSY	I/O	MSG	RST

3) 4 flip flops with the following contents:

3	2	1	0
SEL	EXP I/O	ATN	RST

A flip flop can be set/reset in the following way:

To set/reset bit no. x, output the following value to I/O address 76H:

7	6	5	4	3	2	1	0
0	0	0	0	0	x		1/0

Example:

Set the SEL flip flop to 1:

```
Mov  AL,00000111B
Out  76H,AL
```

The three signals SEL, ATN and RST are directly output on the bus. The flip flop 'EXP I/O' is used to control the DMA as follows:

A data transfer request from the SCSI bus ('C/D' = 0) gives rise to a DMA request if the state of the I/O line matches with the state of the 'EXP I/O' flip flop, else an interrupt is generated.

A control transfer request ('C/D' = 1) generates an interrupt. A read or write to I/O address 10H generates an 'ACK' on the SCSI bus.

8.5 SCSI interface driver

The SCSI interface driver is part of the XIOS disk driver which means that the IOSELDSK, IOREAD and IOWRITE functions are supported (see ref. 3).

The driver uses DMA channel 0 (assigned low priority) for data transfers to and from the winchester disks.

The sector size used in Partner winchester disks is 512 bytes, and the controller is able to transfer up to 255 sectors at a time. This value is restricted to 16 Kbytes by the operating system CCP/M.

The eight SCSI addresses are used to distinguish between different controller and disk combinations.

The four outermost tracks on a Partner winchester disk are reserved for system use. The first sector on track 0 contains initialization and configuration information which is used during system initialization. The contents are dependent on the actual controller and winchester disk type.

Track 0, sector 0:

```
    "RC750 "                ; identification text
    5,1,0,3,1,53,0,0,0,0    ; 10 bytes winchester
                            ; initialization sequence.
                            ; (This sequence is used for
                            ; the DTC510B controller in con-
                            ; nection with a NEC disk.)
    512                     ; sector size
    18,5,31,1,2699,         ; Disk parameter block
    1279,0FFH,0C0H,8000H,   ;
    4,2,3                   ;
    0                       ; Control byte used in read
                            ; and write commands (control-
                            ; ler dependent).
    0,0,0,0                 ; reserved.
    1324,1,75               ; shipping zone values used by
                            ; the Program 'NEDLUK'
```

Track 0, sector 1 and forward contain a loader program image.

The SCSI interface driver is extended with a general SCSI interface command (see appendix A).

Commands to the winchester controller are passed in a 6 bytes long command description block (CDB) with the following format:

	MSB							LSB
BYTE NO.	7	6	5	4	3	2	1	0
0	CLASS CODE			OPCODE				
1	LUN			LOGICAL ADDRESS 2				
2	LOGICAL ADDRESS 1							
3	LOGICAL ADDRESS 0							
4	See actual controller manual for details							
5	See actual controller manual for details							

Byte 1 of the CDB contains a 3-bit LUN (Logical Unit Number) and the five most significant bits of the Logical Address (Logical Address 2). The LUN is contained in every CDB.

The LUN must correspond to the Drive Select Address within the drive.

Drive Select Address 1 = LUN 0
Drive Select Address 2 = LUN 1
Drive Select Address 3 = LUN 2
Drive Select Address 4 = LUN 3

Logical Address is a 21-bit address. It is comprised of 5 bits from byte 1, 8 bits from byte 2 and 8 bits from byte 3. With the use of logical address, a unique address (which corresponds to the Logical Address value) is assigned to each individual sector within a disk drive by the controller.

To better understand the Logical Address concept, view the sectors of any drive as sequentially numbered - starting with 0 (at track 0, sector 0) and ending (with the accumulated total of all sectors) at the last sector, of the last track, of the last disk surface.

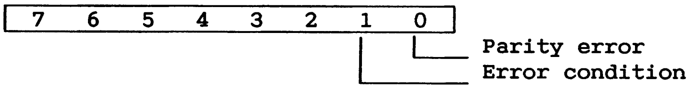
Byte 2 contains 8-bits of the Logical Address (Logical Address 1).

Byte 3 contains the eigth least significant bits of the Logical Address (Logical Address 0).

The significance of Byte 4 and Byte 5 depends on the CDB's Class Code and OpCode. Consult the actual controller manual for further details.

When using the extra XIOS function 16 you must fill in the CDB with correct values before call, therefore consult the actual controller manual for information.

Error conditions that may have occured during the execution of the command are returned in register AL:



The contents of register AL is only valid if register AH is 0.

To fetch the LUN (logical drive number) use the XIOS extra function number 17.

Example:

This example reads track 0, sector 0 and 1 from winchester disk B into buffer.

```
    CSEG
    ORG  100H

    sectors              Equ       2

    wd_command           Equ       Byte Ptr [DI]
    wd_lun_adr2          Equ       Byte Ptr 1[DI]
    wd_adr1              Equ       Byte Ptr 2[DI]
    wd_adr0              Equ       Byte Ptr 3[DI]
    wd_mscnt             Equ       Byte Ptr 4[DI]
    wd_control           Equ       Byte Ptr 5[DI]

    ; Get controller address byte and
    ; logical unit number (LUN) for drive B
    Mov  AL,1
    Mov  AH,0
    Push AX
    Mov  AX,17
    Int  28H
    Mov  CTRAD,AL
    Mov  LUN,AH

    ; Fill in CDB values and push address
    ; on the stack
    Mov  AL,1            ; Drive B
    Mov  AH,1            ; Input
    Push AX
    Push CS
    Mov  AX,Offset CDB
    Push AX

    Mov  AX,512 * sectors; DMA byte count
    Push AX
    Push CS              ; DMA segment
    Mov  AX,Offset BUFFER
    Push AX              ; DMA offset

    Mov  DI,Offset CDB
    Mov  wd_command,08H ; read command
    Mov  AL,LUN
    Mov  wd_lun_adr2,AL
    Mov  CL,5
    Shl  wd_lun_adr2,CL
    Mov  wd_adr1,0      ; logical sector
    Mov  wd_adr0,0
    Mov  wd_mscnt,sectors
    Mov  wd_control,0

    Mov  AX,16
    Int  28H

    ; returns AX = 0 if success

    ; terminate process
    Mov  CL,0
    Int  224

    CDB       Rb        6

    Buffer    Rb        512 * sectors

    LUN       Rb        1

    CTRAD     Rb        1

    END
```

9. Parallel Interface

The parallel interface on the Partner is primarily intended for attachment of printers, but may also be used as a general input/output port.

Concurrent DOS supports the parallel interface as a printer device (printer 0).

9.1 Parallel interface description

An overview of the electrical signals used in the interface 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

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 four 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 (OUT 250H):

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

Bit state	TTL ouput
0	LOW
1	HIGH

Data input port (IN 250H):

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

Pin TTL level	Bit state read
LOW	0
HIGH	1

Control register (OUT 260H):

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	Signal (Pin no.)	Description
0	-,STROBE (1)	See above
1	-,AUTOLF (1)	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 interface has been assigned interrupt vector address 0:98H.

Status input port (IN 260H):

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	Connector Pin no.	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

```
list_flag   Equ     12

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

    ; get dispatcher address
    mov     ax,es:.38H
    mov     dispatcher,ax
    mov     ax,es:.3AH
    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,120H+4*6           ; level 6
    mov     ax,offset ParallelInterrupt
    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 occure 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


ParallelInterrupt:

    ; 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

flagset:
    push             dx
    mov              cl,133
    mov              ds,cs:sysdat
    callf   cs:dword ptr supervisor
    pop              dx
    test             ax,ax
    jz               flagset_ret

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

10. Local Area Network

The network software in the Partner 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 commom datalink service.

This section describes in detail the datalink service interface enabling the programmer to implement higher layer network software using the Partner 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 Partner mainly implemented by the Intel 82586 Ethernet controller (ref. 10). To enable the programmer to access the controller directly, Partner specific information about handling the controller is given.

10.1 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, data link, 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.

10.2 The datalink layer service interface

The data link layer (the RCLLC entity) is implemented as a Concurrent DOS Resisdent 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 Concurrent DOS queue interprocess communication facility. The communication between the driver and a client will have four fundamental forms: REQUEST, CONFIRM, INDICATION and INDICATION ACKNOWLEDGE. These are described in the following.

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	Use
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	Use
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 unsyn-cronized 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	**Use**
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.

NOTICE! **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
```

10.2.1 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.

10.2.1.1 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.

Format of the request buffer:

Byte number	Use
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	**Use**
0-5	reserved by the driver
6-7	the SAP mask used to prevent establishment of undesired connections, cf. section 10.1.
8-45	client defined information

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

10.2.1.2 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.

Format of the returned client information buffer:

Byte number	Use
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.

10.2.1.3 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.

Format of the request buffer:

Byte number	Use
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.

10.2.1.4 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 all outstanding resources i.e. queues or databuffers can be regarded as released.

Format of the returned deactivate buffer:

Byte number	Use
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).

Notice: 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.

10.2.1.5 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	**Use**
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.

10.2.1.6 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	Use
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'.

10.2.1.7 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	Use
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.

10.2.2 Type 1 Service

Type 1 service comprises unacknowledged connectionless data transfer between SAPs.

10.2.2.1 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.

Format of the request buffer:

Byte number	Use
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.

10.2.2.2 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.

Format of the returned data buffer:

Byte number	Use
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'

10.2.2.3 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.

Format of the indication datastructure:

Byte number	Use
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.

10.2.3 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.

10.2.3.1 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	Use
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.

10.2.3.2 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	Use
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.

10.2.3.3 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	Use
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.

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

10.2.3.5 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	Use
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.

10.2.3.6 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.

Format of the returned data buffer:

Byte number	Use
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.

10.2.3.7 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	Use
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.

10.3 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 PARTNER (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.

10.3.1 Controller specific information

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

Interrupt vector

The net controller interrupt is connected to the CPU via the external Intel 8259A interrupt controller. The interrupt vector address is obtained by means of int-28h function 67:

Setting up interrupt vector:

```
;** assumption : the interrupt routine 'NETINT' is placed
;**              in the current code segment
        mov  ax,67
        int  28h         ;returns level 5 vector address
        mov  di,ax       ;save interrupt vector address
        xor  ax,ax
        es,ax            ;es=interrupt table segment base
        mov  ax,offset NETINT
        cli              ;disable all interrupts
        stosw            ;store offset part of net interrupt
                         ;routine
        mov  ax,cs       ;get segment pointer
        stosw            ;store segment part of net
                         ;interrupt routine
        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:

```
        in   al,2        ;8259 interrupt controller I/O
                         ;address - get current enable mask
        and  al,11011111b
                         ;enable net bit 5
        out  2,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.

<u>A channel attention is performed:</u>

```
        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. 10. Ref. 10 prescribes that the System Configuration Pointer (SCP) begins at location Offff:6 (Partner Prom address room). In Partner the SCP is placed in the RAM address room. The SCP segment is 3000h and the SCP offset is Offf6h. The SYSBUS byte in the SCP must be 0 to indicate 16 bits bus word mode.

10.4 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 constitute a functional extension to the type 1 procedures.

In general, an RCLLC protocol element may be a <u>command</u> protocol element or a <u>response</u> 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.

10.4.1 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 10.4.3.

10.4.1.1 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.

10.4.1.2 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 broad-casting 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.

10.4.1.3 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.

10.4.2 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 10.4.3.

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 that 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 10.4.3.4
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.

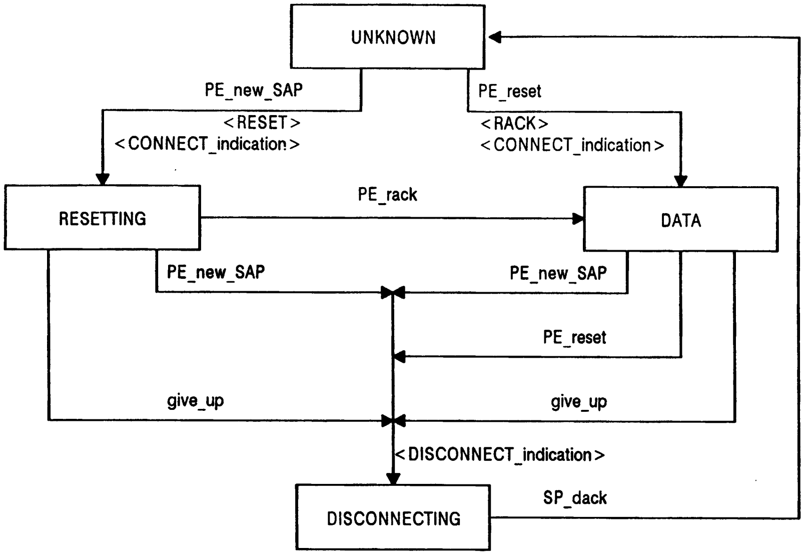


Figure 10-1: State graph for a connection.

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 10.4.3.4), 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 10.4.3.4), 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.

10.4.3 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 octet similarly from left to right. The least significant bit position within an octet, 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.	0	1	2	3	4	5	6	7
octet no. 0	0	0	DSAP					
1	C/R	0	SSAP					
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 octet 0 and bit 1 of octet 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 octet 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; thus 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.:

11000000	UI, Unnumbered Information
1111X101	XID, eXchange IDentification
1100X111	TEST

The use of the Type 1 Information field depends on the type of protocol element, and is described for each element type in section 5.1.

The protocol elements for client network service are UI commands addressed to an individual SAP with three extra octets 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.	0	1	2	3	4	5	6	7		
octet no. 0	0	0	Netno							
1	0	0	Netno						type 1	
2	1	1	0	0	0	0	0	0	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
10000000 (binary 1)	RESET
01000000 (binary 2)	RACK
11000000 (binary 3)	DATA
00100000 (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 section 5.2.

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.

10.4.3.1 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.

10.4.3.2 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.	0	1	2	3	4	5	6	7
octet no. 6	1	0	0	0	0	0	0	1
7	1	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0

10.4.3.3 TEST

The Type 1 Information field is used to hold a test data unit. The associated procedure is described in subsection 10.4.1.2.

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.

10.4.3.4 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 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 onwards contains the client_info passed from the client when the SAP was activated.

10.4.3.5 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.

10.4.3.6 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.

10.4.3.7 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 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.

10.4.3.8 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.

11. Adapters

RC Computer delivers a number of I/O Adapters solving different tasks.

Adapters for installation in the Partner PC are connected via a connector positioned inside the cpu unit and marked with the symbol 'J7'.

Chapter 11.1 together with appendices G and J contains the information necessary to implement new adapters.

11.1 MF140 Dual Satellite Adapter

This chapter describes the MF140 Adapter. MF140 is equipped with two independent asynchronous V24 channels. To serve as an example for implementing adapters to the Partner, schematics are shown in appendix H and a testprogram exercizing the MF140 is listed in appendix I.

11.1.1 MF140 Driver

The 2 channels on MF140 are supported as virtual console 6 and 7 or as auxiliary device 2 and 3.

Access to the channels are gained using the appropriate operating system call (console or auxiliary).

The two channels are controlled using the following Int-28h Functions (Appendix A):

Function 25:	Bypass Protocol
Function 50:	Reset Channel to the State of the Pre-konfiguration
Function 54:	Initialize Serial Communication Controller.
Function 55:	Get Serial Communication Controller Status.
Function 65:	Get Serial Communication Driver Status.

11.1.2 MF140 Hardware Description

Intel 8251A Channel A

Command Instruction Address:	282H
Status Read Address:	282H
Data I/O Address:	280H

For Further Details see the Intel Reference Documentation.

Intel 8251A Channel B

Command Instruction Address:	286H
Status Read Address:	286H
Data I/O Address:	284H

For Further Details, consult the Intel Reference Documentation.

Intel 8254

Counter 0 Address:	300H
Counter 1 Address:	302H
Counter 2 Address:	304H
Control Word Address:	306H

For Further Details, consult the Intel Reference Documentation.

On 8MHz Partner's the following figures should be used as count values to obtain different baudrates:

Count	**Baud**
5000	50
3333	75
2273	110
1667	150
833	300
416	600
208	1200
104	2400
52	4800
26	9600
13	19200

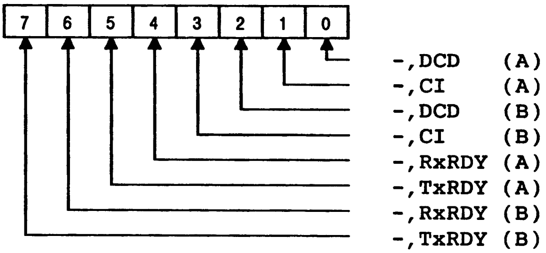
On 6MHz Partner's the following figures should be used as count values to obtain different baudrates:

Count	Baud
3750	50
2500	75
1705	110
1250	150
625	300
313	600
156	1200
78	2400
39	4800
20	9600
10	19200

Status Register

Address: 288H

Encoding:



RxRDY(A), TxRDY(A), RxRDY(B) and TxRDY(B) are OR'ed together and connected to the Adapter connector interrupt pin (level 7 on the main-board Intel 8259 PIT). This means that it is necessary to investigate this Status Register to determine the source of an interrupt. More than one of the Interrupt Sources can be active at the same time (i.e. all active sources should be serviced when the interrupt service routine is activated). Receiver Interrrupt are cleared when the character is read from the Intel 8251A. Transmitter Interrupts are cleared by disabling the transmitter (TxEN) in the Intel 8251A.

The receiver circuit is build in a way that prevents receiving when there is no DCD signal present.

Diagram drawings for the MF140 Adapter is found in appendix I.

11.2 MF141 Parallel/Printer Adapter

This chapter describes the MF141 Adapter.

The MF141 adapter contains a parallel (printer) interface identical to the parallel interface found on the CPU board.

11.2.1 MF141 Driver

The MF141 Adapter is supported by the operating system as printer number 2.

11.2.2 MF141 Hardware Description

The parallel port on the MF141 is controlled in exactly the same way as the builtin parallel port (See chapter 9).

The only difference is in the i/o adress and the interrupt level assignment.

The data register on the MF141 has the address 280H while the Control register has the address 300H.

As for all Adapters the MF141 use level 7 on the main-board Intel 8259 PIT.

11.3 MF142 NRZI Adapter

This chapter contains a description of the MF142 NRZI Adapter. The MF142 is build around an Intel 8274 Serial Communication Controller. Additional logic circuitry adds the possibility to use NRZI data encoding. NRZI (or Invert-on Zero) data encoding is a way to transfer signal-element timing over a basically asynchronous datalink. The operation is as follows: Every time a "Zero" is sent, the signal state is inverted, whereas a "One" leaves the signal state where it was. Synchronization is then derived from the signal transitions. The fact that "Ones" leave the line steady implies that not to many concatenated "Ones" should be sent. As the SDLC has provision for "zero insertion" limiting the maximum number of "Ones" to 7, NRZI is an ideal coding for this protocol.

11.3.1 MF142 Driver

The MF142 is intended for use in connection with terminal-emulator that contains the necessary driver support (RC Computer delivers an 3270 terminal-emulator that use the facilities of the adapter). There is no built-in support of this adapter in the operating system.

11.3.2 MF142 Hardware Description

Channel A of the Intel 8274 is capable of operating in both synchronous and asynchronous mode and has support for X21 signals.

Channel A also has the capability to use DMA (direct memory access). Two DMA request inputs are assigned for this purpose:

DRQ 6:	Channel A transmitter
DRQ 7:	Channel A receiver

The DMA request input must be selected and the DMA channel reserved before the channel can utilize the DMA function (see 2.2 for further information about DMA usage).

Channel B is used as control and status register only.

Channel B supplies channel A with two extra modem signals, namely DSR (Data Set Ready) and CI (Calling Indicator) the value of which may be read in Read Register 0 on channel B:

bit 3:	calling indicator
bit 5:	data set ready

The Intel 8274 serial communication controller interrupt line is connected to the Intel 8259 IR7.

Upon Interrupt from the 8274, two read operations should be performed on I/O address 2B0H simulating "INTA" operations.

The result of the first read is unpredictable and should be disregarded. The second read operation gives a byte identifying the cause of the interrupt.

Channel B transmitter interrupt	48H
Channel B status interrupt	49H
Channel B receiver interrupt	4AH
Channel B special receive interrupt	4BH

Channel A transmitter interrupt	130H
Channel A status interrupt	134H
Channel A receiver interrupt	138H
Channel A special receive interrupt	13CH

The 8274 uses the following I/O addresses:

Write:	
Channel A command	2ECH
Channel A data	2E8H
Channel B command	2EEH
(Channel B data	2EAH)

Read:	
Channel A status	2E4H
Channel A data	2E0H
Channel B status	2E6H
(Channel B data	2E2H)

Asynchronous communication

The baud rate clock may be supplied from one of two sources:

1) From an Intel 8254 programmable interval timer (asynchronous operation). Counter output 0 is used for receive clock generation and counter output 1 for transmit clock generation. Input to this timer is a 4 Mhz clock.

2) From the transmitter and receiver clock pins on the port terminals (synchronous operation).

The Intel 8254 uses the following addresses:

Read:		Write:	
Counter 0:	2D0H	Counter 0:	2D8H
Counter 1:	2D2H	Counter 1:	2DAH
Counter 2:	2D4H	Counter 2:	2DCH
Control word:	2D6H	Control word:	2DEH

The two counters (0 and 1) are programmed to operate in mode 3 (square wave mode) with 16 bit binary count by outputting the values 36H (counter 0) and 76H (counter 1) to the control word address.

The receive/transmit baudrate is set by outputting one of the values in the following table to the appropriate I/O address (2D8H or 2DAH). The value is output as two bytes with the least significant byte first.

Baudrate	Value (async)	Value (nrzi)
50	5000	-
75	3333	53328
110	2273	36368
150	1667	26672
300	833	13328
600	416	6672
1200	208	3328
2400	104	1664
4800	52	832
9600	26	416
19200	3	208

Counter 2 is used to generate X21 timeout and should be programmed to operate in mode 2 with an initial count of 16.

Example:

Initialize Intel 8254 to generate a 1200 baud clock for receiver and a 300 baud clock for transmitter.

```
Mov  DX,2DEH         ;
Mov  AL,36H          ; Initialize counter 0 to:
Out  DX,AL           ; mode 3, 16 bit binary count
Mov  AL,76H          ; Initialize counter 1 to:
Out  DX,AL           ; mode 3, 16 bit binary count
Mov  AX,208          ; Initialize counter 0 count
Mov  DX,2DAH         ; Value to obtain 1200 baud:
Out  DX,AL           ; Least significant byte of value
Xchg AH,AL           ;
Out  DX,AL           ; Most significant byte of value
Mov  AX,833          ; Initialize counter 1 count
Mov  DX,2DCH         ; Value to obtain 300 baud:
Out  DX,AL           ; Least significant byte of value
Xchg AH,AL           ;
Out  DX,AL           ; Most significant byte of value
```

For further information about the Intel 8254 Programmable Interval Timer, Consult the Intel Refererence documentation.

Synchronous Communication

To select the transmitter and receiver clock pins on the port terminals as clock source, set bit 7 in WR 5 (Ch.B) to 1.

1) The modem cable must have a connection between pin 11 and pin 7 (ground) before the signal control logic in the Partner will recognize the X21 signals R(eceive), I(ndication), S(ignal element timing), T(ransmit) and C(ontrol).

2) The X21 input signals are obtained from the Intel 8274 signals in the following way:

   R is the Intel 8274 RxD signal
   I is the Intel 8274 CTS signal

   When the R and the I signals have been low (off) for a period of 16 contigous bit intervals the Intel 8274 signal DCD will be set to indicate either a DCE clear indication or a DCE clear confirmation.

3) The X21 output signals are obtained from the Intel 8274 signals in the following way:

   T is the Intel 8274 TxD signal
   C is the Intel 8274 DTR signal

   T is gated with the Intel 8274 RTS signal to make it possible to let the Intel 8274 send 'all zeroes' in the idle state (RTS=0 means that an 'all zeroes' bit pattern will be sent, RTS=1 means that the TxD signal will be sent).

NRZI Operation

NRZI coding is enabled by setting bit 1 in wr5 (Ch.B) to 1, program Ch.A to "*1 clk" (in case of async), and program the 8254 counters 0 and 1 to a 16 times slower clock frequency (see table above).

11.4 MF143 IEEE488 Adapter

This chapter descripes the MF143 Adapter. The MF143 contains an IEEE488 controller. The Adapter provides the user of the Partner with an interface to the IEEE488 Interface Bus.

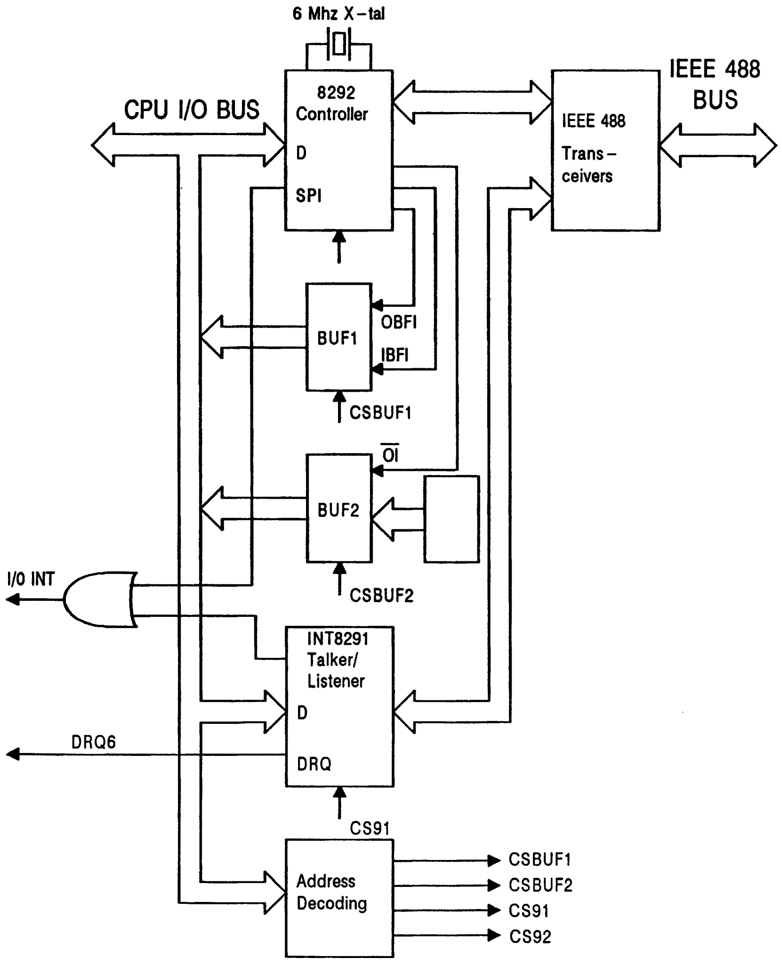
11.4.1 MF143 Driver

There is no built-in support of this Adapter in the operating system. An example driver for this Adapter is included in source form on the disk supplied in the SW1688 software package. The example driver implements a subset of the IEEE488 interface functions by means of a set of subroutines.

11.4.2 MF143 Hardware Description

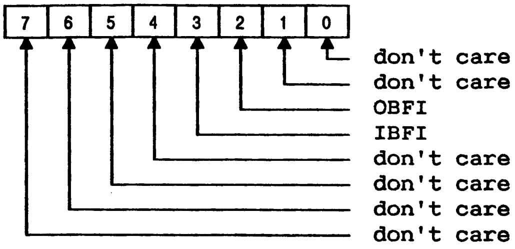
The Adapter is based on the Intel 8291 and 8292 controllers. Further information about these controllers is found in the Intel Reference Documentation.

The figure on the next page shows the architecture and the I/O addresses used on this Adapter.

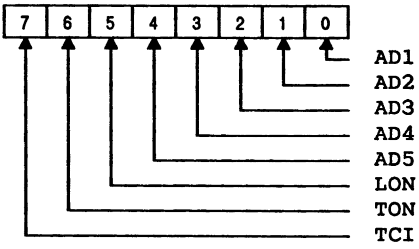


Architecture and I/O-addresses for MF143 adapter.

Through BUF1 the values of the 8298 IBFI and OBFI pins may be read:



Through BUF2 the values of the 8292 TCI pin and seven user defined jumper settings:



The jumper's are placed on the Adapter as follows:

W3		NAME
.1	.14	AD1
.2	.13	AD2
.3	.12	AD3
.4	.11	TON
.5	.10	AD4
.6	. 9	LON
.7	. 8	AD5

When a jumper is mounted, a logical low level will be read from the buffer.

The 8292 device is provided with a 6 Mhz crystal.

The 8291 is clocked by the CPU's 8 Mhz clock.

The Adapter is always the system controller as defined in the IEEE488 standard, but the programmer can release control to another controller ("controller in charge").

11.5 MF144 MODEM

This chapter is a description of an adapter that contains a MODEM and a Serial Communication Controller.

The MODEM is build around an AM7911 FSK MODEM WORLD Chip from Advanced Micro Devices.

The serial communication controller is an Intel 8251A USART.

11.5.1 MF144 Driver

The AM7911 MODEM chip allows various different standards to be used. The driver that is supplied with the operating system supports the use of V21 (300 Full Duplex) and V23 (1200/75-75/1200 baud). Both originator and answer mode are supported.

Additional logic circuitry gives the ability to let the computer:

- Dial a telephone number using either pulse trains or tones.

- Detect a call and generate an answertone

- Get an external telephone line when connected to a PABX.

The MODEM part of the MF144 is controlled by means of int-28h function 68:

```
Entry    AL = 68
         DX = Command string offset
         DS = Command string segment
Exit     AX = Error Code
         SI = Character-index for last character processed
              (= 0, 1, 2 ...)
```

Error Codes:

```
  0 ok-result
  1 modem not installed
  2 no ready-tone
  3 unstable ready-tone (disappeared within 1.2 seconds).
  4 digit expected
  5 illegal key (in number)
  6 modem mode-error
  7 no answer-tone (2100 Hz)
  8 unstable answer-tone (disappeared within 1.5 seconds)
  9 syntax error in command string
```

In case of time-out, the SIGN-BIT of the 16-bit errorcode is added.

The command-string, which controls the function of the MF144-modem, consists of a number of commands, terminated by a NULL-character. Characters less than 33 and greater than 127 are blind. A command is either a letter (lower- and upper-case letters are considered the same) or a letter followed by a string.

In the following descriptions , <i> is optional and means an integer with a command-dependent default value, <c> is a single character, <n> means a subscriber number formed by the characters: 0,1,2,3,4,5,6,7,8,9,A,B,C,D,* or #, which are keys in the telephoneset; the number-string is terminated by a 'period'(i.e. the following string is a correct numbercall command: T06250411.).

Command	Meaning
<	Loudspeaker on.
>	Loudspeaker off.
L	Connect the modem to the telephoneline.
Q	Disconnect modem from the telephoneline.
K<i>	Wait max. <i> seconds for ready-tone (default 6 seconds).
!	PABX, i.e. select city-line from PABX.
S<i>	Wait max. <i> seconds for 2100 Hz-answer-tone (default 20 sec)
M<c>	Set mode and RTS, i.e. establish dataconnection. <c>=0: v.21 Orig 300bps full duplex <c>=1: v.21 Answ 300bps full duplex <c>=2: v.23 (Xmit 1200bps, Receive 75bps) <c>=3: v.23 (Xmit 75bps, Receive 1200bps) <c>=4: v.21 loop back mode
T<n>.	Call subscriber, TONE-CALL - <n> is the number to call.
P<n>.	Call subscriber, PULSE-CALL - <n> is the number to call.
W<i>	Wait <i> seconds - (default 1 second)
R<i>	Wait max. <i> seconds for a call (default infinite).
A<c>	Connect to the telephoneline and generate answer-tone according to <c>. <c>=1: v.21 Answ 300bps full duplex <c>=2: v.23 (Xmit 1200bps, Receive 75bps) <c>=3: v.23 (Xmit 75bps, Receive 1200bps)
D	Enter Digital Loopback Mode

In case the DCD (Data Carrier) are unstable or missing for a period of 5 seconds, while the modem is connected to the telephoneline, the connection is automatically removed. The connection is not removed during the first 15 seconds after the connection is made.

Example:

```
; ASM86 example
; GENCMD example 8080
         Cseg
         Org  100h

Start:
         Mov  DX,Offset Command
         Mov  AX,68
         Int  28H
         Mov  BX,AX
         Shl  BX,1
         Mov  DX,ErrTable[BX]
         Mov  CL,9
         Int  224
         Mov  DX,Offset CrLf
         Mov  Cl,9
         Int  224
         Mov  CL,0
         Int  224

Command  Db   '<'        ; enable loudspeaker
         Db   'L'        ; connect
         Db   'K'        ; wait for ready tone
         Db   '!'        ; get external line
         Db   'K'        ; wait for ready tone
         Db   'T0055.'   ; dial 0055 (use tones)
         Db   'S'        ; wait for answer tone
         Db   'M3'       ; choose send 75/receive 1200 mode
         Db   'W5'       ; wait 5 seconds
         Db   '>'        ; disable loudspeaker
         Db   'Q'        ; disconnect line
         Db   0          ; end of command string

ErrTable Dw   Offset Error0
         Dw   Offset Error1
         Dw   Offset Error2
         Dw   Offset Error3
         Dw   Offset Error4
         Dw   Offset Error5
         Dw   Offset Error6
         Dw   Offset Error7
         Dw   Offset Error8
         Dw   Offset Error9
```

```
Error0   Db   'No Error$'
Error1   Db   'MF144 Not Installed$'
Error2   Db   'Ready Tone Not Detected$'
Error3   Db   'Ready Tone Not Stable$'
Error4   Db   'Number Expected$'
Error5   Db   'Illegal Phone Number$'
Error6   Db   'Illegal MODEM Configuration Code$'
Error7   Db   'Answer Tone Not Detected$'
Error8   Db   'Answer Tone Not Stable$'
Error9   Db   'Unknown Command$'

CrLf     Db   13,10,'$'

         End
```

The Serial Communication Controller on the MF144 is supported as an virtual console (#8) or as an auxiliary device (#4) and the corresponding operating system calls are used to access the device. The Controller is initialized during power-up according to parameters kept in the NVM. These parameters are manipulated using the KONFIG program. In addition to this pre-configuration, it is possible to change the parameters dynamically be means of the int-28h function 54:

```
Entry    AX = 54
         Stack + 2 = Parameter Block Segment
         Stack + 0 = Parameter Block Offset

Parameter Block:
         Byte 0: Virtual Console No. (8)
         Byte 1: Mode (0:Console,1:Printer,2:Satellite)
         Byte 2: Protocol (0:None, 1:XonXoff, 2: Satellite)
         Byte 3: Not Used
         Byte 4: Not Used
         Byte 5: Bits per character for send and receive
         Byte 6: Not Used
         Byte 7: Stop Bits (0:1, 1:1.5, 2:2)
         Byte 8: Parity (0:Odd, 1:Even, 2:No)
         Byte 9: Not Used
```

The MF144 can be reset to the state of the pre-configuration by means of int-28h function 50:

```
Entry    AL = 50
         DL = Virtual Console (=8)
```

The MODEM signals from the MF144 can be obtained by means of the int-28h function 55:

```
Entry    AL = 55
         DL = Virtual Console (=8)
Exit     AH = 8251 Status Register
         AL = DTR*128 + RTS*64 + DSR*8 + CI*4 + DCD*2 + CTS
```

Driver status information concerning the MF144 can be obtained by means of int-28h function 65:

```
Entry    AL = 65
         CL = 0 (Receiver) or 1 (Transmitter)
         DL = Virtual Console (=8)
Exit     AL = State
         AH = Protocol
         BX = Total Buffer Length
         CX = Remaining Buffer Space
         DH = Configuration
```

11.5.2 MF144 Hardware Description

The MF144 is build from various controllers and registers. This chapter describes these controllers and registers, their addresses and their use.

Intel 8251A USART

The 8251A is programmed to operate in asynchronous mode and with clock division 64 (x64).

The Intel8251A Serial Communication Controller uses the following addresses:

Address	Register
282H	Status read
28AH	Control write
280H	Data read
288H	Data write

Intel 8254 Programmable Interval Timer

The Baud Rate Clock is derived from the CPU clock using an Intel 8254 Counter. Counter 1 generates the Receive Clock while Counter 0 generates the Transmit Clock. Both counters are programmed to operate in mode 3. The following table defines counter values for different baudrates and CPU clock frequency.

Baudrate	6Mhz CPU	8Mhz CPU
1200	39	52
600	78	104
300	156	208
150	313	417
75	625	833

Counter 2 is used for generation of timer interrupts. This counter is programmed to operate in mode 0. When the counter output level change to 'high' an interrupt will be generated. The interrupt source is determined by reading control register 2 (see below). The interrupt is cleared by loading the counter register followed by setting the TIMER GATE in control register 3 to "0" (see below).

The following table describes the relation between counter values, timer periods and CPU clock frequency.

Counter Value	6Mhz CPU	8Mhz CPU
1	666,67 nS	500 nS
.	.	.
.	.	.
65536	43,69 mS	32,77 mS

The following addresses are assigned to the Intel 8254:

Address	Register
290H	Read counter #0
298H	Write counter #0
292H	Read counter #1
29AH	Write counter #1
294H	Read counter #2
29CH	Write counter #2
29EH	Write control word

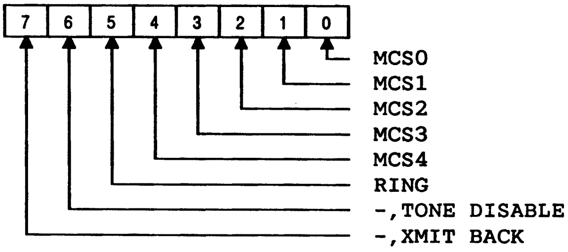
Registers

The MODEM function is controlled by means of a number of registers.

Control Register 0 (MODEM Control):

Address: 2A8H

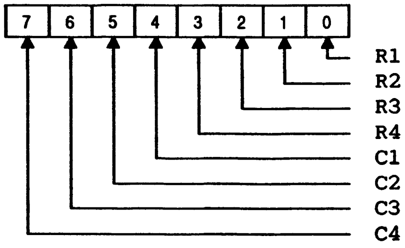
Encoding:



Control Register 1 (Tone Encoding):

Address: 2B8H

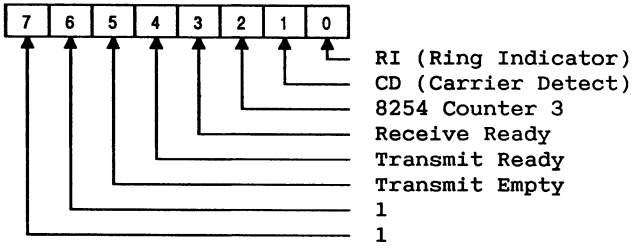
Encoding:



Control Register 2:

Address: 2A0H

Encoding:



The MF144 is connected to interrupt level 7 on the External Interrupt Controller. Five different conditions can give rise to an interrupt from the MF144 MODEM Controller:

1) The Serial Controller has received a character.

2) The Serial Controller has sent a character.

3) The State of the CD (Carrier Detect) Signal has changed.

4) The State of the RI (Ring Indicator) Signal has changed to active.

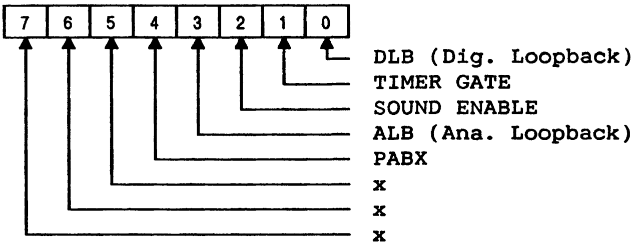
5) Intel 8254 Counter 2 is counted down to 0.

The contents of control register 2 indicates which of the five interrupt conditions that has occured.

Control Register 3:

Address: 2C8H

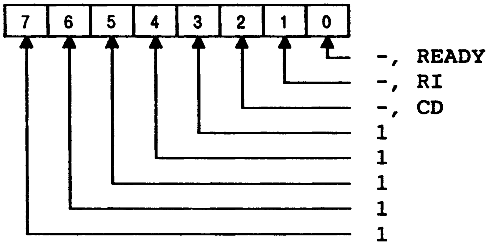
Encoding:



Control Register 4:

Address: 2C0H

Encoding:



Example of use:

1) Initialize 8254 to generate appropriate baudrates.

2) Set DTR bit in 8251A to enable Am7911 Modem Chip and to activate hook switch.

3) Await ready tone from network by polling READY bit in control register 3 ("0" means ready tone present).

4) If connected to a PABX it may be necessary to set the PABX bit in control register 3, to obtain a public line. How long time the PABX bit should be high ("1") depends on the actual PABX. When a public line is obtained an additional ready tone can be detected as described in 3).

5) Dial number. This can be accomplished in two ways: By generating pulses using the hook switch or by generating tones using the tone dialer chip.

   To dial the number "2" using pulses is done in the following way:

```
Set DTR="0"
 Wait T1

 Set DTR="1"
 Wait T2

 Set DTR="0"
 Wait T1

 Set DTR="1"
 Wait T3

 Dial Next Number
```

   T1, T2 and T3 must comply with the requirements of the telephone network.
   To dial the number "6" using tones is done in the following way:

   The R1-R4 and C1-C4 bits in control register 1 are set according to the following table.

1	2	3	A	R1
4	5	6	B	R2
7	8	9	C	R3
*	0	#	D	R4
C1	C2	C3	C4	

   The R and C cooresponding to the actual number should be set to "0", e.g. the number "6" is obtained using the following byte to initialize control register 1:

   10111101B

   When the register has been initialized the tone dialer must be enabled. This is done by setting the TONE DISABLE bit in control register 0 to "1".

6) The called Modem will answer the call by generating an answer tone. If the answer tone has the same frequency as the "space" condition, it could be detected as the

presence of the carrier detect signal. The "space" condition frequency varies depending on the transmission mode used. To get an "space" condition frequency of 2100 Hz (i.e. the CCIT answer tone frequency) the MCS4-1 (Am7911 data sheet page 4) in control register 0 is set to:

MCS4	MCS3	MCS2	MCS1	MCS0
0	0	1	1	1

corresponding to CCIT V.23 mode. The carrier detect signal is reflected in the -,CD bit in control register 4 ("0" means carrier present).

7) After the answer tone is detected, the Am7911 should be initialized to the relevant transmission mode (MCS4-1, refer to Am7911 data sheet page 4) and the RTS signal in 8251A should be asserted to make the Am7911 generate a carrier signal.

8) Now the connection is established and normal data transfer can take place by reading/writing data from/to the 8251A USART.

9) The line is disconnected when the DTR bit in 8251A is set to "0".

12. 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 an RC750 and one of the following computers:

1) Another RC750
2) An RC702
3) An RC703
4) An RC855 Workstation

Together with the FILEX source program included on the RC750 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).

12.1 Requirements

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 and RC750, 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.

12.2 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.

12.2.1 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

12.2.2 Transmission protocol

The transactions described in 8.2.1 are sent by means of the blocked transmission 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

The Int-28h Functions are called by means of a software interrupt on level 28h. Before issuing the interrupt the function number must be loaded in AL. Other parameters are loaded in the appropriate register or pushed on the stack as described in the following. All registers except DS may be changed in an Int-28h function call.

Function 0: **Changes the console mode to graphics mode.**

Entry	AL = 0
	AH = 1=high resolution/2=medium resolution
	DX = Address segment of graphics control block.
	CX = Address offset of graphics control block.

See 4.5.1.

Function 1: **Changes the console mode to character mode.**

Entry AL = 1

See 4.5.2.

Function 2: Scrolling

Entry AL = 2

This Function is used by SCROLL.RSP when soft-scrolling is in progress. This function cannot be used for other purposes.

Function 3: **Return the address of a copy of the nonvolatile memory contents.**

Entry AL = 3

Exit	ES = Address segment
	SI = Address offset

See 3.2.

Function 4: **Return the address of a configuration description.**

Entry	AL	= 4
Exit	ES	= Address segment
	SI	= Address offset

See 3.1.

Function 5: **Recalibrate floppy disk drive.**

Entry	AL	= ~~4~~ 5
	SP+10	= Drive (0/1)
	SP+ 8	= Head (0/1)
	SP+ 6	= Cylinder
	SP+ 4	= Bytecount
	SP+ 2	= DMA segment
	SP+ 0	= DMA offset
Exit	AL	= Floppy disk controller status register

Function 6: **NOT USED**

Function 7: **NOT USED**

Function 8: **Step floppy drive head one track in.**

Entry	AL	= 8
	SP+10	= Drive (0/1)
	SP+ 8	= Head (0/1)
	SP+ 6	= Cylinder
	SP+ 4	= Bytecount
	SP+ 2	= DMA segment
	SP+ 0	= DMA offset
Exit	AL	= Floppy disk controller status register

Function 9: **Step floppy drive head one track out.**

Entry	AL	= 9
	SP+10	= Drive (0/1)
	SP+ 8	= Head (0/1)
	SP+ 6	= Cylinder
	SP+ 4	= Bytecount
	SP+ 2	= DMA segment
	SP+ 0	= DMA offset
Exit	AL	= Floppy disk controller status register

Function 10: **Write a track to floppy disk.**

Entry	AL	= ~~8~~ 10
	SP+10	= Drive (0/1)
	SP+ 8	= Head (0/1)
	SP+ 6	= Cylinder
	SP+ 4	= Bytecount
	SP+ 2	= DMA segment
	SP+ 0	= DMA offset
Exit	AL	= Floppy disk controller status register

Function 11: **Read a track from floppy disk.**

Entry	AL	= ~~7~~ 11
	SP+10	= Drive (0/1)
	SP+ 8	= Head (0/1)
	SP+ 6	= Cylinder
	SP+ 4	= Bytecount
	SP+ 2	= DMA segment
	SP+ 0	= DMA offset
Exit	AL	= Floppy disk controller status register

Function 12: **Write a byte to the sound generator.**

Entry	AL = 12
	DL = byte

Function 13: Get address of disk driver statistics

```
Entry    AL = 13

Exit     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

; Winchester disk controller statistics. Count fields for
; each ERROR CLASS and for each ERROR TYPE. Consult the
; winchester disk controller manufacturers manual for
; details.

; Drive A:
WD_Error_0     RB 16    ; CLASS 0 TYPE 0-15
WD_Error_1     RB 16    ; CLASS 1 TYPE 0-15
WD_Error_2     RB 16    ; CLASS 2 TYPE 0-15
WD_Error_3     RB 16    ; CLASS 3 TYPE 0-15
WD_Error_4     RB 16    ; CLASS 4 TYPE 0-15
WD_Error_5     RB 16    ; CLASS 5 TYPE 0-15
WD_Error_6     RB 16    ; CLASS 6 TYPE 0-15

; Drive B-P:
               RS 15*16*7
```

Function 14: NOT USED

Function 15: Get PFK table

```
Entry    AL = 15

Exit     ES = Address Segment of PFK table
         BX = Address Offset of PFK table
```

PFK table Format:

    Content of first 20 character PFK.
    .
    .
    Contents of last 20 character PFK.
    Content of first 4 character PFK.
    .
    .
    Contents of last 4 character PFK.

See also Function 57: Get PFK Information.

Function 16: **SCSI bus command interface.**

```
Entry   AL    = 16
        SP+10 = Drive*256 + (1=input,0=output)
        SP+ 8 = Command description block segment
        SP+ 6 = Command description block offset
        SP+ 4 = DMA byte count
        SP+ 2 = DMA segment
        SP+ 0 = DMA offset

Exit    AL    = status byte
        AH    = if not zero then a bus phase error occured
                during the command and the contents of AL
                are useless.
        SP+10 = status byte
```

See 8.5.

Function 17: **Get SCSI controller select byte and logical unit number for a drive.**

```
Entry   AL   = 17
        SP+0 = Drive

Exit    AL   = SCSI controller select byte
        AH   = logical unit number
```

NOTE: AX = 0 if drive is not connected to SCSI bus.

See 8.5.

Function 18: **Set Serial Communication Protocol**

```
Entry   CL = Channel No. (0: COMM/V24, 1: RS232)
        DL = Protocol (0:None,1:XonXoff,2:Satellite)

Exit    Protocol Changed and Buffers Reset
```

Function 19: **Return 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).

Entry	AL = 19
Exit	DX = Second count high
	AX = Second count low
	CX = Elapsed 16 mS periods of next second.

NOTE: In case a 73 Hz refresh rate screen is used, all appearances of '16 mS' should be substituted by '13.7 mS'.

Function 20: **Define a character in the alternative character set.**

Entry	AL = 20
	CL = Character number (0-255)
	DS = Address segment of character definition block
	DX = Address offset of character definition block

See 4.3.2.

Function 21: **Return a pointer to a console display list.**

Entry	AL = 21
Exit	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: **Return the current cursor position.**

Entry AL = 22

Exit BH ~~DH~~ = Row
BL ~~DL~~ = Column
BX = DX

See 4.2.4.

Function 23: **Get register 0 and 1 of the Intel 8274.**

Entry AL = 23
CX = channel (0 or 1)

Exit AL = Intel 8274 register 0
AH = Intel 8274 register 1

Function 24: **Initialize the Intel 8274 controller.**

Entry AL = 24
SP+2 = Parameter block segment
SP+0 = Parameter block offset

See 7.2.4.

Function 25: **Bypass XonXoff protocol**

Entry AL = 25
DL = Virtual Console No. (4-8)
CL = Character

The character in CL is sent even if a XOFF (DC3) character has been received.

Function 26-29: PICCOLINE only

Function 30:

Subfunction 0: Set mouse vector.

```
Entry    AL = 30
         CL = 0
         SI = Address offset of interrupt routine
         DX = Address segment of interrupt routine
```

Subfunction 1: Initializes mouse.

```
Entry    AL = 30
         CL = 1
```

Subfunction 2: Deinitializes mouse.

```
Entry    AL = 30
         CL = 2
```

Subfunction 3: Returns the current status of the mouse.

```
Entry    AL = 30
         CL = 3

Exit     When nothing has happened:
         AL = 0

         When Button has been depressed:
         AL = 1
         AH = Character information.

         When mouse has been moved:
         AL = 2
         BX = Delta x
         CX = Delta y
```

See 4.9.

Function 31: Defines palette contents.

```
Entry    AL = 31
         DS = Address segment of palette definition
         DX = Address offset of palette definition
```

See 4.1.3.

Function 32: Set Timer Address

```
Entry   AL = 32
        CX = Address segment of timer routine
        DX = Address offset of timer routine
```

Define timer routine.

The timer routine will be invoked each 16 mS (13.7 mS if a 73 Hz refresh rate screen is used) as part of the CRT interrupt service routine.

This function is used by the netdriver and should not be used for other purposes.

Function 33: Clear Timer Address

```
Entry   AL = 33
```

Passivate timer routine call. This function should only be used by the netdriver.

Function 34: Set Net Reset

```
Entry   AL = 34
```

Define address of routine to be called to reset net controller before warm-boot is performed.

Function 35: Write a string direct to the console buffer.

```
Entry   AL = 35
        DL = Column
        DH = Row
        CX = Count
        DS = Address segment of string
        SI = Address offset of string
```

See 4.2.2.

Function 36: Set cursor position.

```
Entry     DH = Row
          DL = Column
          AL = 36
```

See 4.2.4.

Function 37: Returns current attributes.

```
Entry     AL = 37
Exit      AH = Current attributes
```

See 4.2.5.

Function 38: Set attributes.

```
Entry     AL = 38
          AH = Attributes
```

See 4.2.5.

Function 39: Update physical screen.

```
Entry     AL   39
```

See 4.2.3

Function 40-49: PICCOLINE only

Function 50: Reset SIO channel

```
Entry     AL = 50
          DL = SIO channel
```

See 7.2.4

Function 51: **Get font.**

Returns a character from the character font.

```
Entry    AL = 51
         CX = Character number (0-1023)
         DS = Segment of character definition block
         DX = Offset of character definition block
```

See 4.3.4.

Function 52: **Define font.**

Defines a character in the character font.

```
Entry    AL = 52
         CX = Character number (0-1023)
         DS = Segment of character definition block
         DX = Offset of character definition block
```

See 4.3.3.

Function 53: **Get XIOS version**

```
Entry    AL = 53

Exit     AH = Year (BCD)
         AL = Version number (BCD)
         BH = Month (BCD)
         BL = Day (BCD)
```

Function 54: **Initialize Serial Controller**

```
Entry    AL = 54
         Stack + 2 = Parameter Block Segment
         Stack + 0 = Parameter Block Offset

Parameter Block:
    Byte 0: Virtual Console No. (4-8)
    Byte 1: Mode
            0=Console
            1=Printer
            2=Satellite
    Byte 2: Protocol
            0=None
            1=XonXoff
            2=Satellite
    Byte 3: Receiver Baud Rate
            0=50
            1=75
            2=110
            3=150
            4=300
            5=600
            6=1200
            7=2400
            8=4800
            9=9600
           10=19200
    Byte 4: Transmitter Baud Rate
            0=50
            1=75
            2=110
            3=150
            4=300
            5=600
            6=1200
            7=2400
            8=4800
            9=9600
           10=19200
    Byte 5: Bits per Character for Receiver
            5-8
    Byte 6: Bits per Character for Transmitter
            5-8
    Byte 7: Stop Bits
            0=1
            1=1.5
            2=2
    Byte 8: Parity
            0=odd
            1=even
            2=no
    Byte 9: DTR and RTS
            00H=DTR ON and RTS OFF
            01H=DTR ON and RTS ON
            10H=DTR OFF and RTS OFF
            11H=DTR OFF and RTS ON
```

NOTE: If Mode is Satellite then Protocol must be Satellite too.

Function 55: Get Serial Controller Status

Entry	AL = 55
	DL = Virtual Console No. (4-8)
Exit	AH = Serial Controller Status Register
	AL = Bit 8: DTR
	Bit 7: RTS
	Bit 6: 0
	Bit 5: 0
	Bit 4: DSR
	Bit 3: CI
	Bit 2: DCD
	Bit 1: CTS

Function 56: Get Screen Information

Entry	AL = 56
Exit	AL = Physical console number if the process is owner of a virtual console.
	AH = 0: Monochrome Display, 1: Colour Display.

Function 57: Get Function Key Information

Entry	AL = 57
	CL = Programming value (i.e. value after <esc><:>).
Exit	CL = Max. length (0 if unknown function key).
	ES = Address Segment of key contents.
	SI = Address Offset of key contents.
	DI = Internal Function Key Number

Function 58: Execute Soft Reset

Entry	AL = 58

Function 59: Change Escape Codes

Entry	AL = 59
	CL = Current Escape Character
	CH = New Escape Character
NOTE:	New Escape Codes are used in all consoles. This function can't be used in connection with satellites.

Function 60: Set Screen-off Timer

Entry	AL = 60
	CX = Time in seconds. 0 means disable screen-off.
NOTE:	This function can't be used in connection with satellites.

Function 61: Set Text Mode Fill character

Entry	AL = 61
	CL = Fill Character Value.
NOTE:	Fill character are only changed in the actual console. This function can't be used in connection with satellites.

Function 62: Get Status Line Address

Exit	AL = 62
	CX = Segment of Displayed Status Line
	DX = Offset of Displayed Status Line

Function 63: Simulate Ascii Character Input

Entry	AL = 63
	DL = Ascii Key Value
	DH = 0
NOTE:	If DH contains 255 then DL is interpreted as a console number and this console is set to foreground console.

Function 64: Simulate Scan Code Input

```
Entry    AL = 64
         CL = Keyboard Scan Code

NOTE:    This function can't be used in connection with
         satellites.
```

Function 65: Get SIO Information

```
Entry    AL = 65
         CL = 0 (Receiver) or 1 (Transmitter)
         DL = Virtual console number (4-7)
Exit     AL = State
         AH = Protocol
              0: none
              1: Xon-Xoff
              2: Satellite
         BX = Buffer Length
         CX = Buffer Space (Remaining)
         DH = Configuration
              0: Console
              1: Printer
              2: Satellite
              3: Aux
```

Function 66: Get Palette

```
Entry    AL = 66
         DS = Address Segment of Palette Definition Area
         DX = Address Offset of Palette Definition Area

Exit     Paletter Definition Area Filled In

NOTE:    This function can't be used in connection with
         satellites.
```

Function 67: Get Net Buffer Information

Entry	AL = 67
Exit	ES = NET buffer segment
	BX = NET buffer offset
	CX = NET buffer count
	AX = NET interrupt vector

For netdriver use only.

Function 68: Modem Control

Entry	AL = 68
	DX = Command string offset
	DS = Command string segment
Exit	AX = Error Code
	SI = Character-index for last character processed (= 0, 1, 2 ...)

Error Codes:

0	ok-result
1	modem not installed
2	no ready-tone
3	unstable ready-tone (disappeared within 1.2 seconds).
4	digit expected
5	illegal key (in number)
6	modem mode-error
7	no answer-tone (2100 Hz)
8	unstable answer-tone (disappeared within 1.5 seconds)
9	syntax error in command string

In case of time-out, the SIGN-BIT of the 16-bits errorcode is added

See 11.5.1

Function 69: Display Message

```
Entry    AL = 69
         DI = Physical Console No. (0,1 or 2)
         DX = Address Segment of Message
         CX = Address Offset of Message

Exit     AL = Character from keyboard
```

Message is a string of exactly 80 characters. The message is displayed in the statusline on the specified console (either the main console or one of the satellites). The message remains in the statusline until a key is entered on the console keyboard. The key entered is returned in AL.

Function 70: Set Modem Signals

```
Entry    AL = 70
         DL = Virtual Console No.
         CL = Bit 0 Set RTS ON
              Bit 1 Set DTR ON
              Bit 2 Set BREAK ON
              Bit 3 Set RTS OFF
              Bit 4 Set DTR OFF
              Bit 5 Set BREAK OFF

Exit     Modem Signals Changed
```

Function 71: Set PC Character set

```
Entry    AL = 71
         CX = 1 Set 8-bit pc-mode
              0 Set 7-bit pc-mode
```

This mode setting only affects MS-DOS programs.
In 8-bit pc-mode the keyboard driver converts the national characters to support IBM 8-bit character set. In this mode the alternative characterset is used as the default character set. The CHAR8 program initializes the alternative character font with an IBM PC compatible character set and sets the 8-bit pc-mode. DOS interrupt 17H and DOS interrupt 21H, Function 5 (printer output) converts the national characters properly in this mode.

Function 72: **Get Console Mode**

```
Entry    AL = 72

Exit     AL = mode (bit 0: 7/8 bit mode, bit 1: PC mode)
```

Function 73: **Convert PC characters**

```
Entry    AL = 73
         DL = Character to convert

Exit     DL = Converted Character
```

Converts the national characters to support IBM 8-bit character. This will only affect MS-DOS programs running in 8-bit pc-mode. This Function is used by DOS interrupt 17H and DOS interrupt 21H, Function 5 (printer output) to converts the national characters properly.

Function 74: **Assign Spool Printer**

```
Entry    AL = 74
         DL = Printer Number
         CL = 0   (Release Spool Printer)
            = FFH (Assign Spool Printer)
```

This Function Changes the behaviour of the XIOS Printer Status Function.

When a Printer is Assigned as a Spool Printer The XIOS Printer Status Function Returns the following:

```
    AX = 090FFH ( Cheat Return Meaning: Ready )
    BX = 09FFH
    CX = 090FFH ( Real Return If Ready )
       = 01000H ( Real Return If Not Ready )
```

This Function is used by the Spool System to cheat programs that investigate the printer status before using it (printer status always returns ready in this mode).

Function 75: **Return CPU type**

Entry	AL = 75
Exit	AL = 6 if 6Mhz CPU, 8 if 8Mhz CPU.

B. Peripheral Device I/O Addresses.

Address	Peripheral	I/O Direc- tion	In- ter- rupt	DMA re- quest
0000H	I8259 PIT Initialization.			
0002H	I8259 PIT Operation.			
0010H	SCSI data register.	I/O	2	0
0020H	Keyboard	I	1	
0030H	I8274 channel A command	I/O	INT1	1
0032H	i8274 channel A data	I/O	INT1	1
0034H	I8274 channel B command	I/O	INT1	
0036H	I8274 channel B data	I/O	INT1	
0040H	I8254	I/O		
0050H	Sound+RTC	I/O		
0070H	NVM + DMA select	O		
0072H	SCSI + Comm	I		
0074H	SCSI + control	O		
0076H	Control 70-74H	O		
0080H-FEH	NVM	I/O		
0100H	I82586 channel attention	O		
0180H-19FH	Palette	O		
0200H	WD1797 control	I/O	0	5
0202H	Track register			
0204H	Sector register			
0206H	Data register			
0210H	WD1797 control	O		
0220H	Internal switch setting	I		
0230H	I82730 reset interrrupt	O		
0240H	I82730 channel attention	O		4

C. Interrupt vector assignment

CPU Interrupts:

Source	Number	Vector
Divide error exception	0	0000:0000H
Single step interrupt	1	0000:0004H
Non maskable interrupt	2	0000:0008H
Breakpoint interrupt	3	0000:000CH
INT 0 detected	4	0000:0010H
Array bounds exception	5	0000:0014H
Unused opcode exception	6	0000:0018H
ESC opcode exception	7	0000:001CH
Timer 0 interrupt	8	0000:0020H
DMA 0 interrupt	10	0000:0028H
DMA 1 interrupt	11	0000:002CH
INT 0 interrupt	12	0000:0030H
INT 1 interrupt	13	0000:0034H
INT 2 interrupt	14	0000:0038H
INT 3 interrupt	15	0000:003CH
Timer 1 interrupt	16	0000:0040H
Timer 2 interrupt	17	0000:0044H

SIO (8274) Interrupts (Connected via INT1/INTA1 PIN's):

Source	Number	Vector
Channel B transmitter	0	0000:0100H
Channel B status	1	0000:0104H
Channel B receiver	2	0000:0108H
Channel B special receive	3	0000:010CH
Channel A transmitter	4	0000:0110H
Channel A status	5	0000:0114H
Channel A receiver	6	0000:0118H
Channel A special receive	7	0000:011CH

PIC (8259) Interrupts (Connected via INT0/INTA0 PIN's):

Source	Number	Vector
Floppy controller	0	0000:0120H
Keyboard interface	1	0000:0124H
SCSI interface	2	0000:0128H
Real time clock	3	0000:012CH
CRT	4	0000:0130H
Net controller	5	0000:0134H
Parallel interface	6	0000:0138H
I/O Adapter	7	0000:013CH

Software Generated Interrupts:

IBM ROM BIOS	Number	Vector
VIDEO_I/O (1)	10H	0000:0040H
EQUIPMENT (2)	11H	0000:0044H
MEMORY_SIZE_DETERMINE	12H	0000:0048H
DISKETTE_IO (2)	13H	0000:004CH
KEYBOARD_IO	16H	0000:0058H
PRINTER_IO	17H	0000:005CH

(1) Subfunctions 11,12,13,14 and 15 NOT SUPPORTED
(2) NOT SUPPORTED

DOS 2.0	Number	Vector
PROGRAM TERMINATE	20H	0000:0080H
INVOKE DOS SYSTEM CALL	21H	0000:0084H
TERMINATE ADDRESS	22H	0000:0088H
CTRL-BREAK ADDRESS	23H	0000:008CH
CRITICAL ERROR EXIT ADDRESS	24H	0000:0090H
ABSOLUTE DISK READ (2)	25H	0000:0094H
ABSOLUTE DISK WRITE (2)	26H	0000:0098H
TERMINATE BUT STAY RESIDENT (2)	27H	0000:009CH

(2) NOT SUPPORTED

RC specific	Number	Vector
Int-28h function call	28H	0000:00A0H
Net driver	29H	0000:00A4H
IMC	30H	0000:00A8H

D. Character Set and Keystrokes

This appendix shows the character sets supported by Partner. The first character set is the standard Partner character set. The second is the IBM PC compatible character set, initialized by the CHAR8 program.

The Partner character set shown is the danish character set. The differences between the danish character set and the national variants are shown in the table below.

	35	64	91	92	93	94	96	123	124	125	126
Danish	§	@	Æ	Ø	Å	Ü	‘	æ	ø	å	ü
US ASCII	#	@	[	\	]	^	‘	{	¦	}	~
UK ASCII	£	@	[	\	]	^	‘	{	¦	}	~
German	#	§	Ä	Ö	Ü	^	‘	ä	ö	ü	ß
Swedish	#	É	Ä	Ö	Å	Ü	é	ä	ö	å	ü
Norwegian	§	@	Æ	Ø	Å	Ü	‘	æ	ø	å	ü
Finnish	#	É	Ä	Ö	Å	Ü	é	ä	ö	å	ü

VALUE			
DEC	HEX	KEYSTROKES	
0	0	CTRL@	
1	1	CTRL A	É
2	2	CTRL B	Ä
3	3	CTRL C	Ö
4	4	CTRL D	é
5	5	CTRL E	ä
6	6	CTRL F	ö
7	7	CTRL G	□
8	8	⬅	ß
9	9	⇥	£
10	A	CTRL J	¤
11	B	CTRL K	Æ
12	C	⇤	Ø
13	D	↵	$
14	E	CTRL N	Å
15	F	CTRL O	Ü
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	CTRLW	{
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

VALUE			
DEC	HEX	KEYSTROKES	
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			
DEC	HEX	KEYSTROKES	
160	A0	ALT SPACE	
161	A1	SHIFT ALT !	🬀
162	A2	SHIFT ALT "	🬁
163	A3	SHIFT ALT §	🬂
164	A4	SHIFT ALT $	🬃
165	A5	SHIFT ALT %	🬄
166	A6	SHIFT ALT &	🬅
167	A7	SHIFT ALT '	🬆
168	A8	SHIFT ALT (	🬇
169	A9	SHIFT ALT)	🬈
170	AA	SHIFT ALT *	🬉
171	AB	SHIFT ALT +	🬊
172	AC	ALT ,	🬋
173	AD	ALT –	🬌
174	AE	ALT .	🬍
175	AF	ALT /	🬎

VALUE			
DEC	HEX	KEYSTROKES	
176	B0	ALT 0	🬏
177	B1	ALT 1	🬐
178	B2	ALT 2	🬑
179	B3	ALT 3	🬒
180	B4	ALT 4	🬓
181	B5	ALT 5	▌
182	B6	ALT 6	🬔
183	B7	ALT 7	🬕
184	B8	ALT 8	🬖
185	B9	ALT 9	🬗
186	BA	ALT :	🬘
187	BB	ALT ;	🬙
188	BC	SHIFT <	🬚
189	BD	SHIFT =	🬛
190	BE	SHIFT 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	🬬
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			
DEC	HEX	KEYSTROKES	
224	E0	ALT ‘	α
225	E1	ALT A	β
226	E2	ALT B	γ
227	E3	ALT C	δ
228	E4	ALT D	ε
229	E5	ALT E	ζ
230	E6	ALT F	η
231	E7	ALT G	θ
232	E8	ALT H	ι
233	E9	ALT I	κ
234	EA	ALT J	λ
235	EB	ALT K	μ
236	EC	ALT L	ν
237	ED	ALT M	ξ
238	EE	ALT N	ο
239	EF	ALT O	π
240	F0	ALT P	ρ
241	F1	ALT Q	σ
242	F2	ALT R	τ
243	F3	ALT S	υ
244	F4	ALT T	φ
245	F5	ALT U	χ
246	F6	ALT V	ψ
247	F7	ALT W	ω
248	F8	ALT X	Δ
249	F9	ALT Y	Γ
250	FA	ALT Z	Σ
251	FB	ALT Æ	Λ
252	FC	ALT Ø	Ω
253	FD	ALT Å	#
254	FE	ALT Ü	¨
255	FF	impossible	rc

VALUE			
DEC	HEX	KEYSTROKES	
0	0	CTRL@	
1	1	CTRL A	☺
2	2	CTRL B	☻
3	3	CTRL C	♥
4	4	CTRL D	♦
5	5	CTRL E	♣
6	6	CTRL F	♠
7	7	CTRL G	•
8	8	←	◘
9	9	⇥	○
10	A	CTRL J	◙
11	B	CTRL K	♂
12	C	CTRL L	♀
13	D	↵	♪
14	E	CTRL N	♫
15	F	CTRL O	☼
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	CTRLW	↨
24	18	CTRL X	↑
25	19	CTRL Y	↓
26	1A	CTRL Z	→
27	1B	ESC	←
28	1C	CTRL Ø	∟
29	1D	CTRL Å	↔
30	1E	CTRL Ü	▲
31	1F	CTRL _	▼

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	ALT Æ	[
92	5C	ALT Ø	\
93	5D	ALT Å	]
94	5E	ALT 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

VALUE			
DEC	HEX	KEYSTROKES	
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	SHIFT ALT Æ	{
124	7C	SHIFT ALT Ø	¦
125	7D	SHIFT ALT Å	}
126	7E	SHIFT ALT Ü	~
127	7F	CTRL ←	⌂

VALUE			
DEC	HEX	KEYSTROKES	
128	80	ALT 128	Ç
129	81	ü	ü
130	82	ALT 130	é
131	83	ALT 131	â
132	84	ALT 132	ä
133	85	ALT 133	à
134	86	Å	å
135	87	ALT 135	ç
136	88	ALT 136	ê
137	89	ALT 137	ë
138	8A	ALT 138	è
139	8B	ALT 139	ï
140	8C	ALT 140	î
141	8D	ALT 141	ì
142	8E	ALT 142	Ä
143	8F	SHIFT Å	Å

VALUE			
DEC	HEX	KEYSTROKES	
144	90	ALT 144	É
145	91	Æ	æ
146	92	SHIFT Æ	Æ
147	93	ALT 147	ô
148	94	ALT 148	ö
149	95	ALT 149	ò
150	96	ALT 150	û
151	97	ALT 151	ù
152	98	ALT 152	ÿ
153	99	ALT 153	Ö
154	9A	SHIFT Ü	Ü
155	9B	Ø	ø
156	9C	ALT 156	£
157	9D	SHIFT Ø	Ø
158	9E	ALT 158	₧
159	9F	ALT 159	ƒ

VALUE			
DEC	HEX	KEYSTROKES	
160	A0	ALT 160	á
161	A1	ALT 161	í
162	A2	ALT 162	ó
163	A3	ALT 163	ú
164	A4	ALT 164	ñ
165	A5	ALT 165	Ñ
166	A6	ALT 166	ª
167	A7	ALT 167	º
168	A8	ALT 168	¿
169	A9	ALT 169	⌐
170	AA	ALT 170	¬
171	AB	ALT 171	½
172	AC	ALT 172	¼
173	AD	ALT 173	¡
174	AE	ALT 174	«
175	AF	ALT 175	»

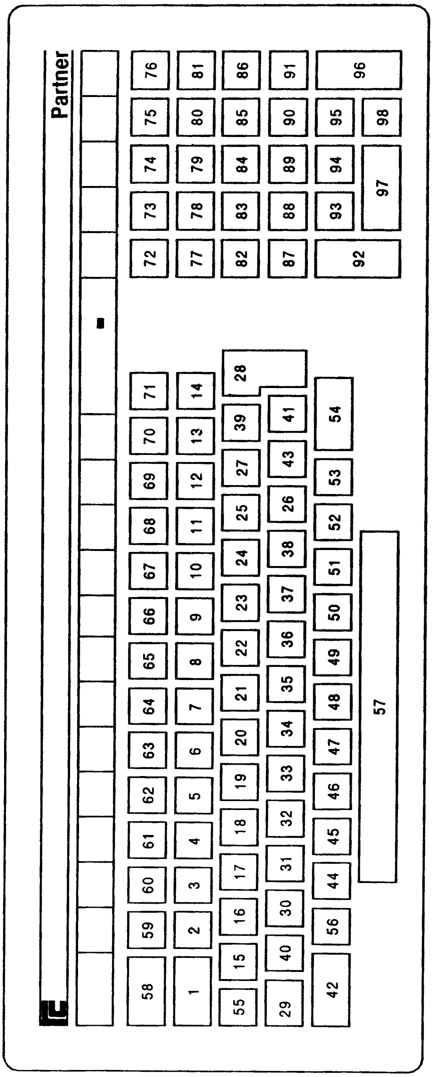
VALUE			
DEC	HEX	KEYSTROKES	
176	B0	ALT 176	░
177	B1	ALT 177	▒
178	B2	ALT 178	▓
179	B3	ALT 179	│
180	B4	ALT 180	┤
181	B5	ALT 181	╡
182	B6	ALT 182	╢
183	B7	ALT 183	╖
184	B8	ALT 184	╕
185	B9	ALT 185	╣
186	BA	ALT 186	║
187	BB	ALT 187	╗
188	BC	ALT 188	╝
189	BD	ALT 189	╜
190	BE	ALT 190	╛
191	BF	ALT 191	┐

VALUE			
DEC	HEX	KEYSTROKES	
192	C0	ALT 192	└
193	C1	ALT 193	┴
194	C2	ALT 194	┬
195	C3	ALT 195	├
196	C4	ALT 196	─
197	C5	ALT 197	┼
198	C6	ALT 198	╞
199	C7	ALT 199	╟
200	C8	ALT 200	╚
201	C9	ALT 201	╔
202	CA	ALT 202	╩
203	CB	ALT 203	╦
204	CC	ALT 204	╠
205	CD	ALT 205	═
206	CE	ALT 206	╬
207	CF	ALT 207	╧
208	D0	ALT 208	╨
209	D1	ALT 209	╤
210	D2	ALT 210	╥
211	D3	ALT 211	╙
212	D4	ALT 212	╘
213	D5	ALT 213	╒
214	D6	ALT 214	╓
215	D7	ALT 215	╫
216	D8	ALT 216	╪
217	D9	ALT 217	┘
218	DA	ALT 218	┌
219	DB	ALT 219	█
220	DC	ALT 220	▄
221	DD	ALT 221	▌
222	DE	ALT 222	▐
223	DF	ALT 223	▀

VALUE			
DEC	HEX	KEYSTROKES	
224	E0	ALT 224	α
225	E1	ALT 225	ß
226	E2	ALT 226	Γ
227	E3	ALT 227	π
228	E4	ALT 228	Σ
229	E5	ALT 229	σ
230	E6	ALT 230	µ
231	E7	ALT 231	τ
232	E8	ALT 232	Φ
233	E9	ALT 233	Θ
234	EA	ALT 234	Ω
235	EB	ALT 235	δ
236	EC	ALT 236	∞
237	ED	ALT 237	φ
238	EE	ALT 238	ε
239	EF	ALT 239	∩

VALUE			
DEC	HEX	KEYSTROKES	
240	F0	ALT 240	≡
241	F1	ALT 241	±
242	F2	ALT 242	≥
243	F3	ALT 243	≤
244	F4	ALT 244	⌠
245	F5	ALT 245	⌡
246	F6	ALT 246	÷
247	F7	ALT 247	≈
248	F8	ALT 248	°
249	F9	ALT 249	∙
250	FA	ALT 250	·
251	FB	ALT 251	√
252	FC	ALT 252	ⁿ
253	FD	ALT 253	²
254	FE	ALT 254	■
255	FF	ALT 255	

E. Keyboard Position Codes

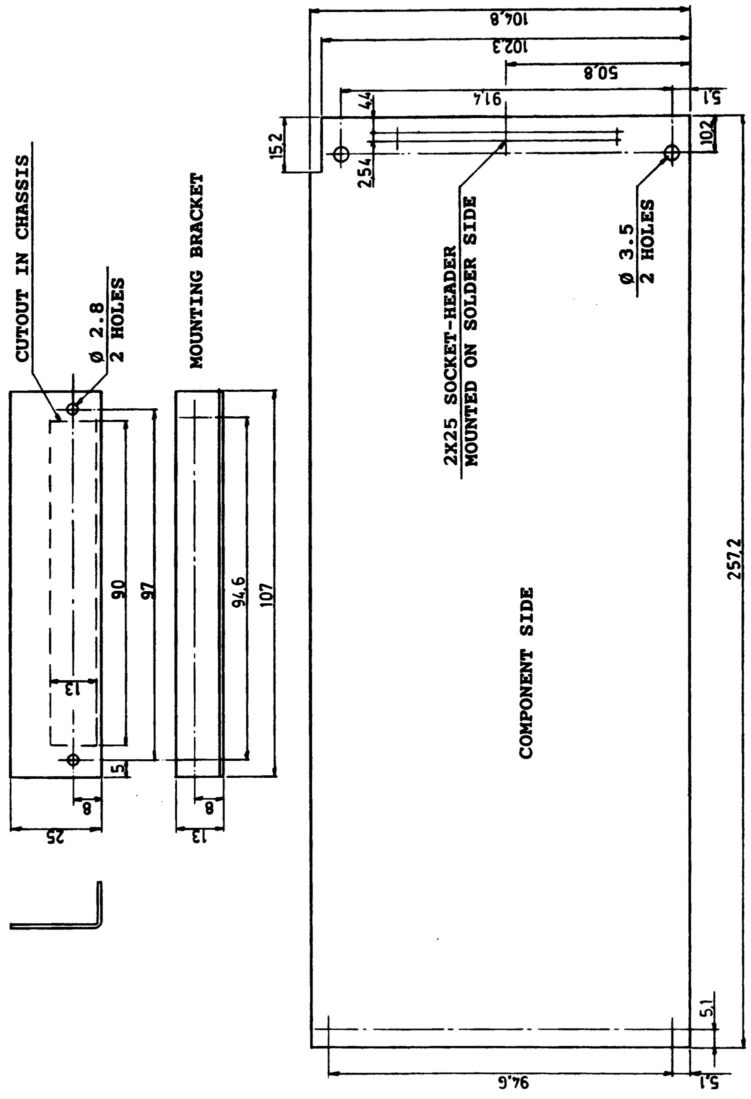


F. Console Escape Sequences

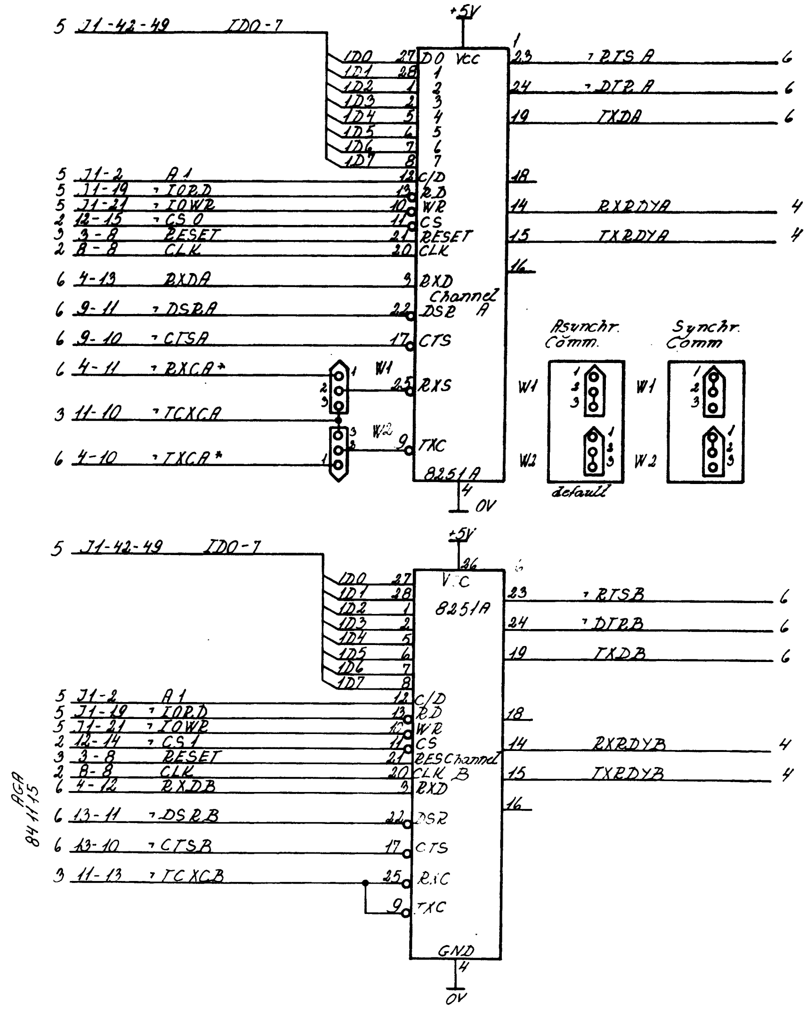
Sequence	Function
ESC A	Cursor Up
ESC B	Cursor Down
ESC C	Cursor Forward
ESC D	Cursor Backward
ESC E	Clear Screen
ESC H	Home Cursor
ESC I	Reverse Index
ESC J	Erase to End of Screen
ESC K	Erase to end of line
ESC L	Insert Line
ESC M	Delete Line
ESC N	Delete Character
ESC O	Insert Character
ESC P	Select Alternative Character Set
ESC Q	Select Standard Character Set
ESC Y xx xx	Position Cursor
ESC a xx	Ignored
ESC b xx	Set Foreground Colour
ESC c xx	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

Sequence	Function
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
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 : xx c...c NUL	Program Function Keys
ESC < xx xx	Scroll Window Up
ESC > xx xx	Scroll Window Up
ESC <238>	Change Function Key Terminator
ESC <239>	Set Transparent Mouse Mode
ESC <240>	Set Normal Mouse Mode
ESC <241>	Set Blinking Cursor
ESC <242>	Set Non-Blinking Cursor
ESC <243> xx	Set Cursor Representation
ESC <244>	Set Soft Scroll
ESC <245>	Set Line Scroll
ESC <246>	Disable Underline Attribute
ESC <247>	Enable Underline Attibute

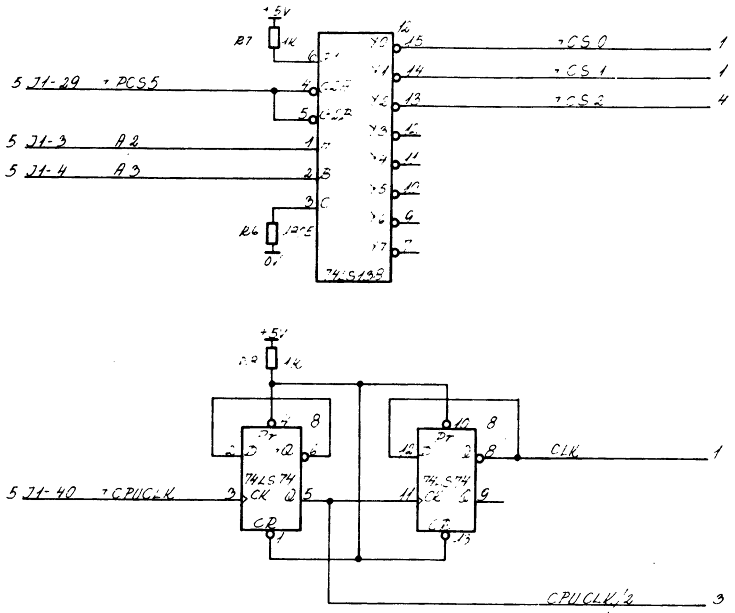
G. Adapter Board Dimensions



H. MF140 Schematics

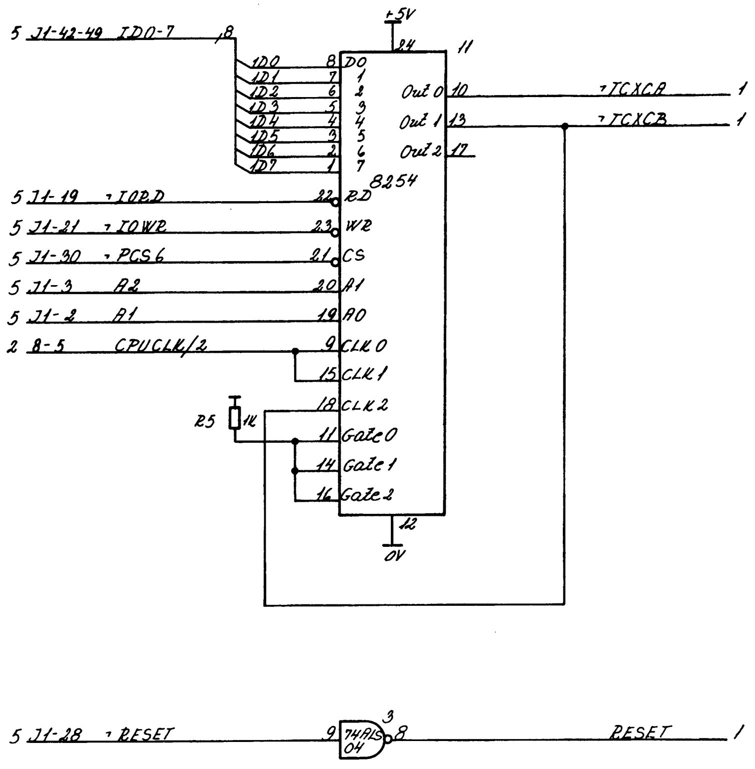


COM751 Programmable Communication Controllers Channel A & B 1



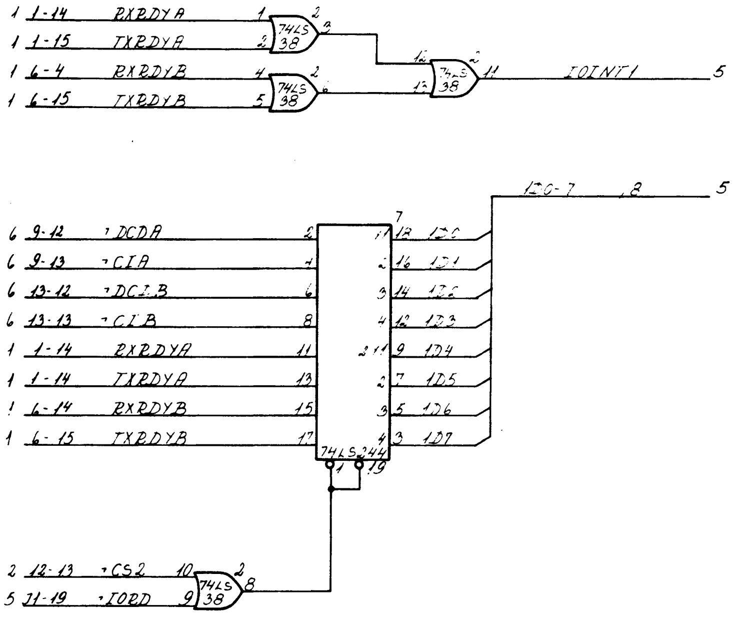
AGA 84 11 15

COM751   I/O Select and Clock   2



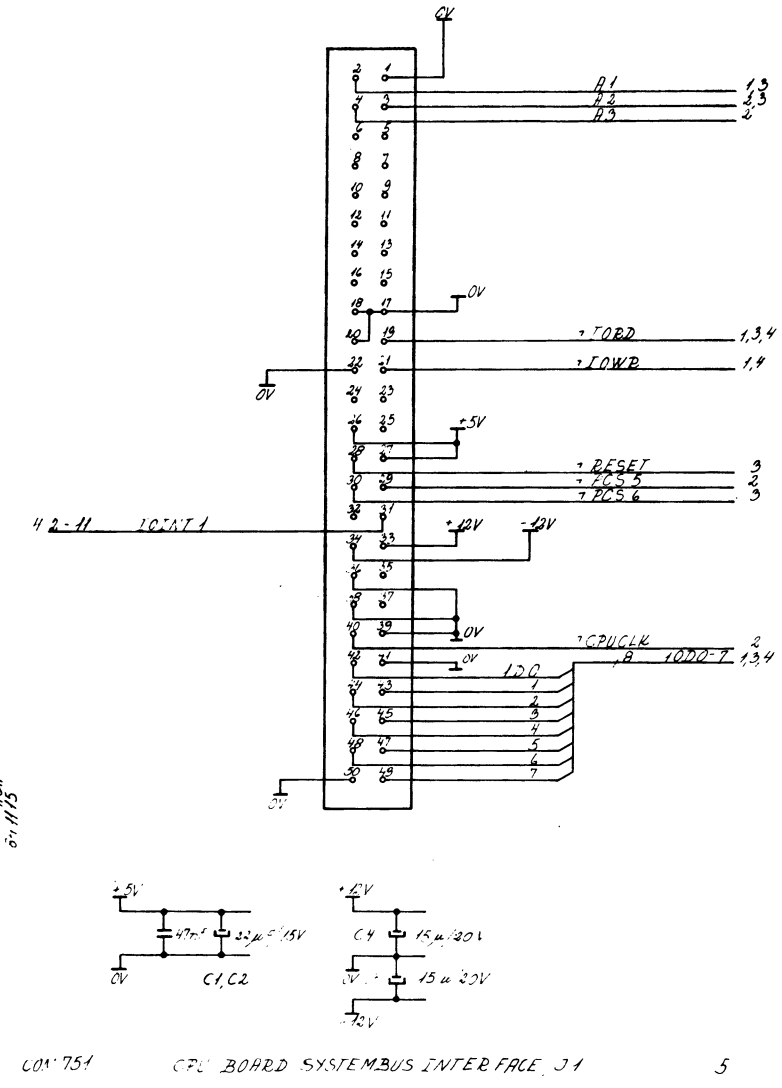
84 11 15 AGA

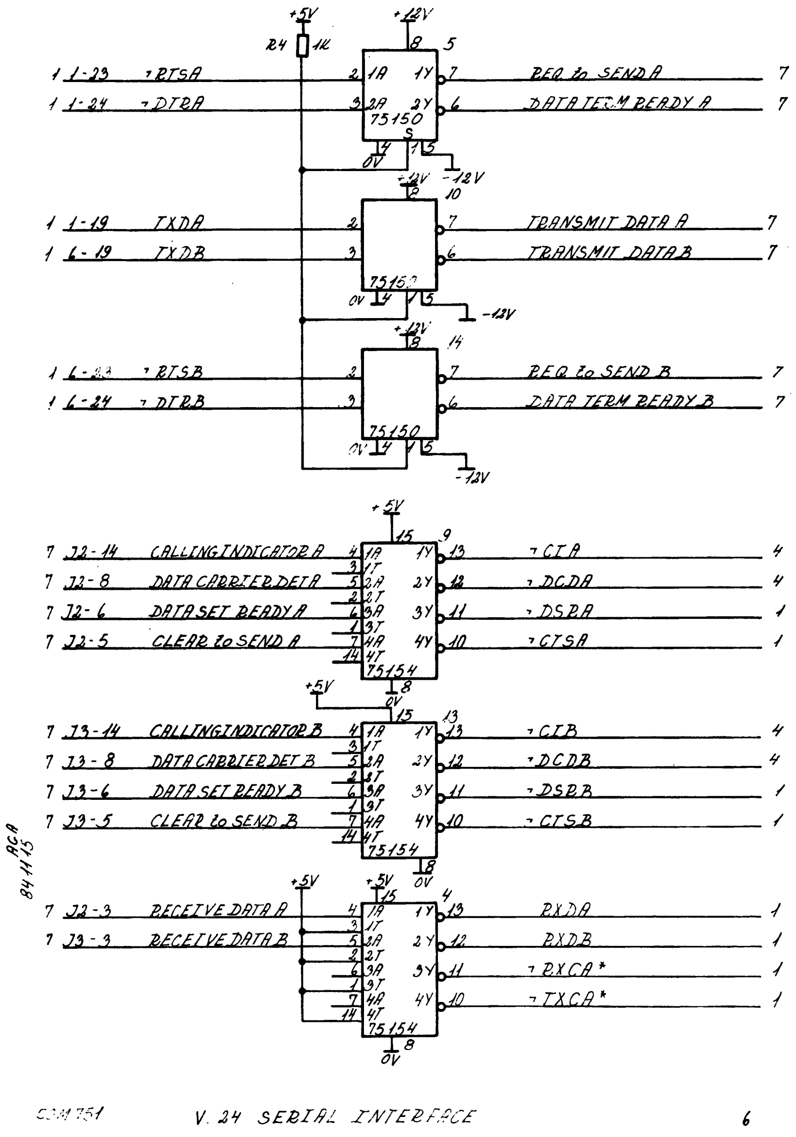
COM 751 Programmable Baud Rate Generator 3



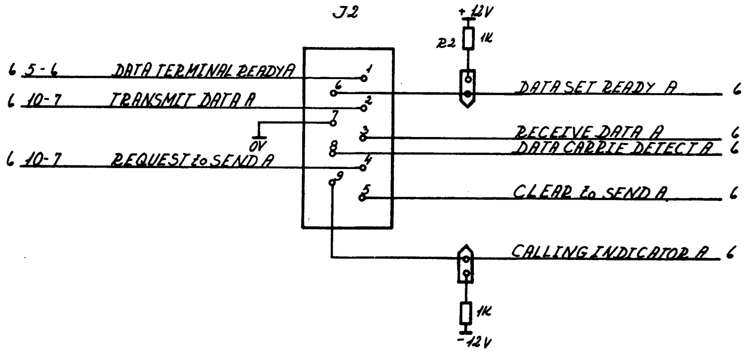
RGA 84/1/15

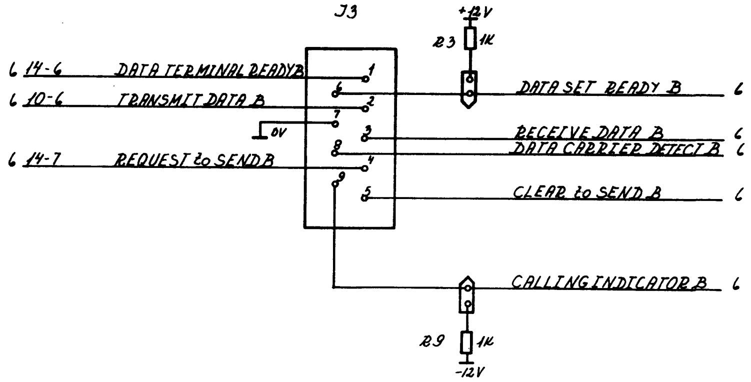
COM 751 Interrupt Logic & Status Buffer 4


0V
A1 1,3
A2 2,3
A3 2
IORD 1,3,4
IOWR 1,4
+5V
RESET 3
PCS 5 2
PCS 6 3
4 2-11 IOINT 1
+12V
-12V
CPUCLK 2
8 IOD0-7 1,3,4
IDO
1
2
3
4
5
6
7
AGA 81.11.15
47nF
22µF 15V
C1, C2
C4
15µ 20V
15µ 20V
-12V
CON 751 CPU BOARD SYSTEMBUS INTERFACE J1
5



+5V
+12V
R4 1K
75150
1 1-23 ¬RTSA
1 1-24 ¬DTRA
REQ to SEND A 7
DATA TERM READY A 7
0V
-12V
1 1-19 TXDA
1 L-19 TXDB
TRANSMIT DATA A 7
TRANSMIT DATA B 7
1 L-23 ¬RTSB
1 L-24 ¬DTRB
REQ to SEND B 7
DATA TERM READY B 7
75154
7 J2-14 CALLING INDICATOR A
7 J2-8 DATA CARRIER DET A
7 J2-6 DATA SET READY A
7 J2-5 CLEAR to SEND A
¬CIA 4
¬DCDA 4
¬DSRA 1
¬CTSA 1
7 J3-14 CALLING INDICATOR B
7 J3-8 DATA CARRIER DET B
7 J3-6 DATA SET READY B
7 J3-5 CLEAR to SEND B
¬CIB 4
¬DCDB 4
¬DSRB 1
¬CTSB 1
7 J2-3 RECEIVE DATA A
7 J3-3 RECEIVE DATA B
RXDA 1
RXDB 1
¬RXCA* 1
¬TXCA* 1
ACA 841115
CSM 751
V.24 SERIAL INTERFACE
6





AGA
84.11.15

COM 751 7

I. MF140 Test Program

```
; This program tests the V24 interface on the MF140
; Adapter.
; MF140 contains two V24 asynchronous channels
; with an onboard baudrate generator (I8254). A cable
; with the following connections must be inserted
; before execution of this program:
;
; Channel A                    Channel B
; Pin no.                      Pin no.
;    2 ───────────>──────────── 3
;    3 ───────────<──────────── 2
;    4 ──┐                  ┌── 4
;    5 <─┤                  ├─> 5
;    6 <─┘                  └─> 6
;    1 ──┐                  ┌── 1
;    8 <─┤                  ├─> 8
;    9 <─┘                  └─> 9
;
;
```

```
        CSEG
        ORG  100H


; Initialize interrupt vector

        MOV  AX,0
        MOV  ES,AX
        MOV  AX,OFFSET sio_int_routine
        MOV  DI,9ch
        CLI
        STOSW
        MOV  AX,CS
        STOSW
        STI


; Initialize segment registers

        MOV  SS,AX
        MOV  SP,OFFSET stack_bottom
        MOV  DS,AX
        MOV  ES,AX


; Write introduction text

        MOV  CL,9
        MOV  DX,OFFSET sio_test
        INT  224


; Test if board is installed

        MOV  DX,220H              ; ioident address
        IN   AL,DX
        CMP  AL,1                 ; ioident=1 ?
        JZ   io_board_present
        MOV  err_no,7             ; v24 board not present !
sio_exit1 :
        JMP  sio_exit
io_board_present :


; Test baud rate generator

        CALL baud_rate_test
        CMP  err_no , 0
        JNZ  sio_exit1
```

```
; check V24 control signals as function of RTS and DTR

     MOV  DX,sio_cmda_addr
     MOV  BX,OFFSET modem_response
     MOV  CX,4
     CALL sio_reset
next_v24:
     MOV  AL,[BX]    ; get DTR and RTS from table
     OUT  DX,AL      ; output to sio
     PUSH CX
     MOV  CX,40H
v24_delay:
     LOOP v24_delay ; wait for sio
     POP  CX
     INC  BX
     IN   AL,DX      ; get status from sio
     AND  AL,10111111B; mask syndet/brkdet pin
     CMP  AL,[BX]
     JNZ  v24_status_error
     INC  BX
     PUSH DX
     MOV  DX,status_buf_addr
     IN   AL,DX
     POP  DX
     CMP  AL,[BX]
     JNZ  v24_status_error
     INC  BX
     LOOP next_v24
     MOV  AL,0       ; reset sio
     OUT  DX,AL
     CMP  DX,sio_cmdb_addr
     JZ   sio_transfer_test
     MOV  DX,sio_cmdb_addr
     MOV  CX,4
     JMP  next_v24
v24_status_error:
     MOV  err_no,3
     JMP  sio_exit


;  Transmit data from channel A to B and vice versa

sio_transfer_test :
     MOV  BX,OFFSET txm_bufa_addr
     MOV  txm_bufa_pointer,BX
     MOV  BX,OFFSET txm_bufb_addr
     MOV  txm_bufb_pointer,BX
     CALL set_buf              ; initialize txm buffer
     MOV  BX,OFFSET rec_bufa_addr
```

```
        MOV  rec_bufa_pointer,BX
        MOV  AX,bufsize
        CALL reset_buf
        MOV  BX,OFFSET rec_bufb_addr
        MOV  rec_bufb_pointer,BX
        MOV  AX,bufsize
        CALL reset_buf
        IN   AL,02      ; fetch int-mask
        AND  AL,7FH
        OUT  02,AL      ; enable int from i/o board
        MOV  DX,sio_cmda_addr
        MOV  AL,00100111B; txm rec enable chan a
        OUT  DX,AL
        MOV  DX,sio_cmdb_addr; enable channel b
        MOV  AL,00100111B ;transmit and receive
        OUT  DX,AL
        MOV  delay_count , 00
   sio_delay:
        MOV  CX,0FFFFH
   sec_delay:
        LOOP sec_delay
        CMP  err_no,0   ; stop if error has occured
        JNZ  sio_exit
        MOV  CL,02H
        MOV  DL,'*'
        INT  224                 ; writeout '*'
        INC  delay_count
        CMP  delay_count,45; error if more than 15 sec
        JZ   time_error
        MOV  BX,rec_bufa_pointer
        SUB  BX,OFFSET rec_bufa_addr
        CMP  bx,bufsize
        JNZ  sio_delay
        MOV  BX,rec_bufb_pointer
        SUB  BX,OFFSET rec_bufb_addr
        CMP  BX,bufsize
        JNZ  sio_delay
        JMP  buffer_test
   time_error:
        MOV  err_no,4
        JMP  sio_exit


   ; check the receive buffer

   buffer_test:
        MOV  CX,bufsize
        MOV  BX,0
```

```
    buf_test_loop:
         MOV  AX,txm_bufa_addr[BX]
         CMP  AX,rec_bufb_addr[BX]
         JNZ  trans_error
         MOV  AX,txm_bufb_addr[BX]
         CMP  AX,rec_bufa_addr[BX]
         JNZ  trans_error
         INC  BX
         LOOP buf_test_loop
         JMP  sio_exit
    trans_error:
         MOV  err_no,6


;          write error messages to the consol

    sio_exit:
         MOV  AL,err_no ;
         MOV  AH,0
         MOV  DX,OFFSET sio_text2 - OFFSET sio_text1
         IMUL DL
         ADD  AX,OFFSET sio_text1
         CALL AX
         MOV  CL,9                   ; write text
         INT  224
         MOV  CL,8fH
         MOV  DL,0FFH                ; terminate process
         INT  224
    sio_text1:
         MOV  DX,OFFSET sio_ok
         RET
    sio_text2 :
         MOV  DX,OFFSET error_1
         RET
         MOV  DX,OFFSET error_2
         RET
         MOV  DX,OFFSET error_3
         RET
         MOV  DX,OFFSET error_4
         RET
         MOV  DX,OFFSET error_5
         RET
         MOV  DX,OFFSET error_6
         RET
         MOV  DX,OFFSET error_7
         RET
         MOV  DX,OFFSET error_8
         RET
```

```
;  Reset the buffer defined by
;            BX : buffer start address
;            AX : buffer size in words

    reset_buf :
         PUSH CX
         PUSH DX
         MOV  CX,AX
         MOV  DX,BX
         PUSH BX
         MOV  BX,DX
    reset_loop:
         MOV  WORD PTR[BX],0
         INC  BX
         LOOP reset_loop
         POP  BX
         POP  DX
         POP  CX
         RET


; This procedure inserts a counting
; pattern in the transmit buffers

     set_buf :
         PUSH AX
         PUSH BX
         PUSH CX
         MOV  CX,bufsize
         MOV  BX,txm_bufa_pointer
         MOV  AX,0
    set_bufa:
         MOV  WORD PTR[BX],AX
         INC  AL
         INC  AH
         ADD  BX,2
         LOOP set_bufa
         MOV  CX,bufsize
         MOV  BX,txm_bufb_pointer
         MOV  AX,0
    set_bufb :
         MOV  WORD PTR[BX],AX
         INC  AL
         INC  AH
         ADD  BX,2
         LOOP set_bufb
         POP  CX
         POP  BX
         POP  AX
         RET
```

```
;  Procedure to test and initialize the
;  baud rate generator 8254

    baud_rate_test :
         PUSH DX
         PUSH AX
         PUSH CX
         MOV  DX,cntrwadd_8254
         MOV  AL,controlw_count0
         OUT  DX,AL
         MOV  AL,controlw_count1
         OUT  DX,AL
         MOV  AL,controlw_count2
         OUT  DX,AL
         MOV  DX,count0_addr
         MOV  AL,baud_rate0_lsb
         OUT  DX,AL
         MOV  AL,baud_rate0_msb
         OUT  DX,AL
         MOV  DX,count1_addr
         MOV  AL,baud_rate1_lsb
         OUT  DX,AL
         MOV  AL,baud_rate1_msb
         OUT  DX,AL
         MOV  DX,count2_addr
         MOV  AL,1
         OUT  DX,AL
         MOV  AL,0
         OUT  DX,AL
         MOV  CX,100h
    delay_8254 :
         LOOP delay_8254
         MOV  AL,80h
         MOV  DX,cntrwadd_8254; counter latch read
         OUT  DX,AL
         MOV  DX,count2_addr
         IN   AL,DX
         TEST AL,baud_rate2_lsb
         JZ   timer_excit
         MOV  err_no,5
    timer_excit:
         POP  CX
         POP  AX
         POP  DX
         RET

    ; end of  8254 test
; This procedure resets and initializes channel A
; and channel B , PCI 8251

    sio_reset :
         PUSH AX
         PUSH CX
         PUSH DX
         MOV  CX,3
         MOV  DX,sio_cmda_addr
         MOV  AL,0
    sio_a_reset:
         OUT  DX,AL
         CALL sio_res_delay
         LOOP sio_a_reset
         MOV  AL,40h
         OUT  DX,AL     ; reset chan. a sio
         MOV  AL,sio_a_mode
         CALL sio_res_delay
         OUT  DX,AL
         MOV  DX,sio_cmdb_addr
         MOV  AL,0
         MOV  CX,3
    sio_b_reset:
         OUT  DX,AL
         CALL sio_res_delay
         LOOP sio_b_reset
         MOV  AL,40h
         OUT  DX,AL     ; reset channel b sio
         MOV  AL,sio_b_mode
         CALL sio_res_delay
         OUT  DX,AL
         POP  DX
         POP  CX
         POP  AX
         RET

    ; end of init routine



;  This procedure delays program with 16 loop instructions

    sio_res_delay :
         PUSH CX
         MOV CX , 10H
    sio_res_loop :
         LOOP sio_res_loop
         POP CX
         RET
```

```
                         RB  398

    local_stack          DW  0
    user_ss              DW  0
    user_sp              DW  0



;      interrupt service routine for the 8251a
;      devices on the IO board

    sio_int_routine:
         PUSH AX
         MOV  user_ss,SS
         MOV  user_sp,SP
         MOV  SP,OFFSET local_stack
         MOV  AX,CS
         MOV  SS,AX
         PUSH DS
         MOV  DS,AX
         PUSH BX
         PUSH DX
         MOV  loop_counter,0; initialize
    intr_source :
         INC  loop_counter  ; test for illegal interrupt
         MOV  AX,loop_counter
         CMP  AX,bufsize*6
         JZ   illegal_intr
         MOV  DX,status_buf_addr
         IN   AL,DX     ; get staus byte
         TEST AL,00010000B        ; RxRDYA
         JNZ  rec_cha
         TEST AL,01000000B        ; RxRDYB
         JNZ  rec_chb
         TEST AL,00100000B        ; TxRDYA
         JNZ  transm_cha
         TEST AL,10000000B        ; TxRDYB
         JNZ  transm_chb
         JMP  end_of_intr

    illegal_intr:
         MOV  err_no,8
         MOV  AL,00000100B        ; STOP SIO
         MOV  DX,sio_cmda_addr
         OUT  DX,AL
         MOV  DX,sio_cmdb_addr
         OUT  DX,AL
```

```
        IN   AL,2 ; get int mask from intr. controller
        OR   AL,8DH                ; disable interrupt -
        OUT  2,AL                  ; from IO board
        JMP  end_of_intr


;  Receive channel a interrupt service routine

    rec_cha:
        MOV  DX,sio_cmda_addr
        IN   AL,DX                 ; get status information
        AND  AL,000111000B         ; frame-, overrun-
                                   ; ,parity error
        JZ   rec_data_a
        MOV  err_no,1
    rec_data_a:
        MOV  DX,sio_dataa_addr
        IN   AL,DX      ; get data from sio chan a
        MOV  BX,rec_bufa_pointer
        MOV  [BX],AL               ; store received data
        INC  rec_bufa_pointer
        JMP  intr_source


; Receive channel b interrupt service routine

    rec_chb:
        MOV  DX,sio_cmdb_addr
        IN   AL,DX                 ; get status information
        AND  AL,000111000B         ; FE , OE and PE errors
        JZ   rec_data_b
        MOV  err_no,1
    rec_data_b:
        MOV  DX,sio_datab_addr
        IN   AL,DX                 ; get data from sio b
        MOV  BX,rec_bufb_pointer
        MOV  [BX],AL               ; store data byte
        INC  rec_bufb_pointer
        JMP  intr_source


; Transmit channel a interrupt service routine

    transm_cha:
        MOV  DX,sio_dataa_addr
        MOV  BX,txm_bufa_pointer
        MOV  AL,[BX]               ; get byte to send
        OUT  DX,AL
        INC  BX
        MOV  txm_bufa_pointer,BX
        CMP  BX,OFFSET txm_bufa_addr+bufsize
        JNZ  intr_source_1
        MOV  DX,sio_cmda_addr
        MOV  AL,00000100B          ; disable transmit
        OUT  DX,AL
    intr_source_1:
        JMP  intr_source


; Transmit channel b interrupt service routine

    transm_chb:
        MOV  DX,sio_datab_addr
        MOV  BX,txm_bufb_pointer
        MOV  AL,[BX]               ; get byte
        OUT  DX,AL                 ; transmit data
        INC  BX
        MOV  txm_bufb_pointer,BX
        CMP  BX,OFFSET txm_bufb_addr + bufsize
        JNZ  intr_source_1
        MOV  DX,sio_cmdb_addr
        MOV  AL,00000100B          ; disable transmit
        OUT  DX,AL
        JMP intr_source


; End of sio interrupt routine

    end_of_intr:
        MOV  AL,20h                ; EOI to pic
        OUT  0,AL
        MOV  DX,0ff22H             ; EOI to CPU
        MOV  AX,8000H
        OUT  DX,AX
        POP  DX
        POP  BX
        POP  DS
        MOV  SP,user_sp
        MOV  SS,user_ss
        POP  AX
        IRET


        RB 400H                    ; Memory for stack
    stack_bottom  DW  00
```

```
;        Data variables

    sio_cmda_addr        EQU        282H
    sio_cmdb_addr    EQU 286H
    status_buf_addr EQU 288H
    bufsize              EQU        512        ; 1k bytes
    cntrwadd_8254        EQU        306H
    controlw_count0      EQU        00110110B
    controlw_count1      EQU        01110110B
    controlw_count2      EQU        10110110B
    count0_addr          EQU        300H
    count1_addr          EQU        302H
    count2_addr          EQU        304H
    sio_a_mode           EQU        01111110B ; 1 stopbit,
                                              ; even parity
    sio_b_mode           EQU        01111110B ; parity
                                              ; enable,
                                              ; 8 char,async
    baud_rate0_lsb       EQU        23
    baud_rate0_msb       EQU         0
    baud_rate1_lsb       EQU        23
    baud_rate1_msb       EQU         0
    baud_rate2_lsb       EQU         1
    baud_rate2_msb       EQU         0
    sio_dataa_addr       EQU        280H
    sio_datab_addr       EQU        284H
    cr                   EQU        13
    lf                   EQU        10

                         rb         1
    txm_bufa_addr        RW         512
    txm_bufb_addr        RW         512
    rec_bufa_addr        RW         512
    rec_bufb_addr        RW         512
    txm_bufa_pointer     RW         1
    txm_bufb_pointer     RW         1
    rec_bufa_pointer     RW         1
    rec_bufb_pointer     RW         1
    delay_count          RW         1
    loop_counter         RW         1
    err_no               DB         0

    sio_ok               DB   cr,lf
                         DB   'END OF TEST'
                         DB   cr,lf,'$'

    error_1              DB   cr,lf
                         DB   'V24 data transfer error'
                         DB   'fe,oe,pe'
                         DB   cr,lf,'$'
    error_2             DB  cr,lf
                        DB  'illegal interrupt from 8251'
                        DB  cr,lf,'$'

    error_3             DB  cr,lf
                        DB 'error in v24 status check'
                        DB cr,lf,'$'

    error_4             DB cr,lf
                        DB 'v24 data transfer time out'
                        DB cr,lf,'$'

    error_5             DB cr,lf
                        DB 'timer error'
                        DB cr,lf,'$'

    error_6             DB cr,lf
                        DB 'v24 data transfer error'
                        DB cr,lf,'$'

    error_7             DB cr,lf
                        DB 'I/O-board is not installed'
                        DB cr,lf,'$'

    error_8             DB cr,lf
                        DB 'illegal interrupt'
                        DB ' from i/o board'
                        DB cr,lf,'$'

    sio_test            DB cr,lf
                        DB 'TEST OF I/O-BOARD'
                        DB ' WITH TWO ASYNCH. V24 '
                        DB 'CHANNELS'
                        DB cr,lf,lf,'$'


; Table to calculate the response of the loop-back
; of status signals

modem_response:
              DB  00000000B ; input to sio
              DB  00000101B ; output from sio
              DB  00001111B ; output from status buffer
              DB  00100000B ; input to sio
              DB  10000101B ; output from sio
              DB  00001111B ; output from status buffer
              DB  00100001B ; input to sio
              DB  10000101B ; output from sio
              DB  00101111B ; output from status buffer
DB   00000010B ; input to sio
DB   00000101B ; output from sio
DB   00001100B ; output from status buffer
DB   00H       ; input to sio b
DB   05H       ; output from sio b
DB   0fH       ; output from statusbuffer
DB   20H       ; input to sio b
DB   85H       ; output from sio b
DB   0fH       ; output from status buffer
DB   21H       ; input to sio b
DB   85H       ; output from sio b
DB   8fH       ; output from status buffer
DB   02H       ; input to sio b
DB   05H       ; output from sio b
DB   03H       ; output from statusbuffer


END
```

J. Adapter Connector

The pin assignments for the adapter connector is as follows:

Pin	Signal	Description
1	Ground	
2-16	A1-A15	Addressbus capable of addressing 32768 I/O ports
17-18	Ground	
19	-,IORD	When active, the addressed device should place the relevant data on the databus.
20	Ground	
21	-,IOWR	When active, relevant data is present on the databus and may be strobed into the addressed device.
22-25	IOIDENT	Strap options for identification of the attached I/O devices.
26-27	+5V	
28	-,RESET	Low when a hardware reset is issued.
29	-,PCS5	Decoded chip select output. Active on addresses 280H to 2FEH.
30	-,PCS6	Decoded chip select output. Active on addresses 300H to 37EH.
31	IRQ	Interrupt request. Assigned to interrupt 7 on the INTEL8259 interrupt controller.
32	Sound	Analog summation point for the sound device. The sum of the signals on this pin is sent to the loudspeaker in the crt enclosure.
33	+12V	
34	-12V	
35	DRQ6	DMA request input 6.
36	Ground	
37	DRQ7	DMA request input 7.
38-39	Ground	
40	-,CPUCLK	Clock pulse from CPU.
41	Ground	
42-49	ID0-7	Databus.

Interrupt handling

The I/O expansion connector has been assigned interrupt level 7 on the INTEL 8259 interrupt controller which is initialized to be level triggered. Level 7 is special in that erroneous interrupts caused by noise spikes will result in a interrupt on this level. Such an erroneous interrupt can be detected by the interrupt service routine by inspecting the "in service bit" register. If there is no "in service bit" for level 7 the interrupt was erroneous.
An example of an interrupt routine for an I/O expansion board may be found in appendix I.

DMA handling

Expansion boards has been assigned two DMA request lines: DRQ 6 and DRQ 7, i.e. an expansion board could use the two DMA channels simultaneously. Further information about DMA usage may be found in chapter 2.2.

Z. 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. Partner brugervejledning.
   SW1500D
   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

Index

Page numbers may refer either to the page containing the referenced word, or to the page containing the first line of the text block in which the referenced word occurs.

A

adapter board dimensions, 241
adapter, connector pin assignments, 265
adapters, 163
adapters, DMA handling, 266
adapters, interrupt handling, 266
alphanumeric mode, 38
alternative character set, 45
arithmetic co-processor, 7
arithmetic co-processor, 7
asynchronous communication, 87
attribute bits, 41
attribute byte, 43
audio output, 22
auto configuration, 23
auxiliary device, 88

B

baudrate, 22
BELL, 22

C

channel A, 87, 89
channel B, 87, 89
CHAR8 program, 73
character definition block, 44
character definitions, 43
character set, 219
character sets, 43
checksumbyte, 29
client, 128
colour select characters, 52
communication support, 87
Concurrent DOS flag, 12
Concurrent DOS operating system, 9

configuration, 23
configuration map, 23
CONFIRM, 130
conin, 33
conout, 33, 39
console control characters, 48
console device, 88
console driver, 41
console escape sequences, 48, 239
console module, 33
control block, 17
control characters, 48
CPU, 7, 17
CPU Interrupts, 215
CRT controller, 7, 33
cursor blink mode, 26
cursor height, 26
cursor position, 42

D

data link layer, 129
datalink service, 127
device I/O addresses, 213
direct memory access: see DMA
disk driver, 114
disk format, 107
disk system, 107
display buffer, 41
DMA, 7
DMA channels, 17, 19, 21
DMA request sources, 20
DR-NET, 127
drive designations, 107
drive letters, 107

E

escape sequences, 239
ethernet controller, 7, 148
expansion board dimensions, 241
expansion board, connector pin assignments, 265
expansion boards, 163
expansion boards, DMA handling, 266
expansion boards, interrupt handling, 266

F

file transfer program, 187
FILEX, 187
flag assignments, 13
flags, 12
flexible disk drives, 107
floppy disk controller, 7, 111
floppy disk driver, 113
floppy disks, 107
foreground colour, 27
function keys, 57, 72

G

graphics control block, 63
graphics mode, 38, 62

H

handshake scheme, 88
hardware configuration map, 23

I

I/O Adapters, 163
I/O addresses, 213
I/O board, connector pin assignments, 265
I/O expansion board dimensions, 241
I/O expansion connector, 7
IBM PC compatible character set, 73
IEEE488 adapter, 171
IMC, 127
INDICATION, 131
INDICATION ACKNOWLEDGE, 132
Int-28h functions, 12, 193
Intel 8087, 7
Intel 8251A, 174, 179
Intel 8254, 90, 180
Intel 8259A, 7, 18, 149
Intel 8274, 7, 18, 87
Intel 8274 initialization, 95
Intel 8291 and 8292, 171
Intel 80186, 17
Intel 80730, 7
Intel 82586, 7, 148

Intel 82730, 33
interrupt controller, 7, 17
interrupt system, 18
interrupt vector assignment, 215
IR inputs, 18
ISO model, 128

K

keyboard driver, 71
keyboard interface, 7, 71
keyboard position codes, 237
keystrokes, 219
KONFIG program, 23, 88

L

LAN, 127
list device, 88
load device, 27
Local Area Network, 127
local area network controller, 7
loop-back test facility, 136
loudspeaker, 79

M

MAC, 128
Medium Access Control (MAC) layer, 128
MF140 Dual Satellite Adapter, 163
MF140 schematics, 243
MF140 test program, 251
MF141 Parallel/Printer Adapter, 166
MF142 NRZI Adapter, 166
MF143 IEEE488 Adapter, 171
MF144 modem, 174
modem, 174
mouse interface, 73

N

nationality code switch, 24
NETDRV, 129
network software, 127
noise generator, 83

non-volatile memory: see NVM
non-volatile memory: see NVM
NRZI Adapter, 166
NVM, 7, 23
NVM layout, 25

P

palette, 35
palette selector, 36
parallel interface, 119, 166
PC keyboard, 72
PC-mode, 72
peripherals, 7, 17
PIC interrupts, 216
pixel memory, 67
pixel memory, 34, 43, 62
position code, 71
position codes, 237
programmable function keys, 57
programmable function keys, 72

Q

queues, 14

R

RCLLC procedures, 150
RCLLC protocol, 127
RCLLC protocol element, 150, 157
RCLLC Service Access Point (SAP), 128
RCLLC services, 133
RCLLC station, 128
real time clock, 7
real-time clock, 75
references, 267
registers, 11, 17
REQUEST, 130
reservation, 14
reserved queue names, 14
resolution in pixels, 34
RS232-C/V24, 89

S

SAP, 128
satellite adapter, 163
screen buffer, 41
scroll mode, 26
SCSI bit vector, 24
SCSI bus interface, 7
SCSI interface, 113
SCSI interface driver, 114
serial communication controller, 7, 89
serial interface, 87
serial number, 25
SIO interrupts, 216
sound device, 7, 79
synchronization, 12
synchronous communication, 87

T

test program for MF140, 251
timers, 7, 17, 22
Tmp queues, 14
tone output, 79
type number, 25

U

UDA, 11

V

V.24 handshake scheme, 88
V.24 interface test program, 251
V24/COM, 89
virtual console, 41

W

WD1797, 7, 111
winchester disk drives, 107
window, 42
window handling, 68

X

X.21, 87, 92
XIOS, 9
XIOS console driver, 48
XIOS routines, 9

Other

8-bit PC-mode, 73

LÆSERBEMÆRKNINGER

Titel: Partner Programmer's Guide, Version 3.0 **RCSL Nr.:** 991-10560

A/S Regnecentralen af 1979 bestræber sig på at forbedre kvalitet og brugbarhed af sine publikationer. For at opnå dette ønskes læserens kritiske vurdering af denne publikation.

Kommenter venligst manualens fuldstændighed, nøjagtighed, disposition, anvendelighed og læsbarhed:

____

____

____

____

Angiv fundne fejl (reference til sidenummer):

____

____

____

____

Hvordan kan manualen forbedres:

____

____

____

____

Andre kommentarer:

____

____

____

____

____

Navn: ______________________ **Stilling:** ______________

Firma: ______________________________________

Adresse: ____________________________________

Dato: ________

På forhånd tak!

.... **Fold her** ....

.... **Riv ikke - Fold her og hæft** ....

Frankeres som brev

A/S REGNECENTRALEN af 1979

Informationsafdelingen
Lautrupbjerg 1
2750 Ballerup