PIOC
Software Guide

ND-60.161.3 EN

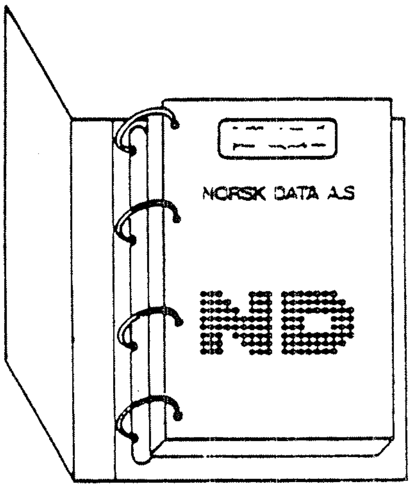
PIOC
Software Guide

ND-60.161.3 EN

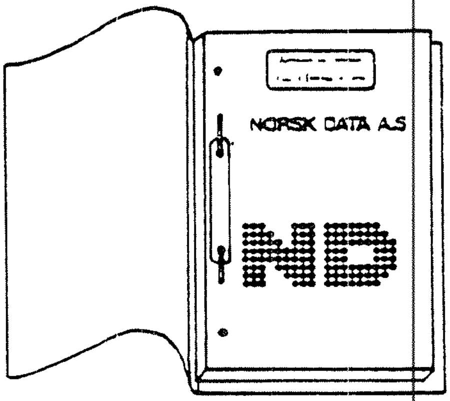
This manual is in loose-leaf form for ease of updating. Old pages may be removed and new pages easily inserted if the manual is revised.

The loose-leaf form also allows you to place the manual in a ring binder (A) for greater protection and convenience of use. Ring binders with 4 rings corresponding to the holes in the manual may be ordered in two widths, 30 mm and 40 mm. Use the order form below.

The manual may also be placed in a plastic cover (B). This cover is more suitable for manuals of less than 100 pages than for large manuals. Plastic covers may also be ordered below.



A: Ring Binder



B: Plastic Cover

Please send your order to the local ND office or (in Norway) to:

Norsk Data A.S
Graphic Center
P.O. Box 25, Bogerud
0621 Oslo 6, Norway

ORDER FORM

I would like to order

...... Ring Binders, 30 mm, at nkr 20,- per binder

...... Ring Binders, 40 mm, at nkr 25,- per binder

...... Plastic Covers at nkr 10,- per cover

Name ....

Company ....

Address ....

....

City ....

4

PRINTING RECORD

Printing	Notes
04/83	VERSION 01
04/84	VERSION 02
06/85	VERSION 03

PIOC Software Guide
Publ.No. ND-60.161.3 EN

Norsk Data

Norsk Data A.S
Graphic Center
P.O.Box 25, Bogerud
0621 Oslo 6, Norway

Manuals can be updated in two ways, new versions and revisions. New versions consist of a complete new manual which replaces the old manual. New versions incorporate all revisions since the previous version. Revisions consist of one or more single pages to be merged into the manual by the user, each revised page being listed on the new printing record sent out with the revision. The old printing record should be replaced by the new one.

New versions and revisions are announced in the Customer Support Information (CSI) and can be ordered as described below.

The reader's comments form at the back of this manual can be used both to report errors in the manual and to give an evaluation of the manual. Both detailed and general comments are welcome.

These forms and comments should be sent to:

Documentation Department
Norsk Data A.S
P.O. Box 25, Bogerud
0621 Oslo 6, Norway

Requests for documentation should be sent to the local ND office or (in Norway) to:

Graphic Center
Norsk Data A.S
P.O. Box 25, Bogerud
0621 Oslo 6, Norway

Preface:

THE PRODUCT

PIOC (Programmable Input Output Controller) is an interface card built around a MC68000 microprocessor.

ND-865 PIOC	with 128 Kbytes of memory, and
ND-867 PIOC	with 512 Kbytes of memory.
ND-10493C	PIOC Basic Software.

The card has 4 full duplex serial communication lines and a local memory (RAM) of 128 or 512 Kbytes.

Several PIOC modules can be installed on the ND-100 or on the ND-500 series of mini computers.

THE MANUAL

This manual describes the PIOC Basic Software version C of May 1985, for use with the ND-100 and the ND-500 computers.

It discusses the available calls to the PIOC operating system, synchronization of processes and input/output to the communication lines.

It also describes the PIOC-MONITOR, for debugging and supervising purposes, and how to compile, link and load a PIOC application program.

The appendices contain reference material such as symbol name tables, error codes and exception vector assignments.

CHANGES FROM PREVIOUS VERSION:

PIOCOS

Version C is compatible with version B, mainly errors have been corrected. However, new compilation might be necessary to get code and data separated, and the names of the include files should be changed.

PIOC-MONITOR

The PIOC-MONITOR-C has some new features and commands. It will only run with SINTRAN III release J, revision 6000 or higher.

Generating PIOCOS

The configuration procedures have been completely changed. Old mode files can not be used.

Tools

The include files have new names to match the ND file name standard. This should be taken into account when compiling application programs.

THE READER

The typical reader of this manual should have a good understanding of both software and hardware, and be familiar with computers from Norsk Data. Readers might be System Programmers or Application Programmers.

PREREQUISITE KNOWLEDGE

The reader should be familiar with the following Norsk Data Software Products:

RT (Real Time) Programming in SINTRAN III

PLANC programming language

The SINTRAN III Real Time Loader

The ND-500 Linkage-Loader and ND-500 Monitor

RELATED MANUALS

PIOC Reference Manual	ND-02.003
SINTRAN III Communication Guide	ND-60.134
SINTRAN III Real Time Loader	ND-60.051
ND-500 Loader/Monitor	ND-60.136
PLANC Reference Manual	ND-60.117
HDLC Interface......................	ND-12.018

MC68000 16 bit Microprocessor Users Manual, published by Motorola Inc.

Modifications and additions to the text are marked with a vertical bar in the margin area of the page, to the left on an even numbered page, and to the right on an odd numbered page.

TABLE OF CONTENTS

Section Page

List of Figures.

Title Page

List of Tables.

List of Programming Examples.

1 INTRODUCTION TO PIOC

This chapter gives a brief overview of the PIOC concept, its purpose, what it consists of, and its typical applications.

1.1 The purpose of PIOC

PIOC is a Programmable I/O Controller interface card for the ND-100 and ND-500 computers. It is based on the powerful Motorola MC68000 microprocessor.

Its main purpose is to run complex data communication protocols and thus relieve ND-100 of such communication overhead. In addition, users may write their own programs, to utilize special communication protocols or data reduction routines, before the result is passed to the ND-100.

PIOC Basic Software is mostly known with the name PIOCOS, although this is only one but the most important part of PIOC Basic Software. Because it is shorter, the name PIOCOS will also be used in this manual.

PIOC means Programmable Input Output Controller. PIOCOS is therefore the Operating System for this controller.It is used as the basis for other ND products but also delivered in connection with an appropriate PLANC compiler and the ND Linkage-Loader to customers who want to develop their own application.

Software and applications that can be used with PIOC are being continuously developed. Thus the range of usage of the PIOC will expand in the future.

1.2 The PIOC Hardware Architecture

The illustration below represents a PC board which contains the whole controller. The processor on this controller is a MC68000 which works on a local memory of 128 or 512Kbyte.

The MC68000 can externally be controlled (RESET, HALT etc.) by a control register which is accessible from the ND-100. The ND-100 can also access the whole memory of the PIOC (including DMA) but not vice versa.

A timer on the PIOC is used to generate a (programmable) clock (baud rate) for the four SIOs and an interrupt to MC68000 to form a real time clock for PIOCOS. The Serial Input Output controllers are used for communications. The data transfer from and to the SIOs is done by DMA with the exception of asynchronous input because of handling XON/XOFF.

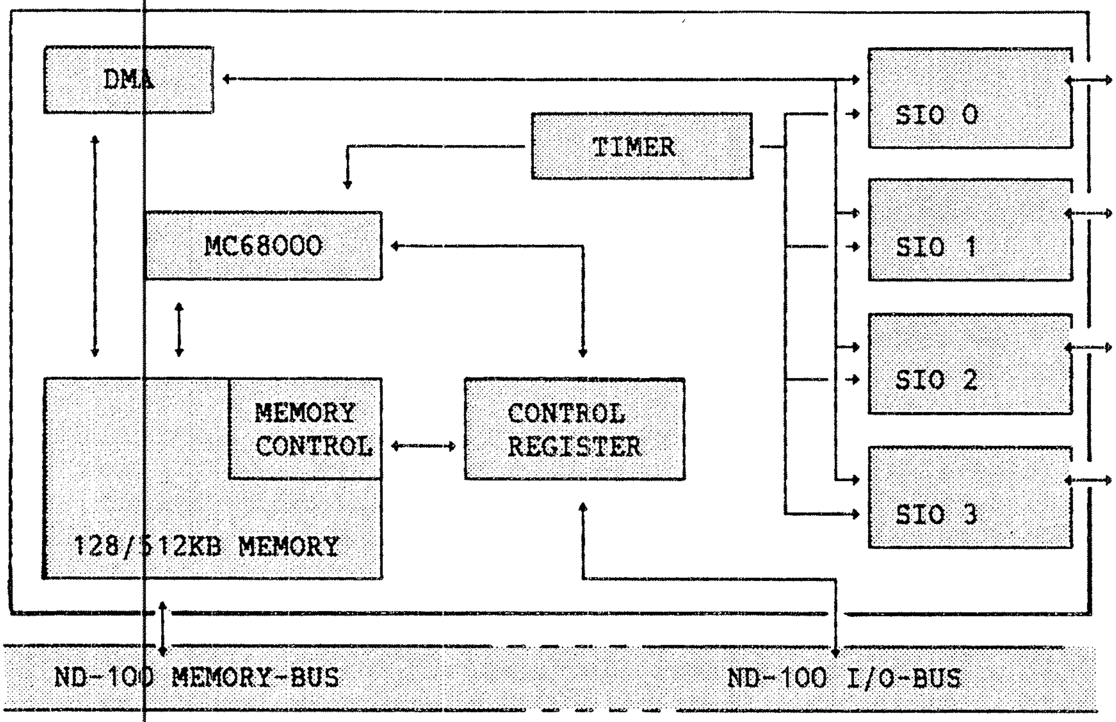


Figure 1. Main parts of a PIOC.

The basic PIOC supports RS 232C (V24/V28) and RS 422 (V11=X27) on all four channels and may operate in both synchronous (HDLC, SDLC or BISYNC) or asynchronous mode. The transfer speed may be up to 800 Kbits/second on one line or up to 38400 bits/second on all lines simultaneously.

1.3 The PIOC Software

The PIOC Basic Software includes the tools needed for loading, running, and debugging PIOC programs. Your job is to use these tools to develop PIOC applications.

The PIOC Basic Software consists of:

- PIOC-MONITOR for debugging, loading and starting a PIOC application
- PIOCOS, the PIOC real time operating system including serial link drivers for:
  - asynchronous communication modes
  - synchronous communication modes
- Tools for developing PIOC applications:
  - include files to incorporate standard definitions of codes and variable names when compiling source programs
  - procedures to generate PIOCOS tailored to a particular installation

Other software products that are necessary for the whole development task, are available as separate items:

- PED full screen editor, supporting a variety of terminals for writing and modifying source modules
- PLANC-MC68000 compiler, to produce object modules in Nord Relocatable Format (:NRF).
- ND Linkage-Loader, to build executable PIOC applications

  NB! The ND Linkage-Loader must be version G, that runs in the ND-100 (the :PROG file, not the ND-500 domain).

For the purpose of running a multiprogrammed system in PIOC, the PIOCOS real time operating system must be loaded in the PIOC memory, together with the application programs.

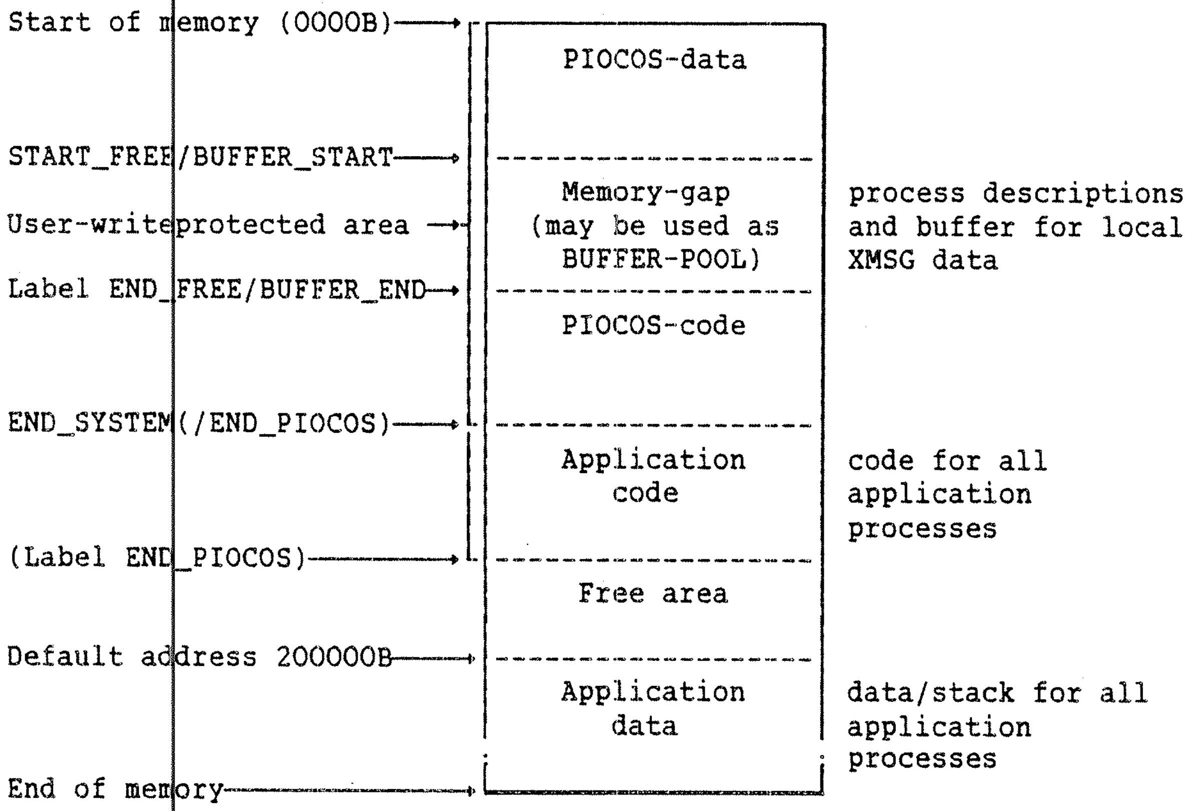

Start of memory (0000B)
START_FREE/BUFFER_START
User-write protected area
Label END_FREE/BUFFER_END
END_SYSTEM(/END_PIOCOS)
(Label END_PIOCOS)
Default address 200000B
End of memory
PIOCOS-data
Memory-gap (may be used as BUFFER-POOL)
PIOCOS-code
Application code
Free area
Application data
process descriptions and buffer for local XMSG data
code for all application processes
data/stack for all application processes

Figure 2. PIOC memory map

The illustration above shows the loading addresses for the PIOCOS module and the user written application program, with some automatically generated symbolic labels.

The object codes and data areas are loaded at different start addresses, if the compiler option $SEPARATE-DATA ON has been used.

2 WORDS AND EXPRESSIONS - QUICK REFERENCE

This chapter gives a brief explanation of the most commonly used expressions throughout this manual. You will also find references to the chapters and sections where they are discussed in more detail.

call

PIOC processes, commonly written in the ND high-level programming language PLANC, may ask for operating system assistance, by using system calls.

The PIOC operating system, PIOCOS, offers many calls. Each call makes use of parameters to tell PIOCOS exactly what type of assistance it requests. A more detailed explanation is found on page 13.

exception

An exception is a signal from a device or from the PIOC microprocessor (MC68000). The exception is set up when a special situation occurs. There are two types of exceptions, asynchronous and synchronous.

Asynchronous exceptions are set up when hardware devices, for example those connected to one of the four PIOC communication lines, wants to interrupt a process, or in case of a communication line error.

Synchronous exceptions are set up in case a process is trying to execute an illegal processor instruction, trying to fetch a word from an odd address, or if it is executing instructions whose normal behaviour is trapping.

fixed memory

To run any PIOC process, the whole of the PIOC memory must be fixed. The memory may consist of many smaller fixed segments in a contiguous area, or may be only one or more fixed segment of 128 Kbytes. The largest segment that can be fixed by SINTRAN III operating system is 128 Kbytes.

To fix a segment, it must be of the type non-demand, i.e., the whole segment must be placed into memory before the program can be started. Fixing memory is done by the SINTRAN command @FIXC (fix contiguous), the PIOC-MONITOR panel command SEGMENT-LOAD or the ND-100 monitor call MON PIOC (MON 255).

kick channel

The kick channel is an area in the PIOC memory which serves the purpose of transmitting information from PIOC to ND-100 and vice versa.

The channel consists of eight slots, each with two mailboxes. Two processes must reserve the same slot to be able to communicate via the kick channel. See also page 64.

magic number

A magic number identifies a port. Two PIOC processes may communicate through ports. When a port is reserved it gets a number, a magic number, and it may also be given a name.

When a process wants to send a message to an other process, the message may be addressed with the magic number of the destination port. Or XROUT may find the destination process, if its port name is specified. A magic number is not reserved for a specific port.

mailbox

Mailboxes are locations in the consecutive area in the PIOC memory which is called the kick channel. They are used for ND-100 - PIOC communication.

A ND-100 process that wants to send information to a PIOC process, must send it to the mailbox. The PIOC process can then fetch the information from there. The mailboxes and the kick channel are described in detail on page 64.

message

Message is a term used in the XMSG system. A message is a block of data containing the information a PIOC process wants to send to an other PIOC process.

The message must contain both the text and the 'address' of the receiving process. The messages are transferred via ports, either directly, giving the destination process' magic port number, or via the routing program XROUT, giving the destination port name.

monitor call

In this manual, the term 'monitor call' is used when a ND-100 process uses the SINTRAN III monitor call PIOC (MON 255) to communicate with a PIOC process.

PIOC process calls to PIOCOS are named system calls or just calls.

physical level server

PHLS is the part of PIOCOS that handles input and output on the four communication lines of PIOC.

There is one PHLS for each line. The PHLS controls and supervises all data traffic between PIOC and the peripheral devices connected to it.

PIOCOS

ND-100 runs under the SINTRAN III operating system, the PIOC under PIOCOS operating system.

PIOCOS occupies the lower part of the PIOC memory. The purpose of PIOCOS is to manage all ongoing activities and to give assistance to all the running processes.

port

XMSG (and XROUT) can receive and send messages from and to ports. Communication between processes goes via their respective ports.

In XMSG you may refer to a port by the port name. In this case the message is routed via the XROUT program. Or you may refer to it by its magic number. The message is then sent directly from the sender's port to the destination port. The port number is used by the sending process to tell XMSG where the message comes from.

process

A program may only run in the PIOC if it is declared as a process or part of one. This means that the program must be given a dynamic (stack) and static data area, and an area holding administrative information about it (process description).

A process may consist of several programs and a program may be part of several processes. If the process description gets erased, the process is no longer alive. This means that PIOCOS no longer knows its name and where to find it in memory. More about processes is found on page 11.

process description

holds information for administrating the process. Only PIOCOS may access the process description, but the process itself or others may delete it.

Each PIOC process has such a description.

In this release of PIOC, version C, there is a maximum of 30 process descriptions process state

A process will always be in one out of four states, ACTIVE, SUSPENDED, DORMANT or DEAD.

When ACTIVE or SUSPENDED, the process is running: it uses processor time or it waits for some event to occur. When DORMANT, the process is passive. The fourth state, DEAD, is used about processes whose description has been deleted. The process then no longer exists.

The four states, and the calls that bring a process from one state to the other, is shown on page 12.

process type

There are two types of processes in PIOC, user and system processes.

The type reflects the purpose of the process. Some need more privileges than others to do a more supervising type job. Both types are declared with the 'create process' call to PIOCOS.

See user process and system process.

service point

Each of the four PIOC communication lines has a physical level server for transmitting and receiving data frames according to the HDLC format standard.

The physical level server consists of three service points. They handle line input, line output and supervision and control of the line. This is described in more details on page 46.

slot

A slot is a location in the PIOC kick channel. The slot is used for communication between processes, from PIOC to ND-100 or vice versa.

Each of the eight slots contains two mailboxes, one for information sent from ND-100, the other for information sent from PIOC. See also page 64.

system call

See call.

system process

System processes may bypass the memory protection scheme and thus access the whole PIOC memory but are not allowed to execute privileged instructions in the PIOC (MC68000) central processor.

I/O operations must executed by issuing calls to PIOCOS. See call.

timing scheduler

The timing scheduler is a process in PIOCOS which takes care of scheduling events for processes at given intervals or times.

The events may be set up at absolute or relative times, in seconds or basic time units (5 ms).

user process

User processes are the type of processes running in the PIOC which may be compared with the user written programs in the main computer system.

User processes are capable of executing calls to PIOCOS, like a user written program in ND-100 may execute monitor calls to SINTRAN III. The user process may, however, not execute privileged processor instructions and have no direct I/O access.

local XMSG

A subset of the XMSG system, which is an optional part of SINTRAN III, is included in PIOCOS.

This makes it possible for PIOC processes to communicate with each other through ports.

XROUT

See local XMSG.

word

One "word" in the ND-100 computer is 16 bits, ie., 2 bytes. 1Kword (1024 words) are 2Kbytes (2048 bytes).

3 PIOCOS

PIOCOS is a real time operating system, running in the PIOC microprocessor. It gives system assistance to PIOC processes, similar to SINTRAN III in ND-100. PIOCOS supports multiprogramming, and covers these main areas:

- Process initiation.
- Interprocess communication and synchronization.
- Time scheduling.
- Exception handling.
- Dynamic process control.
- ND-100 communication.

3.1 What is a Process

A process comprises a program, a process description and an associated data area. The process may share code with other processes but must have a unique process description and data area. The process description holds information for process administration, and may be accessed by the PIOC operating system PIOCOS.

A process is identified by a process name, given by the FCREATE function, and a process number returned by PIOCOS. Each process has an associated priority in the range of 1 to 15. 15 is the highest priority, 1 the lowest. Processes will be scheduled according to their priority.

Each process must have its own stack and data area. The user is responsible for providing sufficient stack area, by using the PLANC statement INISTACK, as the first statement in a main program.

This statement sets the highest address of the stack in register A7. Note that this register must not be used by the process for any other purpose.

This version of the PIOC Basic Software, release C, allows up to 30 process descriptions.

A process will always be in one of four states:

- ACTIVE (running/not running)
- SUSPENDED
- DORMANT
- DEAD

It is ACTIVE when the processor (MC68000) is allocated to it, or when it is waiting for processor time. All active processes are administered by the scheduler which is responsible for allocating the processor to the process having the highest priority.

A process is SUSPENDED when it is waiting for an event. When the event occurs, the process becomes active.

When a process is passive, it is in the DORMANT state.

A process is DEAD if its process description has been deleted. It can never be reached again and is of no use.

The illustration below shows how the various system calls bring a process from one state to another:

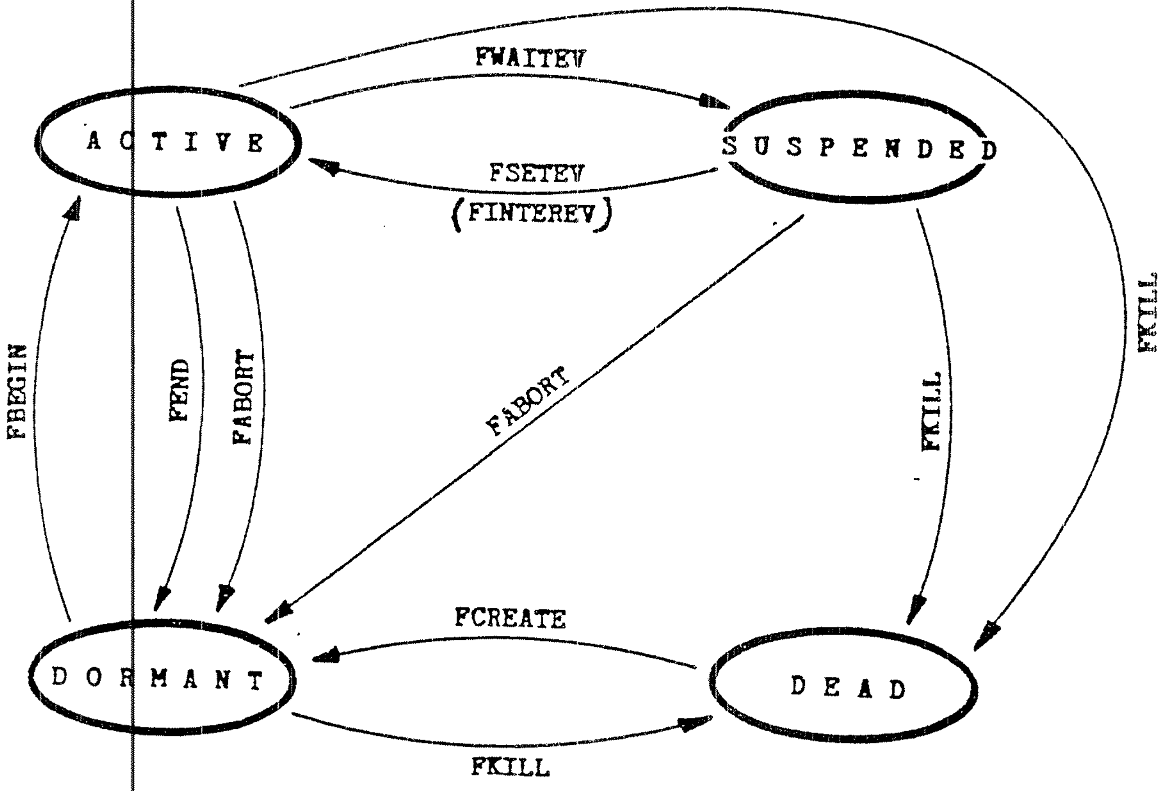


Figure 3. PIOC process state diagram.

3.2 The Process Hierarchy

Processes running in PIOC may operate on different privilege levels. User processes (PIOC processes) are allowed to do certain operations, while the processor (MC68000) has all privileges:

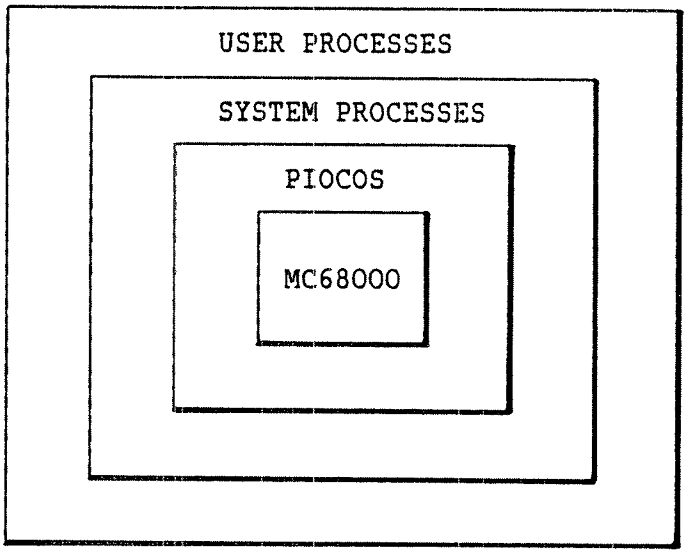


Figure 4. PIOC process hiearchy.

USER PROCESSES: Access to a limited part of the PIOC memory.
No direct I/O operations, must call PIOCOS.
Nonprivileged instructions only.

SYSTEM PROCESSES: Access to every part of PIOC memory and I/O addresses.
No direct I/O operations, must call PIOCOS.

PIOCOS: Full memory access.
Unlimited I/O access.
May execute all privileged instructions.

MC68000: All possible privileges.

3.3 System Calls to PIOCOS

The PIOC processes request system assistance by system calls. They are carried out by a MC68000 TRAP 2 assembly instruction.

The microprocessor MC68000 has 16 internal 32 bits registers: A0 through A7 and D0 through D7. System calls to PIOCOS make use of two of them: A0 and D0.

All system calls require the call number to be in D0 register. On return, D0 contains a return code.

If the call executes successfully, D0 receives U1OK (*) from PIOCOS. All D0 values different from U1OK indicates an error situation.

Appendix A, on page 127, contains a list of all possible return (error) codes, both the symbolic names and the numeric values.

The register A0 is used by most system calls. Most calls require A0 to hold a parameter block address. The following example illustrates how to execute system calls from a PIOC program:

```
%       The routine 'CALL_PIOCOS' executes any PIOCOS system call
$INCLUDE PIOC-FUNCVAL-C:DEFS
ROUTINE VOID,VOID (FCODE,INTEGER POINTER,INTEGER):&
        CALL_PIOCOS (CALLNO,PARADDR,CSTATUS)

$*      MOVE.W   CALLNO,D0         %  Start of MC68000 assembly
$*      MOVEA.L  PARADDR,A0        %
$*      TRAP     #2                %
$*      MOVE.W   D0,CSTATUS        %  End of MC68000 assembly

        IF CSTATUS >< U1OK THEN              % This error test
             OUTPUT (1,'A','$ERROR : ')      % should be expanded
             OUTPUT (1,'I3',CSTATUS)         % or modified to suit
        ENDIF                                % your preference
ENDROUTINE
```

Example 1. Routine to execute PIOCOS system call.

(*) Each call and error code has symbolic names, with corresponding numbers. The file PIOC-FUNCVAL-Cxx:DEFS, shown in appendix A on page 127, contains these names. In this manual we use these symbolic names rather than numbers, which makes the programs easier to read.

Each call is described separately using the following syntax:

Syntax	Explanation
DO : call name	Register DO contains a number identifying the call name, and DO receives a status code from PIOCOS when the call has been executed. A list of call names and their call numbers is found in appendix A.
AO ----> par.1 % 2 bytes	Register AO contains the memory address of the first parameter (number 1) in the set of parameters. Parameter 1 is 2 bytes long.
=par.2 % 4 bytes	Parameter 2 (4 bytes) is found at memory address AO+2 bytes. The equal sign shows that the parameter is returned from PIOCOS after the call has been executed.
par.3 % 4 bytes	Parameter 3 (4 bytes long) is at memory address AO+2+4 bytes.
. . . . n bytes	
par.n % 2 bytes	Parameter n (2 bytes long) is at memory address AO+2+4+ n bytes.

3.4 Process Creation and Modification

FCREATE - creates a new process description.

The call is used to make a process executable. That means to tell PIOCOS what to name it, where to find it and at which priority to run. The call is used to create both ordinary PIOC processes and PIOC system processes. The system processes are used for special purposes, such as controlling devices connected to PIOC.

```
                DO : FCREATE        (DO holds the FCREATE call no.)

AO ----->       =PROCESS NO                  % 2 bytes    (given by PIOCOS)
                PROCESS NAME                 % 4 bytes    (given by user)
                PROCESS TYPE / PRIORITY      % 2 bytes    (given by user)
                START ADDRESS                % 4 bytes    (given by user)
```

PROCESS NO is an integer variable that is used for later reference to a process. It is returned from PIOCOS if the call executes successfully. Other system calls need this parameter as an input to be able to identify the desired process.

PROCESS NAME is in some cases also used for later reference. It allows greater flexibility when programming multiprocessing systems.

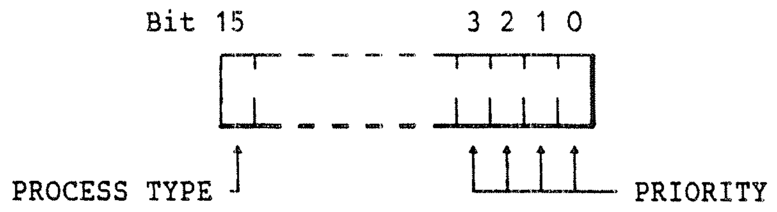


Figure 5. Process TYPE and PRIORITY format.

PROCESS TYPE determines whether this is a PIOC user process or a PIOC system process. A '1' in the most significant bit of the most significant byte indicates a system process, a '0' a user process.

PRIORITY reflects the importance of the process compared to the other processes. Priority ranges from 1 to 15, and is set with the four least significant bits in the least significant byte.

START ADDRESS is the memory address where the process' program starts.

The process is set in the dormant state and may be made ready for execution with the FBEGIN call.

On return, DO contains the status of the operation.

FBEGIN - makes a dormant process ready for execution.

This call puts the dormant process into an active state, ie., makes it ready for execution. When the scheduler allocates the processor to it, execution starts at START-ADDRESS given in the FCREATE call.

```
                    DO : FBEGIN

        AO ---->     PROCESS NO          % 2 bytes
```

On return, DO contains the status of the operation.

FEND - the 'normal termination' call.

If a process executes the FEND call, it enters a dormant state. This is called a 'normal termination'. Any process may execute the FBEGIN call to start the terminated process again.

```
                    DO : FEND
```

On return, DO contains the status of the operation.
Any port opened in the ND-100 XMSG system is closed.
(See page 76).

FABORT - forces a process to terminate.

This call forces the process with the corresponding PROCESS NO to terminate, ie., makes it dormant. Another process may execute the FBEGIN call to start the aborted process again.

```
                    DO : FABORT

        AO ---->     PROCESS NO          % 2 bytes
```

On return, DO contains the status of the operation.
Any port opened in the ND-100 XMSG system is closed.
(See page 76).

FKILL - deletes the description of a process.

The call deletes the process description of the specified process. This brings it into the dead state. The process is then no longer recognized by PIOCOS and cannot be run. A new process description must be created (with the FCREATE call) before the process can be started.

```
                DO : FKILL

      AO ---->      PROCESS NO          % 2 bytes
```

On return, DO contains the status of the operation.
Any port opened in the ND-100 XMSG system is closed.
(See page 76).

3.5 Process Identification

Each process is identified by a name and a number. The name is specified by the user, while the number is returned from PIOCOS when the process is declared with the FCREATE call. The following calls may be used if a process wants to identify itself or another process, knowing the process name, but not the process number or vice versa.

FWHOAMI - gives a process its process number.

The call is used by a process to identify itself.

```
                    DO : FWHOAMI

        AO ---->    =PROCESS NO         % 2 bytes
```

PIOCOS returns PROCESS NO to the requesting process.

On return, DO contains the status of the operation.

FPROSNO - returns process number if the process name is known.

This call may be used by a process that wants to know the process number of another process, of which it only knows the name.

```
                    DO : FPROSNO

        AO ---->    =PROCESS NO         % 2 bytes
                    PROCESS NAME        % 4 bytes
```

On return, DO contains the status of the operation.

FPRNAME - returns process name if the process number is known.

This call may be used by a process that wants to know the name of another process, if it only knows the process number.

```
                    DO : FPRNAME

        AO ---->    PROCESS NO          % 2 bytes
                    =PROCESS NAME       % 4 bytes
```

On return, DO contains the status of the operation.

3.6 Process Synchronization

The process synchronization is administered by the scheduler. The scheduler may set up EVENTS for a PIOC process. An event influences the process only if it receives the expected EVENT.

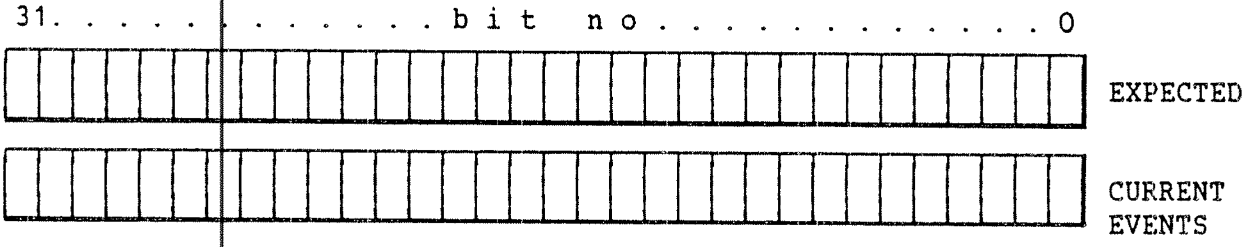

31. . . . . b i t n o 0
EXPECTED
CURRENT EVENTS

Figure 6. Process description registers.

The process description contains two 32 bits registers for storing information about the expected events and the actual events occurring. A process may state the type of event it wants to react to by setting bits in the one register, while events occurring causes bits to be set in the other register. As events occur, the event bits are OR'ed with previous event bits. An event will always cause a comparison of the two registers. They are AND'ed. If the AND operation results in one or more '1's, the event leads to a 'wake-up' signal (kick) for the suspended process.

Events reserved by XMSG

The following two events are reserved for communication between PIOCOS and a user process:

NXMEVEN - bit 31 is used by the ND-100 XMSG system, for tasks (processes) running in the PIOC which communicates with tasks running in ND-100. (See ND-100 XMSG, on page 76.)

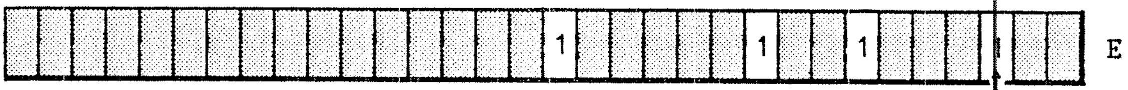
XMEVENT - bit 30 is used by the PIOCOS XMSG system, for tasks which communicates with other tasks in the PIOC. (See local XMSG, on page 32.)

The other bits 29 - 0 can be used for application oriented events.

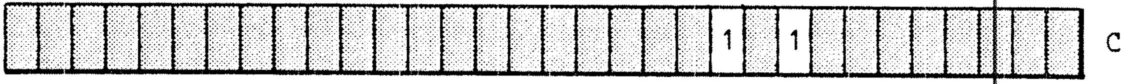
When a process sets up the expected bit mask, it brings itself into a suspended state. Events may occur, but the process remains suspended until an expected event occurs. This makes the process active again. The next illustration shows what may happen in a typical situation:

E = Expected bits, C = Current event bits, blank fields mean logical zero, 1 means logical one.

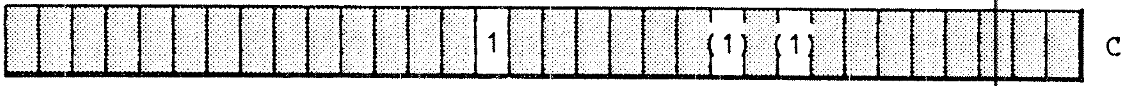
1. A process has set up the following expected event bits, and enters a suspended state.



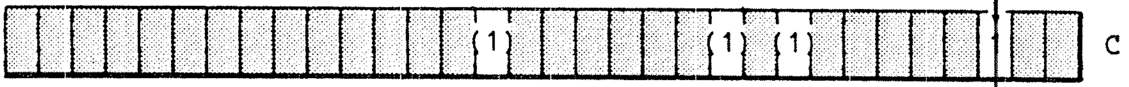
2. An event occurs with the following bit pattern. The process remains suspended, because E.AND.C is zero.



3. An event occurs with the bit pattern below. (1) from prev. event. The process remains suspended, because E.AND.C is zero.

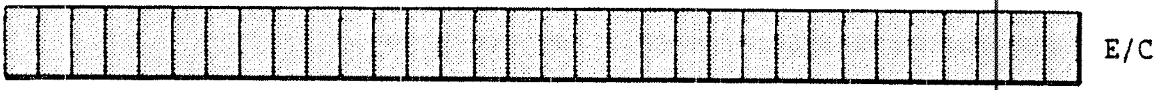


4. An event occurs with the bit pattern below. (1) from prev. event. E.AND.C is now TRUE (bit 3 set in both registers) and the event causes the process to be activated.

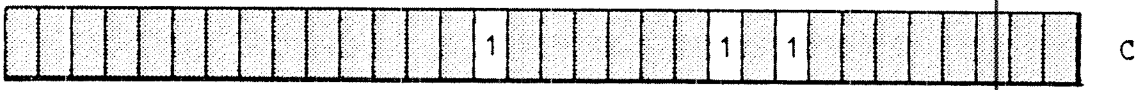


5. The 'expected' bit pattern is cleared. If the process wants to be controlled by an event again, a new bit pattern must be set up.

For the FWAITEV function also the 'current' pattern is cleared:



For the FSELWAIT function only the bits marked by 'expected' are cleared in the 'current' pattern:



The least significant bit of the event bits set is used for communication between PIOCOS and the user process. When a process requests a resource (ie., buffer, I/O, semaphores) from the PIOCOS, it is notified that the request is being fulfilled by the generation of events by PIOCOS. In this way a process may request multiple resources and use the FWAITEV call (described below) to remain in the suspended state until a suitable event set occurs.

Four calls are used for setting up, waiting for and testing events:

FWAITEV - sets up the expected event bit pattern.

This call sets up the expected event bits, and the process enters the suspended state, waiting for an event with the corresponding bits. Each time an event occurs, the event bits are tested against the expected event bits.

```
                DO : FWAITEV

     AO ---->   =CURRENT EVENT BITS  % 4 bytes
                EVENT BITS           % 4 bytes
```

EVENT BITS are the 4 bytes containing the bits the process shall react upon. When an event occurs and if one or several of the expected event bits are present, "=CURRENT EVENT BITS" is returned from PIOCOS and cleared in the process description. "=CURRENT EVENT BITS" reflects all the events that have occurred since the call was executed.

On return, DO contains the status of the operation.

FSELWAIT - sets up the expected event bit pattern.

This call is identical to FWAITEV except for the fact that only the corresponding bits of "=CURRENT EVENT BITS" and "EVENT BITS" are cleared in the process.

```
                DO : FSELWAIT

      AO ---->  =CURRENT EVENT BITS   % 4 bytes
                 EVENT BITS           % 4 bytes
```

EVENT BITS are the 4 bytes containing the bits the process shall react upon. When an event occurs and if one or several of the expected event bits are present, =CURRENT EVENT BITS is returned from PIOCOS. =CURRENT EVENT BITS reflects all the events that have occurred since the call was executed.

On return, DO contains the status of the operation.

FSETEV - sets up an event for a process.

This call is available for processes and drivers, and is thus the only call allowed for exception handlers.

EVENT BITS will be OR'ed with the current event bits of the process. If the process is currently waiting for the corresponding event bits, it will be activated.

```
                    DO : FSETEV

          AO ----> PROCESS NO          % 2 bytes
                   EVENT BITS          % 4 bytes
```

PROCESS NO is the identification of the receiving process.

EVENT BITS are the bits following this event.

On return, DO contains the status of the operation.

FREADEV - reads the current event bits.

Before a process executes the FWAITEV call, it may use the FREADEV call to read the bit pattern of the process' current event bits.

```
                    DO : FREADEV

          AO ----> =EVENT BITS          % 4 bytes
```

Current event bits are returned in the =EVENT BITS parameter. If the call executes successfully, the current event bits register is set to zero.

On return, DO contains the status of the operation.

3.6.1 Time Controlled Monitoring

FINTEREV - schedules periodical events.

This call causes the timing scheduler to set up a future event and if desired, to repeat it periodically for the actual process. It only affects processes in ACTIVE or SUSPENDED state.

```
                        DO : FINTEREV

            AO ---->    PROCESS NO          % 2 bytes
                        EVENT BITS          % 4 bytes
                        INTERVAL            % 4 bytes
                        TIME                % 4 bytes
                        UNIT-SIZE           % 2 bytes
```

PROCESS NO is the identification of the process receiving the event.

EVENT BITS gives the bits following this event.

INTERVAL is used to distinguish between two operations:

- INTERVAL = 0 means: schedule one event at the time specified in the TIME parameter.
- INTERVAL > 0 means: schedule periodical events with intervals INTERVAL. The first interval is at the time TIME (relative or absolute determined by UNIT-SIZE).

TIME is used for giving the time of the first event (relative or absolute determined by UNIT-SIZE).

UNIT-SIZE gives the type unit for the TIME and INTERVAL parameters.

UNIT-SIZE = 0:	basic time units	absolute
UNIT-SIZE = 1:	seconds	absolute
UNIT-SIZE = 2:	basic time units	relative
UNIT-SIZE = 3:	seconds	relative

basic time unit = 5 ms

On return, DO contains the status of the operation.

FINTERDEL - stops sending the events initiated with FINTEREV.

This call is the opposite of FINTEREV, it stops the periodical events, if any.

```
                    DO : FINTERDEL

          AO ----> PROCESS NO          % 2 bytes
                    EVENT BITS          % 4 bytes
```

EVENT BITS must match the bit pattern of the periodically scheduled events previously set up with the FINTEREV call. If several periodical event sets are scheduled for the actual process, only the one with the matching bit pattern is deleted. However, if EVENT BITS=0, all periodical events scheduled for the process are deleted.

On return, DO contains the status of the operation.

3.7 How to Synchronize Process Execution Through EVENTs

The purpose of this section is to show how two PIOC processes may set up events for each other, and thereby synchronize their activities. This is not a complete example, rather an explanation of how to use the calls explained so far in this chapter.

The PIOC process DONA(LD) is started manually from the PIOC-MONITOR. It has priority 1 (default). It creates the process SNOO(PY), gives it priority 5, starts it, and they start to communicate.

The diagram below shows the necessary calls in both processes. The syntax for this pseudo program is:

```
     xxx --> PARAMETER-1 : the process puts a value in this parameter.
     yyy --> PARAMETER-2 : the process puts a value in this parameter.
    <CALL>               : the process executes the < > function.
     PARAMETER-3 <== zzz : PIOCOS gives this param. a value on return.
     PARAMETER-4 (pp)    : The parameter has the value (pp).
PRINT "some text"        : Explanation of what is going on.
```

```
 activity in               activity in              activity in
 DONA                        PIOCOS                        SNOO
-----------------------------------------------------------------

    <WHOAMI>                                  SNOO's program code is
     PROC_NO_DON       < ======  2            in memory at address
                                              105000B. But it is not
     PRINT "I think I'll                      declared as a process,
            create an                         and it is not running.
            other process"

       'SNOO' --> PROC_NAME
            5 --> PROC_PRI
     105000B  --> START_ADDR.
    <CREATE>
     PROC_NO_SNO       < ======  3

     PRINT "I think I'll
            start it
            as well"

     PROC_NO_SNO (3)
    <BEGIN>
```

```
                                              PRINT "Good Morning.
                                                     Wonder who I am?"

                                                  <WHOAMI>
                              3  =====  >          PROC_NO_ME

                                                   PROC_NO_ME (3)
                                                  <PRNAME>
                         'SNOO' =====  >           PROC_NAME

                                              PRINT "Hey, I'm "
                                              PRINT PROC_NAME

                                              PRINT "I'll do some
                                                     work"

                                                         .
                                                         .
                                                         .

                                               PRINT "I'm going to
                                                      sleep waiting
                                                      for event bits
                                                      0 and 2."

                                                  5 --> EVENT_BITS
                                                 <WAITEV>


PRINT "Better ask
       SNOO to start
       working again.
       Wonder what
       event will  wake
       him up? I'll try
       ten different
       events."

     PROC_NO_SNOO (3)
     1 --> EVENT_BITS
    <SETEV>
             ...0001 ======>     TEST AGAINST
     ADD 1 TO EVENT_BITS         5 (...0101).
     IF EVENT_BITS =9            IF 0 OR 2 THEN --->  PRINT "Someone gave
        THEN                                                 me a wakeup
     ELSE GOTO                                               signal"

                                                      PRINT "I don't want
                                                             to run
                                                             anymore..."

                                                      PRINT "Goodbye."

                                                     <END>

                                                     (SNOO stops)
```

```
PRINT "If SNOO woke
       up during
       EVENT_BITS
       between 1 and
       9, I have to
       execute the
       rest of the
       loop now.
       (SNOO has
       higher priority
       than me)"

PRINT "I'm tired of
       this game, and
       SNOO makes me
       sick. I'll kill
       him"

     PROC_NO_SNO (3)
    <KILL>                                        (SNOO gets killed)

PRINT "Goodbye."

    <END>

     (DONA stops)
```

Example 2. Synchronizing process execution through events.

3.8 Exception Processing

Exception Processing is associated with interrupts, trap instructions, tracing or other exceptional conditions.
PIOCOS distinguishes between two types of exceptions: Asynchronous and Synchronous:

- Asynchronous exceptions are caused by interrupts from devices connected to PIOC or communication line errors.

- Synchronous exceptions are caused by a MC68000 assembly instruction or during execution of such an instruction.

3.8.1 Asynchronous Exceptions

When an exception occurs, an Asynchronous Service Routine (ASR) will be activated. You must write one ASR for each possible interrupt type and each interrupt must be connected to a proper ASR by a system call.

The Asynchronous Service Routine always executes in supervisor state, using the processor supervisor stack. The service routine is capable of executing privileged instructions. An interrupt service routine may be interrupted by ASRs with higher hardware priority than itself.

"SET EVENT" (FSETEV) is the only call to PIOCOS that may be included in an asynchronous service routine. The routine is responsible for saving and restoring all used registers and must be terminated with the RTE assembly instruction. To write such routines, you need a good understanding of the MC68000 instruction set.

FCRDRV - declares an Asynchronous Service Routine.

Asynchronous Service Routines (ASR's) must be part of a system process, and are declared as follows:

```
                DO : FCRDRV                 % "CReate DRiVer"

        AO ---->  VECTOR NUMBER             % 2 bytes
                  ASR ADDRESS               % 4 bytes
```

The ASR ADDRESS is the address of the ASR routine which is activated when an interrupt comes from the specified VECTOR NUMBER. (See appendix C on page 137.)

If an interrupt occurs with no driver created for this interrupt, the asynchronous exception is handled like a synchronous exception. (See next section).

3.8.2 Synchronous Exceptions

Synchronous exceptions arise either from the processor recognizing abnormal conditions during execution of processor instructions, or from the use of instructions whose normal behaviour is trapping.

Synchronous exceptions are caused by:

- The instructions TRAP (trap), TRAPV (trap on overflow), CHK (check register against boundaries) or DIV (divide).
- Illegal instructions.
- Word fetch from odd addresses.
- Privilege violations.

When synchronous exceptions occur, PIOC-MONITOR-C will usually write a message to the user's terminal, in which it specifies what exception and where it occurred:

```
PIOCOS aborted because of <error-message>  <exception-no>
at address: <zzzzzzz>
```

Figure 7. PIOCOS exception message

A list of exception numbers and the corresponding message can be found in appendix C, on page 137. If the exception number is outside range of numbers in the list, the message issued is:

'Undefined interrupt/exception occurring'

PIOCOS is then aborted, which means it is not possible to continue the job. Especially is the result of some PIOC-MONITOR commands no longer defined (you will get a message like "PIOC-N100-driver not ready").

However, there is a function in PIOCOS to define your own handler for such exceptions. It should be used by very experienced PIOC programmers only.

FTRAPH - declares a traphandler routine

```
              DO : FTRAPH            % define a trap handler

   AO ----> TRAP-HANDLER ADDRESS    % 4 bytes
```

The trap handler has to handle <u>all</u> exceptions, which will otherwise cause the above mentioned message.

Note that fatal errors (see below) will not call the trap handler.

On the stack, the trap handler will find:

- the trap vector number (2 bytes). Note that most of the unused traps/interrupts have a common number (256D).
- an address (4 bytes) where the exception occurred (or about where - depends on trap number)
- a return address (4 bytes). Note, that a return with this address will cause the above mentioned message and abortion of PIOCOS

3.8.3 FATAL ERRORS

Fatal errors use the same mechanism to write a terminal message as the exceptions mentioned above. However they can not be handled by a trap handler.

They occur when PIOCOS detects an software error, which can not be handled in a usual way (e.g. errors in the startup phase of PIOCOS).

The written "trap vector number" is in this case usually one of the errors from PIOC-FUNCVAL-C:DEFS.

In some cases only the address will give sufficient information about the error. The address will then point into the PIOCOS area. In such a case, please contact ND Technical Support.

3.9 Local XMSG and XROUT Message Transfer and Routing Systems

A subset of the XMSG, TASK-TASK COMMUNICATION SYSTEM of SINTRAN III, is implemented as a part of PIOCOS. This allows processes executing within the same PIOC to exchange data between them.

There is also a "remote" XMSG system which allows PIOC processes to exchange data with processes running in other PIOC's or in the ND-100, the system calls for these are described on page 76.

A more detailed description of XMSG is found in SINTRAN III Communication Guide ND-60.134.

Processes in PIOC communicate with each other and synchronize their activities by means of messages which are transmitted and received through ports.

A message is a block of data occupying an area of consecutive locations in memory called a message-buffer. The sending process normally opens a port, reserves message-buffer space and transfers its data via its port into the buffer. The message may then be received and read by the respective process.

A process may own many ports, but two processes may never own the same port. Ports are identified by a Port Number, Port Name or Magic Number.

- Port Number is returned to the process when the call 'open port' (XFOPN) is used. This is a unique number.

- Port Name is given a port as soon as the name (in ASCII) is sent as a message to the system process called XROUT, which is the message administrator in XMSG.

- Magic number is returned when the call 'get message status' (XFMST) is used by a process receiving a message, or when the call 'receive next message' (XFRCV) is used by a process when it is ready to handle a new request.

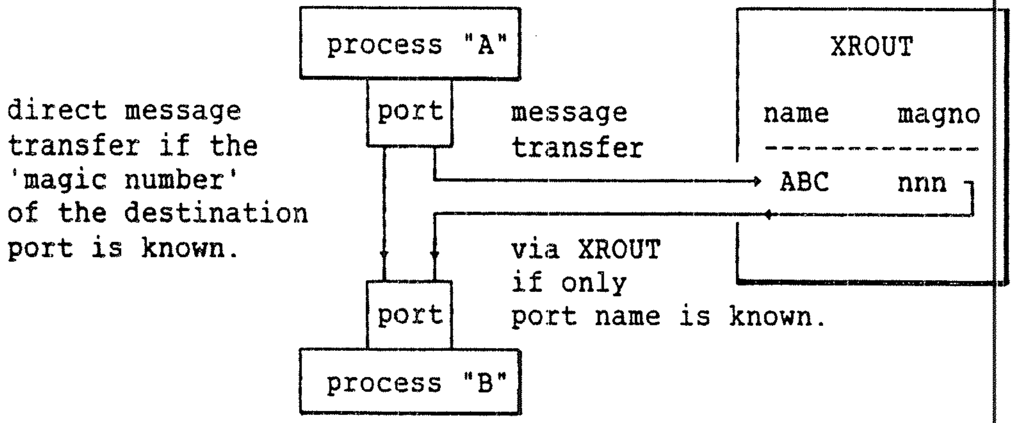


Figure 8. XMSG Port to Port Communication.

When a process wants to send a message to another process, a message containing both the text and the destination port name must be sent. XROUT will then forward a signal to the receiver to tell this process that there is mail for it. When the message is asked for by the receiving process and passed to its port, it can then ask for the sending port's magic number. Messages can be sent directly to the destination port using its magic number, if it is known.

The system call FXMSG may perform many functions. The function is specified as the first parameter pointed to by the A0-register. Some functions may again act upon the chosen option. The option value is OR'ed with the function value. The standard syntax is:

```
                    D0 : FXMSG

        A0 ---->    FUNCTION  OR <OPTION>           % 2 bytes
                    function dependent parameters
                    .
                    .
                    .
```

On return, D0 contains the status of the operation.

The OR-operation is performed as follows:

```
      TRUE =: BIT (FUNCTION,OPTION)

Example:  5B =: FUNCTION                    %     ...000101
         20B =: OPTION                      %     ...010000
        TRUE =: BIT (FUNCTION,OPTION)       %     ...010101
       OUTPUT (1,'I2',FUNCTION)

Result : 25B
```

3.9.1 Reserving and Releasing Ports

The following functions controls the creation and/or deletion of ports:

XFOPN - opens a port.

```
                   DO : FXMSG

        AO ---->   XFOPN  OR <OPTION>        % 2 bytes
                   =PORT                     % 2 bytes
```

When a process asks for a port to be opened, the port number is returned in the =PORT parameter.

The following open option may be OR'ed to XFOPN:

XFPRM - Permanent open. This option is used if the port is to be opened permanently. This means that it only can be closed with an explicit close or with the CLOSE parameter set to -2. See the function XFCLS below.

XFCLS - closes ports.

The XMSG message system allows a specific port, all temporary ports or all ports to be closed.

```
                   DO : FXMSG

         AO ---->  XFCLS                % 2 bytes
                   PORT                 % 2 bytes
```

PORT is used to specify the number of the port to be closed.

If PORT = -1, all temporary (non permanent) ports are closed.
If PORT = -2, all ports are closed.

When a port is closed, all secure messages currently queued on that port are set to 'return' and 'non-secure' and returned to their senders. All 'non-secure' messages are released. Secure messages are described on page 38 (send current message).

3.9.2 Getting and Releasing Message Buffer Space

XFGET - gives the process message buffer space.

A process may reserve certain message buffer space:

```
                   DO : FXMSG

        AO ---->   XFGET               % 2 bytes
                   SIZE / =MESADDRESS  % 4 bytes
```

Buffer-space is allocated on a per task basis, and a total of the current allocation is kept. Requests for more space than the limit allows will not be honoured, but a process can temporarily exceed this limit as a result of messages being sent to it (since this makes it the owner of the message buffer).

Only the process owning the buffer-space is allowed to read from it and write to it, send it to someone else, or release it.

The XMSG system remembers how much useful data there is in a message and puts this into a parameter. This parameter is called the <u>message length</u> (don't confuse with buffer size) and is initially set to zero. After each write operation the current length is compared with the previous length, so no message characters are lost:

```
LEN:=MAX(LEN,INDX+1)
```

When the last byte of a message is read, an XMSG flag called 'whole-message-read' is set. This causes the message length parameter to be set to zero just before the next write operation is done.

XFREL - releases message buffer-space.

A process may release the message buffer-space it owns:

```
                   DO : FXMSG

        AO ---->   XFREL               % 2 bytes
                   SIZE / =MESADDRESS  % 4 bytes
```

3.9.3 Set Current Message for a Process or for a Port

Since many system calls implicitly operate on the current message, it is useful to be able to set the latter.

```
                    DO : FXMSG

        AO ---->    XFSCM                 % 2 bytes
                    MESADDRESS            % 4 bytes
                    PORT                  % 2 bytes
```

The specified message is set as current for the specified port. If the PORT = 0, the message is set as current for the process.
If MESADDRESS = -1, the current message of the process is set as the current message of PORT.

3.9.4 Reading and Writing Messages

XFREA - reading a message.

Reading the message is performed by using

```
                   DO : FXMSG

         AO ----> XFREA                % 2 bytes
                   UADD                 % 4 bytes
                   ULEN / =NBYTES       % 2 bytes
                   DISP                 % 2 bytes
```

Data is read from the current message starting with displacement DISP (rounded up to the next word boundary) into the user buffer specified by UADD (length ULEN). NBYTES is set to the actual number of bytes read.

If DISP = -1, the reading starts from the last character read in the previous read operation.

If the last byte in the message is read, the 'whole-message-read' flag is set, and the next XFWRI (Write message) will reset the current message length.

Note that the displacement (DISP) must always be rounded up to the next word boundary.

XFWRI - writing a message.

Data is transferred from the user space to a message by calling:

```
                   DO : FXMSG

         AO ----> XFWRI                % 2 bytes
                   UADD                 % 4 bytes
                   ULEN / =NBYTES       % 2 bytes
                   DISP                 % 2 bytes
```

If the 'whole-message-read' flag has been set, it will be reset, and 'current length' is set to 0 before any writing is carried out. If the displacement (DISP) is -1, a value equal to the current message length is assumed, providing an append call. If the displacement is odd, 1 is added, and a zero byte is inserted in the message. If DISP+NBYTES is not greater than the message size, ULEN bytes are copied from UADD into the message buffer and the message length is increased to DISP+NBYTES (+1 if DISP is odd). NBYTES returns the actual number of bytes transferred.

3.9.5 Sending And Receiving Messages

XFSND - send current message.

When a process wishes to send a message to another process it calls

```
                    DO : FXMSG

            AO ----> XFSND  OR  <OPTION>        % 2 bytes
                     MAGNO                      % 4 bytes
                     PORT                       % 2 bytes
```

The default message is sent from the local port (PORT) to the destination port (MAGNO).

The following send options may be OR'ed to XFSND:

XFSEC - Secure message. The message will be returned to the sending port if it cannot be delivered or if the handling program terminates while the message is 'current'.

XFHIP - High-priority message. It will be chained to the head of the receiver's queue instead of the tail. If there are other priority messages in the queue, it will be inserted after them. The receiver will be informed of this when he executes his next receive (XFRCV).

XFFWD - Forwarding. The sender information in the message will not be updated so that the receiver will be informed that the message was sent from the previous sending port.

XFROU - Ignore the MAGNO parameter and send the message to XROUT. The message contents must then be in a form comprehensible to XROUT. The rules for this are described on page 41.

XFBNC - Bounce-message. The message sent by the receive call XFRCV is returned to the sender instead of being received.

If MAGNO = -1, the message will be sent back to the port from which it was last sent.

XFRCV - receive next message.

When a process is ready to handle the next request, it calls

```
                   DO : FXMSG

        AO ---->   XFRCV OR <OPTION> / =TYPE       % 2 bytes
                   PORT                            % 2 bytes
                   = MESADDRESS                    % 4 bytes
                   = NBYTES                        % 2 bytes
                   = MAGNO                         % 4 bytes
                   = DATA ADDR                     % 4 bytes
```

If a message is waiting at the specified port (PORT), it is received (unchained from the message queue) and MAGNO contains the magic number of the sending port. MESADDRESS contains the address of a status block for the message and NBYTES contains the message length in bytes. DATA ADDR points directly to the memory address of the user data part of the message.

The message type is returned through the TYPE parameter.

The following receive options may be OR'ed to XFRCV:

XFWTF - Wait for message. If no message is waiting at the port, the task is delayed until a message appears at the port.

XFWAK - Wake. If no message is waiting at the port, the next message that arrives at the port leads to an EVENT with event bit 30 being set up for the process owning the port. Events were discussed in detail on page 20.

This phenomenon allows timed-out waits to be executed: When the EVENT is set up, the message has not yet been received, so the receive function (XFRCV) must be repeated. This 'wake' option can be enabled on more than one port at a time, and will cause all wake requests for the process to be cleared when the event comes.

If neither of the options are specified and there is no message waiting at the port, DO reflects this.

A successful XFRCV leads to the returned message becoming the current message for the process and the port. If the message is secure (XFSEC option in the XFWRI function), and if the process aborts before the current message is cleared, the message will be returned to the sender, with the status reflecting the reason.

The 'current message' is cleared by:

- releasing or sending it to someone else, or
- receiving another secure message.

3.9.6 Message Status

XFMST - extract senders's magic number.

A process may extract the sender's magic number from a received message:

```
                    DO : FXMSG

        AO ---->    XFMST                   % 2 bytes
                    MESADDDRESS             % 4 bytes
                    = MAGNO                 % 4 bytes
                    = NBYTES                % 2 bytes
```

It may be argued that this requires an extra call, but

1) One often just sends messages back to the senders (MAGNO=-1).

2) Otherwise one can read the magic number once, and after that use the port information returned by XFRCV to identify the sender.

3.9.7 XROUT - Routing and Service Task

As mentioned earlier, XROUT is a special system process that allows processes to find each other by providing a port naming scheme.

If XROUT handles a request from process A to get in touch with process B, the following restrictions must be considered:

XROUT will not give A's port number to process B, but transmits the message from A along with its port's magic number to B. Process B can then look at A's message and 'call' back if it wants to, and even give A its own magic number.

XROUT is implemented as a standard PIOC process, and is called as an option from the XMSG call 'send current message' (XFSND).

3.9.7.1 XROUT Message Formats

Messages are sent to XROUT using the XFROU option of the XFSND call in XMSG. XROUT offers many services. The type of service is specified in message byte 1.

Byte 0 - A serial number returned unchanged by XROUT in order to allow processes to have many requests outstanding.

Byte 1 - The service number (symbol XSxxx) of the service being requested / =Status of the operation.

Byte 2,3 - Length of the remainder of the message in bytes.

Byte 4 - ...parameters.

Each parameter has the form:

Parameter byte 0 (4) - Parameter number (0 is null). Integers have positive values, strings negative.

Parameter byte 1 (5) - Length of parameter in bytes

Parameter byte 2..(6..) - Parameter data.

The parameter number is dependent of the service. Parameters not specified for a given service assume default values. All parameter blocks start on even byte boundaries in the message.

In general the following parameter types are used:

Integer — Integers will be treated as signed, so that the sign bit will be extended if necessary. This allows the sender to decide how much space in the message he wishes to use for an integer, and allows the user to take appropriate action when receiving.

ASCII strings — The length is defined by the parameter length byte. If a fixed length record is desired, the record will be filled up with blanks.

3.9.7.2 Type of Services

XSNUL - returns a null status message

XROUT returns a message of two bytes containing the reference number. No parameters.

XSLET - send letter

This service is used to find a remote port. The only parameter used by XROUT is:

Identifier (type string)

XROUT extracts the identifier and looks up the string in its name table. If a match is found, the whole message is forwarded (call XFFWD) to the matching magic number. If no match is found, the call code is set to an error value, and the message returned to the sender.

XSNAM - name a port

In order for a port to get a name, the name must be declared to XROUT. This is done by the XSNAM service with one parameter:

Identifier (type string)

XROUT looks up the name in the name table. If it is present, an error message is returned, otherwise the name is entered, and the sending port's magic number is included.

XSCNM - clear name

When the validity of a port name has expired, the clear name service removes the specified name from the name table.

XSGNM - get name of a port

Any process can get the name of a given port by sending a message containing the magic number (integer) as parameter 1. The return message will contain the port name appended as parameter 2 (type string), if there is enough space in the message.

4 PHLS, THE PHYSICAL LEVEL SERVER

The Physical Level Server provides the means to transmit and receive data on the serial communication lines in a PIOC.

There is one Physical Level Server (PHLS) for each communication line in PIOC.

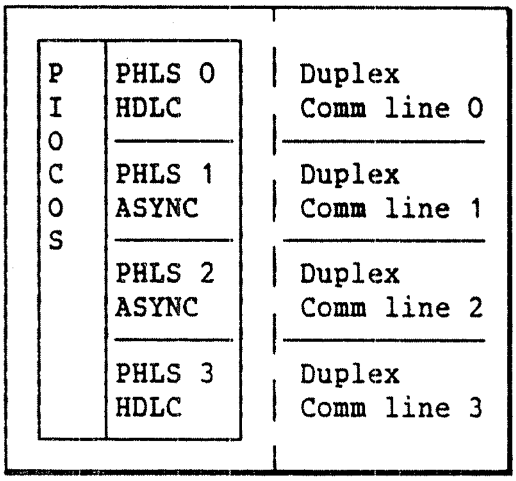


Figure 9. Physical level servers in PIOC.

Each of the four communication lines may provide service in the HDLC or ASYNCHRONOUS mode. The type of PHLS that corresponds to a particular communication line is selected in the PIOC Basic System loading, where the installation generates a PIOCOS operating system tailored to its requirements. Details of this step is described on page 96.

During the initiation of a communication line, an automatic check is made against the defined configuration. If it does not conform, an error status will be returned.

4.1 The Service Points of a Physical Level Server

Each PHLS has three service points: One service point CONTROLS AND SUPERVISES the communication line, the second RECEIVES data, and the third TRANSMITS data.

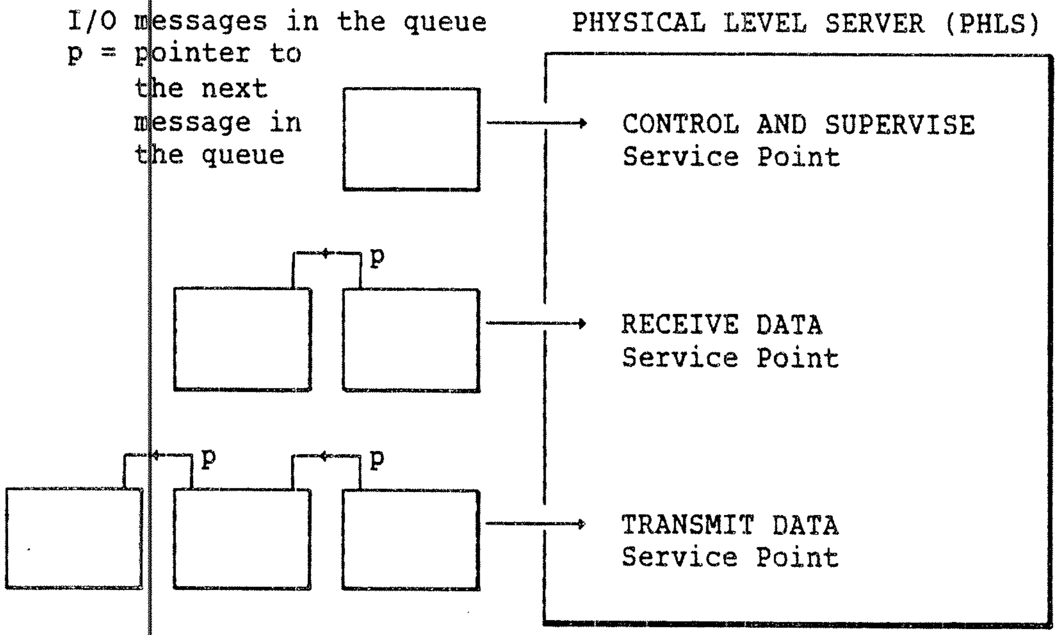

I/O messages in the queue
p = pointer to the next message in the queue
PHYSICAL LEVEL SERVER (PHLS)
CONTROL AND SUPERVISE Service Point
p
RECEIVE DATA Service Point
p
p
TRANSMIT DATA Service Point

Figure 10. Service points in the PHLS.

When activating a PHLS, the request must be addressed to the proper service point. For example: data to be transmitted must be given to the output service point.

Each PHLS may be active at all three service points simultaneously. Requests to and from a service point are passed through a queue of I/O messages. The PHLS must be activated in one of the service points with a parameter pointing to the first message in a queue of I/O messages. The PHLS will then process the I/O messages one by one until the end of the queue.

The PHLS is activated in one of its service points by the following system call:

```
                    DO : FPHLS

        A0 ---->    PHLS type             % 1 byte
                    PHLS number           % 1 byte
                    Service Point         % 2 bytes
                    Message address       % 4 bytes
```

PHLS type: Indicates the type of the PHLS. 0 means HDLC type, 1 means ASYNCHRONOUS type. (*)

PHLS number: Indicates which Physical Level Server (PHLS) is to be used. A PHLS has the number 0, 1, 2 or 3, reflecting the communication line to which it belongs.

Service Point: Indicates which of the three service points is to be activated:

- Control and Supervise Service Point: 0
- Receive Service Point: 1
- Transmit Service Point: 2 (*)

Message Address: Points to the first message in a queue of I/O messages to be processed by the PHLS at the given service point.

(*) Instead of using these numeric values the user should include the file PIOC-FUNCVAL-Cxx:DEFS into the source file with a $INCLUDE statement, and use the symbolic names. See appendix A.

All I/O buffer areas for all of the PHLS must not exceed 64 Kbytes, and they must be allocated within a 64 Kbytes address boundary.

PIOC memory configuration:

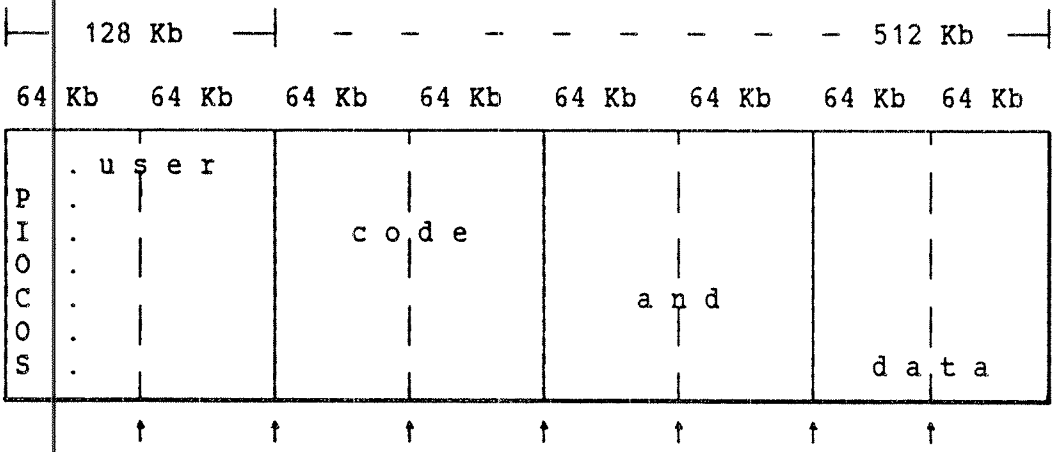
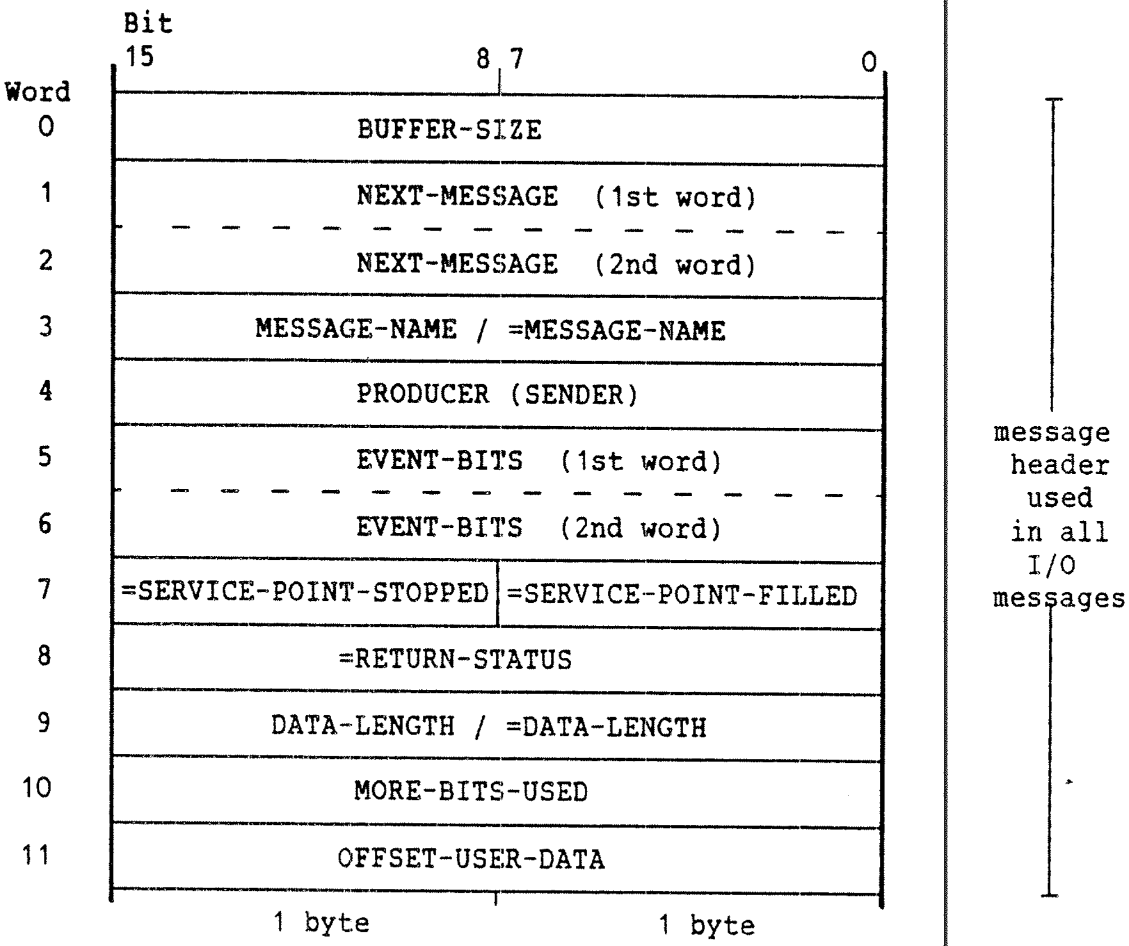


Figure 11. Address boundaries for allocating I/O buffers.

4.1.1 I/O Message Description

The I/O messages for the different message names have different sizes and formats. But all I/O messages must have the same message header:



Word	Bit 15 – 8	Bit 7 – 0
0	BUFFER-SIZE	
1	NEXT-MESSAGE (1st word)	
2	NEXT-MESSAGE (2nd word)	
3	MESSAGE-NAME / =MESSAGE-NAME	
4	PRODUCER (SENDER)	
5	EVENT-BITS (1st word)	
6	EVENT-BITS (2nd word)	
7	=SERVICE-POINT-STOPPED	=SERVICE-POINT-FILLED
8	=RETURN-STATUS	
9	DATA-LENGTH / =DATA-LENGTH	
10	MORE-BITS-USED	
11	OFFSET-USER-DATA	
	1 byte	1 byte

Figure 12. Message header in all I/O messages.

BUFFER-SIZE: Specifies the maximum number of bytes in the message, <u>including the message header</u>.

NEXT-MESSAGE: Contains a pointer to the next message in the queue.
<u>Must be within the 64 Kb I/O message area.</u>

MESSAGE-NAME: Identifies the message type, which is defined as either a request or a response.
When sent to the PHLS, the message name must contain the request name (XXX_REQUEST).
When the message has been processed by the service point, the message name is changed to the corresponding response name (XXX_RESPONSE).

PRODUCER (SENDER): This word must contain the process number of the process which shall get the following event when the message is ready.

EVENT-BITS: The event-bits to be set for the user of the service point when PHLS has completed the I/O operation.

SERVICE-POINT-STOPPED: This parameter will be set to 1 if PHLS has processed the I/O message and there are no more messages in the queue. The service point must now be restarted by a new monitor call.

SERVICE-POINT-FILLED: This parameter will be set to 1 when the message has been processed by PHLS.

RETURN-STATUS: Status of the operation on the I/O message.

DATA-LENGTH: Specifies the number of bytes in the user data field, including the OFFSET-USER-DATA parameter.

MORE-BITS-USED: Number of bits used after the last byte (maximum 7 bits). The MORE-BITS-USED parameter is not to be included in the DATA-LENGTH parameter.

OFFSET-USER-DATA: Number of bytes between the message header and the user data of the message.

The FPHLS call is used to control all the messages to and from PHLS. The type of function depends on which user message you choose. Some basic messages are listed below:

Table 1. PHLS message types.

User message	PIOCOS message	Valid only for service point:
INIT_REQUEST	INIT_RESPONSE	Control and Supervise
CONN_REQUEST	CONN_RESPONSE	Control and Supervise
DIS_REQUEST	DIS_RESPONSE	Control and Supervise
TRAN_REQUEST	TRAN_RESPONSE	Transmit Data
RECV_REQUEST	RECV_RESPONSE	Receive Data

The user specifies XXX_REQUEST messages, while PIOCOS sends XXX_RESPONSE messages when the request has been executed by PHLS.

A logical sequence of messages to the PHLS is

first - request to initialize the line: INIT_REQUEST
second - request to connect: CONN_REQUEST
then - request to send or receive data: TRAN_REQUEST and RECV_REQUEST
at any time - request to disconnect the line: DIS_REQUEST

The next sections explain the three service points and their general purpose in more detail.

4.1.2 Control and Supervise Service Point

This service point will only handle a queue of one element, thus the pointer to the next message should be NIL. This service point is used to initialize the serial line communication controllers, and to establish, supervise and disconnect a connection to a remote partner. Stopping and starting the Transmit and the Receive data Service Points are done here. The mode (HDLC or ASYNCHRONOUS) is defined here.

A successful CONN_REQUEST starts the two other service points of the same PHLS, and a DIS_REQUEST stops them.

4.1.3 Transmit Data Service Point

The part of the message containing the user data (not the message header) will be transmitted on the serial communication line. The service point only handles I/O messages with the message name TRAN_REQUEST. Upon completion of the transfer, the message name will be set to TRAN_RESPONSE by PIOCOS.

The service point will not process any messages before a CONN_REQUEST is processed by the Control and Supervise Service Point of the same PHLS.

The messages sent for this service point have the following format:

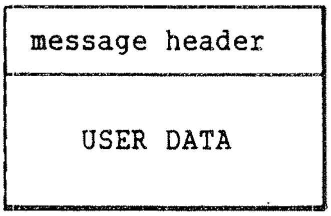


Figure 13. Transmit data message format.

message header: As described on page 49.

USER DATA: The user data part of the message is found here. Note that some bytes appear to be untouched according to the OFFSET-USER-DATA parameter in the message header.

4.1.4 Receive Data Service Point

The user data part of the I/O message will be filled with data from the serial communication line. The service point only handles I/O messages with the message name RECV_REQUEST. When the I/O message is filled, the message name will indicate RECV_RESPONSE. The data length parameter will be set to indicate the size of the data frame.

The service point will not process any messages before a CONN_REQUEST is processed by the Control and Supervise Service Point of the same PHLS.

The messages sent for this service point have the following format:

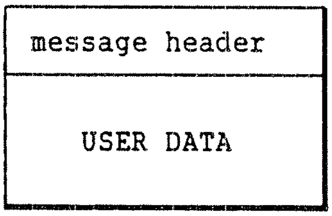


Figure 14. Receive data message format.

message header: As described on page 49.

USER DATA: The user data part of the message is found here. Note that some bytes appear to be untouched according to the OFFSET-USER-DATA parameter in the message header.

4.2 PHLS used for HDLC Communication

The PHLS may be used to transmit and receive data as HDLC frames on the serial communication lines. The frame format meets the ISO IS3309.2 standard. HDLC frames are fully compatible with SDLC (Synchrounous Data Link Control) frames. If you are not familiar with the HDLC concept, we recommend the ND manual 'High Level Data Link Control Interface', ND 12.018.

4.2.1 Control and Supervise Service Point

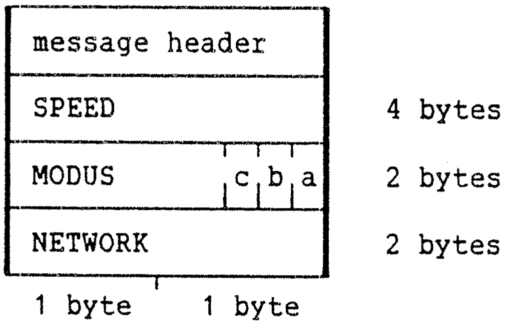
This service point will only handle a queue of one element, thus the pointer to the next message should be NIL. This service point is used to initialize the serial line communication controllers, and to establish, supervise and disconnect a connection to a remote partner. Stopping and starting the Transmit and the Receive Data Service Points are done here.

A successful CONN_REQUEST starts the two other service points of the same PHLS, and a DIS_REQUEST stops them.

4.2.1.1 Initialize (INIT_REQUEST)

The request is used to clear the communication controller and then initialize it to send and receive data in the HDLC frame format.

The message has the format:



message header	
SPEED	4 bytes
MODUS c b a	2 bytes
NETWORK	2 bytes
1 byte 1 byte	

Figure 15. Initialize request HDLC message format.

message header: As described on page 49.

SPEED

Defines the line speed in bits/second. The following baud rates can be selected for HDLC transmissions:

Table 2. Baud rate for HDLC transmissions.

50 baud	4800 baud
110 -"-	9600 -"-
300 -"-	19200 -"-
600 -"-	38400 -"-
1200 -"-	:
2400 -"-	614400 -"-
819200 baud (max rate to another PIOC only)	

MODUS

Sets three different modes of operation:

a) Bit 0 selects normal or test output mode:

0 = normal output, the serial output is connected to the line

1 = test output, the serial output is looped back to the serial input line

b) Bit 1 selects half or full duplex communication,

0 = full duplex,
1 = half duplex.

c) Bit 2 selects the type of IDLE signal to be sent between data transmissions.

0 = the FLAG byte (0111110) is sent to indicate idle transmission,
1 = the line is set HIGH (logical 1's) between data transmissions.

NETWORK

Not presently used.

4.2.1.2 Connect (CONN REQUEST)

The request is used to start the Transmit Data Service Point and the Receive Data Service Point. FLAGS (01111110) will be transmitted on the communication line until the Transmitter Service Point is active.

At this point the message has no parameters, and thus consists only of the message header.

4.2.1.3 Disconnect (DIS REQUEST)

The request will stop all activities on the Transmit Data Service Point and the Receive Data Service Point. Then ABORT (11111111) will be transmitted on the communication line.

At this point the message has no parameters, and thus consists only of the message header.

4.2.2 Transmit Data Service Point

The part of the message containing the user data (not the message header) will be transmitted on the serial communication line. The service point only handles I/O messages with the message name TRAN_REQUEST. Upon completion of the transfer, the message name will be set to TRAN_RESPONSE by PIOCOS.

The Transmit Data Service Point will always send the user data part of a I/O message as one HDLC frame.

The service point will not process any messages before a CONN_REQUEST is processed by the Control and Supervise Service Point of the same PHLS.

The messages sent for this service point have the following format:

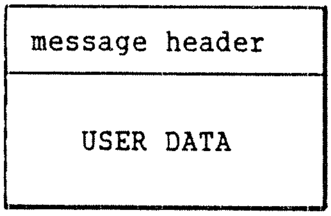


Figure 16. HDLC transmit data message format.

message header: As described on page 49.

USER DATA: The user data part of the message is found here. Note that some bytes appear to be untouched according to the OFFSET-USER-DATA parameter in the message header.

4.2.3 Receive Data Service Point

The user data part of the I/O message will be filled with data from the serial communication line. The service point only handles I/O messages with the message name **RECV_REQUEST**. When the I/O message is filled, the message name will indicate RECV_RESPONSE. The data length parameter will be set to indicate the size of the data frame. Note that the 16 bit CRC will follow the last user data byte, but it will not be included in the data length parameter.

The service point will not process any messages before a CONN_REQUEST is processed by the Control and Supervise Service Point of the same PHLS.

The messages sent for this service point have the following format:

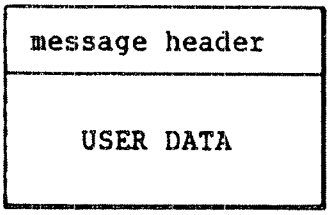


Figure 17. HDLC receive data message format.

message header: As described on page 49.

USER DATA: The user data part of the message is found here. Note that some bytes appear to be untouched according to the OFFSET-USER-DATA parameter in the message header.

4.3 PHLS for Asynchronous Communication

The PHLS may be used to transmit and receive data in the asynchronous mode on the serial communication lines.

4.3.1 Control and Supervise Service Point

This service point will only handle a queue of one element, thus the pointer to the next message is of no interest. This service point is used to initialize the serial line communication controllers, and to establish, supervise and disconnect a connection to a remote partner. Stopping and starting the Transmit and the Receive data Service Points are done here.

A successful CONN_REQUEST starts the two other service points of the same PHLS, and a DIS_REQUEST stops them.

4.3.1.1 Initialize (INIT_REQUEST)

The request is used to clear the communication controller and then initialize it to send and receive data in asynchronous format.
The message has the format:

message header		
SPEED		4 bytes
MODUS		2 bytes
NETWORK		2 bytes
BITS/CHAR		2 bytes
STOP-BITS		2 bytes
XON	XOFF	2 bytes
PARITY		2 bytes
1 byte	1 byte	

Figure 18. ASYNC initialize message format.

message header: As described on page 49.

SPEED

Defines the line speed in bits/second. The following baud rates can be selected for ASYNC transmissions:

Table 3. Baud rate for ASYNC transmissions.

50	baud	2400	baud
110	-"-	4800	-"-
300	-"-	9600	-"-
600	-"-	19200	-"-
1200	-"-	38400	-"-
153600	baud (max rate to another PIOC only)		

MODUS

If set to 1, serial output is looped back to serial input. The baud rate is set according to the speed parameter. The serial output will also be connected to the line.

NETWORK

Not presently used.

BITS/CHAR

Number of bits per character. Applicable values are 6,7 or 8.

STOP-BITS

Number of stop bits. 10 means 1 stop bit, 15 means 1.5 stop bits, while 20 means 2 stop bits.

XON

Defines the XON character. The character that will be sent to request temporary suspension of output. Normally CTRL/S, octal 23, but may be set to any other value.

XOFF

Defines the XOFF character. The character that must be sent to continue transmitting output after an XON has been set. Normally CTRL/Q, octal 21, but may be set to any other value.

If the XON character is set to the same value as the XOFF character, the function is disabled.

PARITY

Defines the type of parity. 0 means no parity, 2 means odd parity, while 3 means even parity.

4.3.1.2 Connect (CONN REQUEST)

This request is used to start the Transmit Data Service Point and the Receive Data Service Point. The communication line will be held high (all 1's) until the Transmitter Service Point is active.

At this point the message has no parameters, and thus consists of the message header only.

4.3.1.3 Disconnect (DIS_REQUEST)

This request will stop all activities at the Transmit Data Service point and the Receive Data Service Point. Then ABORT (11111111) will be transmitted on the communication line.

At this point the message has no parameters, and thus consists of the message header only.

4.3.2 Transmit Data Service Point

The part of the message containing the user data (not the message header) will be transmitted on the serial communication line. The service point only handles I/O messages with the message name TRAN_REQUEST. Upon completion of the transfer, the message name will be set to TRAN_RESPONSE by PIOCOS.

The service point will not process any messages before a CONN_REQUEST is processed by the Control and Supervise Service Point of the same PHLS.

The transfer will stop if the Receive Data Service Point receives an XOFF character as specified in the INITIATE message to the Control and Receive Service Point. The reception of an XON will make the transmitter continue to transfer data.
The messages sent for this service point have the following format:

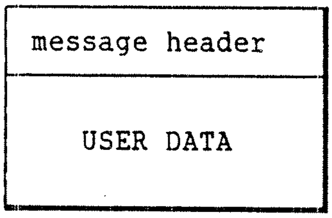


Figure 19. ASYNC transmit data message format.

message header: As described on page 49.

USER DATA: The user data part of the message is found here. Note that some bytes appear to be untouched according to the OFFSET-USER-DATA parameter in the message header.

4.3.3 Receive Data Service Point

The user data part of the I/O message will be filled with data from the serial communication line. The service point only handles I/O messages with the message name RECV_REQUEST. When the I/O message is filled, the message name will indicate RECV_RESPONSE. The data length parameter will be set to indicate the size of the data.

All received XON and XOFF characters will be masked out and not included in the data part of the message.

The service point will not process any messages before a CONN_REQUEST is processed by the Control and Supervise Service Point of the same PHLS.
The messages sent for this service point have the following format:

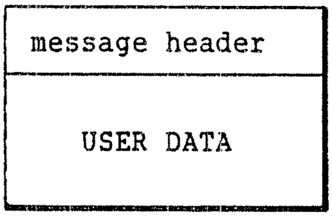


Figure 20. ASYNC receive data message format.

message header: As described on page 49.

USER DATA: The user data part of the message is found here. Note that some bytes appear to be untouched according to the OFFSET-USER-DATA parameter in the message header.

5 ND-100 - PIOC INTERCONNECTION

The PIOC memory is physically located on the PIOC interface board. Two versions of PIOC are available: the ND-865 with 128 Kbytes memory, and the ND-867 with 512 Kbytes memory. This memory is accessible from both the ND-100 and the PIOC processors, and it is defined as a part of the total memory configuration.

The PIOC programs are placed on one or more segments, each occupying up to 128 Kbytes on the segment file(s). When the PIOC program is activated, the segment(s) are copied into SINTRAN III's swapping area, either by the PIOC-MONITOR or a user written ND-100 RT-program.

The PIOC segment(s) must be FIXED into the memory before being able to start the PIOC program.

From PIOC's point of view, the memory will have logical and physical addresses from 0 to 128Kb or 512Kb respectively. The lower part of this memory contains the PIOC operating system PIOCOS, 17 to 29 Kbytes depending on the configurations, while the upper part contains user processes (here called PIOC-USER part).

An ND-100 RT-program in SINTRAN III may share data with the programs (processes) in the PIOC-USER part of PIOC. This is done by placing the PIOC-USER segment in the alternative page table. By use of ALTON/ALTOFF, the RT-program may read or write data in the PIOC-USER part of memory, while a PIOC process is not allowed to access the ND-100 memory directly.

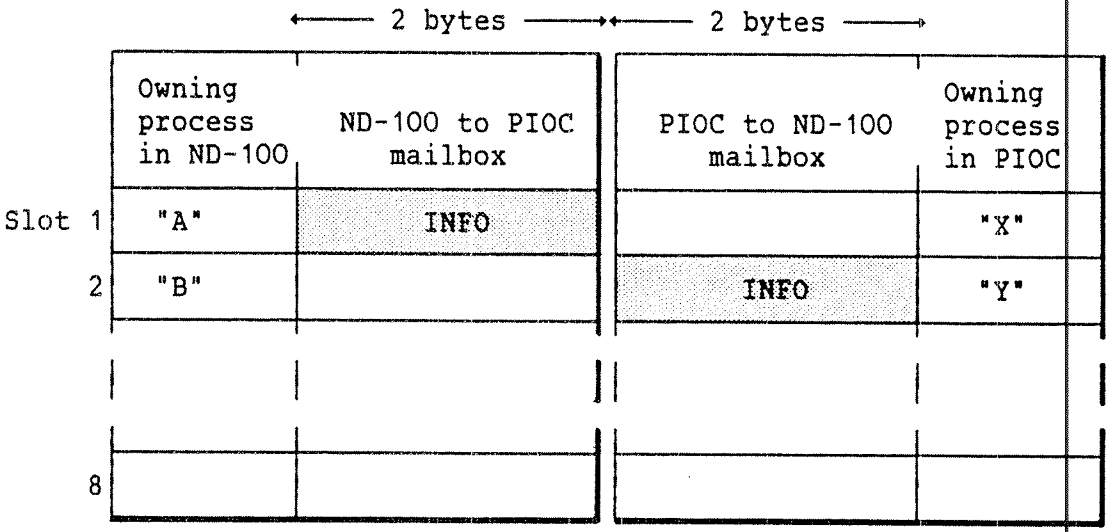
Note that it is highly inadvisable to have common data longer than 2 bytes in PIOC memory which is written to and read by both processors (MC68000 and ND-100) simultaneously. The PIOC's memory is not protected during accesses, thus one processor may read data while the other is modifying it.

5.1 How to Synchronize ND-100 and PIOC Processes

Processes in PIOC and ND-100 may communicate through kicks, which are a result of a call to the operating system (either SINTRAN III or PIOCOS). A kick from a PIOC process to a ND-100 process makes the ND-100 process active (brings it out of 'RT-WAIT'). A kick from a ND-100 process to a PIOC process results in an EVENT being created for the process in PIOC. (See page 20.)

5.2 Looking Closer into the Kick-mechanism

The processes needing to communicate may make use of a kick channel. This is a mail system which allows up to eight pairs of processes to establish a temporarily fixed communication link with each other. The kick channel has eight slots, one for each pair of processes. Each slot may be reserved by two communicating processes (one in ND-100 and one in PIOC). The processes send messages to each other via their common slot in the kick channel. When a message is sent to the kick channel by one process, a kick is generated for the other process. The kick is a signal to the other process meaning that information is waiting for it. The information itself is a 2 byte data word which is stored in a mailbox. Each slot has two mailboxes, one for information sent from the ND-100 process, and one for information sent from the PIOC process. When a process has been 'kicked', the information may be fetched from the appropriate mailbox in the slot.



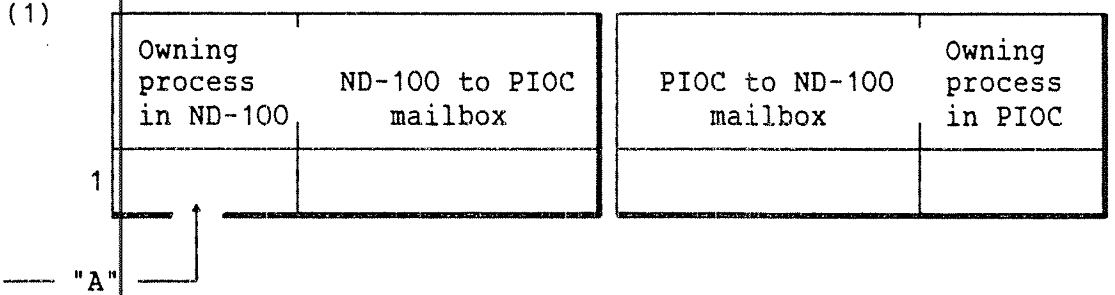
	← 2 bytes →		← 2 bytes →	
	Owning process in ND-100	ND-100 to PIOC mailbox	PIOC to ND-100 mailbox	Owning process in PIOC
Slot 1	"A"	INFO		"X"
2	"B"		INFO	"Y"
8				

Figure 21. Kick channel in PIOC memory.

Data in the form of 2 bytes, 16 bit, may be passed between the two communicating processes (one PIOC process and one ND-100 process). But previously they must have reserved the same slot (figures (1)&(2) below). The KICK function places the information in the mailbox belonging to the slot in the kick channel. It is then retrieved by the FETCH function.

As an ND-100 process sends these 2 bytes of information to the mailbox of its reserved slot (figure 3 below), a kick is passed to the PIOC process which has reserved the other side of the slot (figure (4) below). It may then fetch the information (figure (5) below). The illustrations below shows the logical sequence of commands/functions that two processes have to perform to allow one process to send information to the other. The numbers to the left of each figure refer to the corresponding numbers in this paragraph. The functions specified in parentheses are discussed later in this chapter.

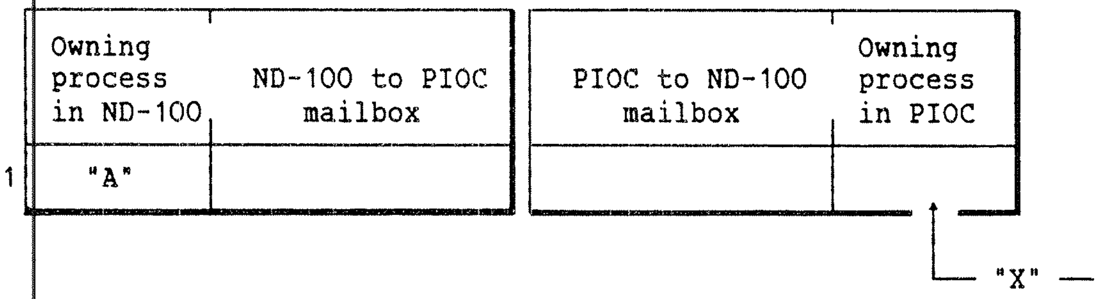
(1)



	Owning process in ND-100	ND-100 to PIOC mailbox	PIOC to ND-100 mailbox	Owning process in PIOC
1				

The ND-100 process "A" reserves the ND-100 side of slot number 1 (function: RES_SLOT).

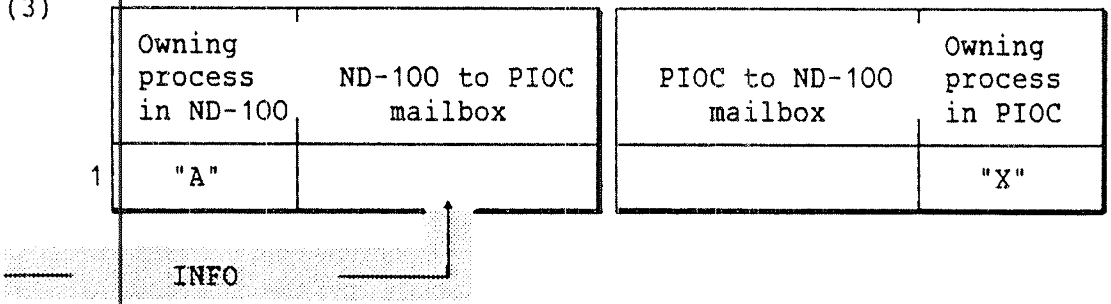
(2)



	Owning process in ND-100	ND-100 to PIOC mailbox	PIOC to ND-100 mailbox	Owning process in PIOC
1	"A"			

The PIOC process "X" reserves the PIOC side of slot number 1 (call: FRES_SLOT).

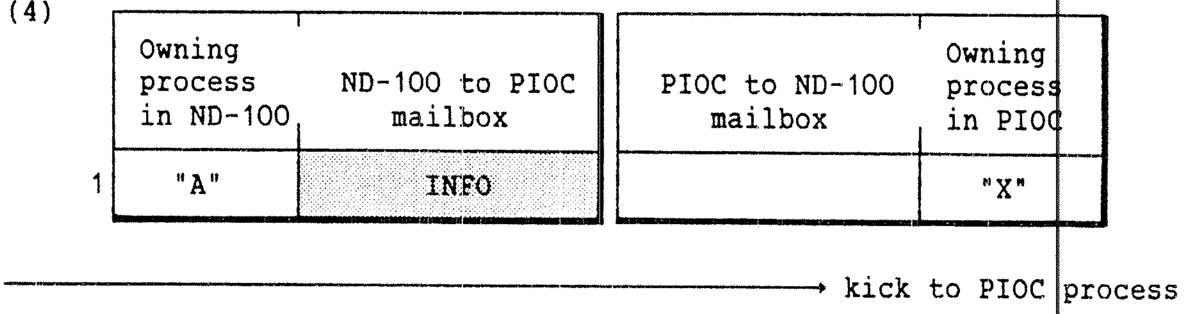
(3)



	Owning process in ND-100	ND-100 to PIOC mailbox	PIOC to ND-100 mailbox	Owning process in PIOC
1	"A"			"X"

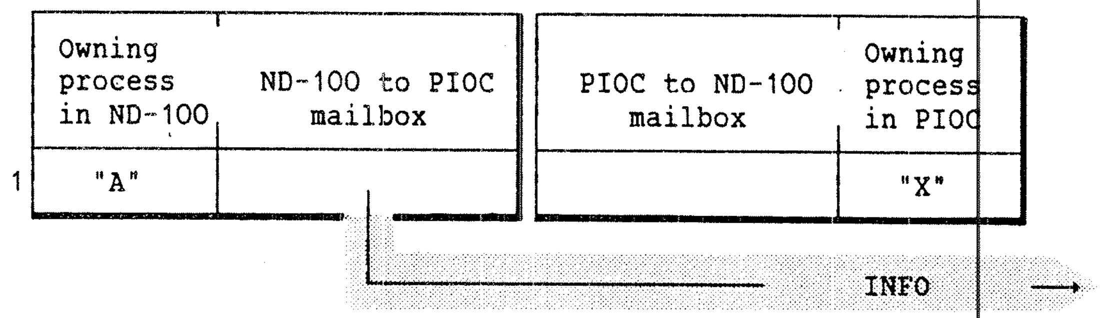
The ND-100 process sends INFO to its reserved slot (function: KICK),

(4)



and a kick is sent to the PIOC process (which has reserved the PIOC side of the same slot). (Implicit when the KICK function is used.)

(5)



The PIOC process "X" may then fetch the INFO (call: FFETCH).

The same thing happens when a PIOC process sends information to the mailbox of its reserved slot: The corresponding ND-100 process gets 'kicked' and it may fetch the information at once or later.

5.3 What Type of Information is Transferred in the Kick Channel?

The kick channel mailboxes are 2 bytes (16 bit word) registers. The amount of information a process can send in 2 bytes is of course very limited. Often a process may need to send more data. The kick channel may then be used to send the address to a larger data area where the actual data is found. In this way the mailbox does not contain any valid data, but a pointer only. The data itself may also be sent as single words to the mailbox, one at a time, but this is not advisable since the receiving process must be 'kicked' for each word.

5.4 MON PIOC (MON 255) - The PIOC Monitor Call in SINTRAN III

PIOC (MON 255) is a general monitor call used in ND-100 programs to access PIOC functions and processes. The monitor call performs the function given by the ND-100 T-register.

Common for all functions is that the ND-100 X-register must contain the PIOC logical device number (LDN), and the T-register contains an error code, if any, on return.

The library PIOC-N100LIB-C:BRF supplied with PIOC Basic Software contains library routines for the monitor call PIOC. These routines, partly written in assembler, may be called from PLANC programs, or any other high-level programming language. The PLANC Reference Manual (ND-60.117) contains information on how to call PLANC routines from other programming languages.

Note that the monitor call MON PIOC can only be called by user SYSTEM, from an RT-program or from ring 2. Otherwise the error code -10B is returned.

Explanation of parameter values for the PIOC (MON 255) call
Here you find an explanation of parameter values, used by one or several of the monitor call functions.

A) Parameters supplied by the user:

LDN the logical device number specifies which PIOC module to access. This number is defined within the SINTRAN III operating system according to the following table:

Table 4. Logical Device Number (LDN).

PIOC module number:	LDN log.dev.no:
0	1700
1	1701
2	1702
3	1703
4	1704
5	1705
6	1706
7	1707
8	1710
9	1711
10	1712
11	1713

The PIOC module number is selected with a thumbwheel switch (12J) on the PIOC module itself. For details, please see the PIOC Reference Manual, ND-02.003, chapter 2.

SLOT The PIOC kick channel contains eight slots. The SLOT parameter must be a number from 1 to 8, depending on which slot you want to access.

FUNCTION This parameter contains the number of the desired function. It is stored as an integer in the range 0 through 7 in the T-register. The available functions and their corresponding T-register values are listed below.

Table 5. MON PIOC functions

Function	T-register	Short explanation
RES_SLOT	0	Reserves the ND-100 side of a slot.
REL_SLOT	1	Releases the ND-100 side of a slot.
KICK	2	Sends a kick to a PIOC-process.
FETCH	3	Reads a message from a PIOC process.
SEGLOAD	4	Loads segment into PIOC memory.
UNLOAD_PIOC	5	Unloads all segments from PIOC mem.
START_PIOC	6	Starts the PIOC.
STOP_PIOC	7	Stops the PIOC.

B) Parameter supplied by SINTRAN III:

RETVAL receives an error code, if any, from SINTRAN III. If the function performes a successful execution, RETVAL is 1. Possible RETVAL values are:

Table 6. MON PIOC return values (octal).

1:	Successful execution.
2:	No answer from PIOC ND-100 driver.
-10:	No privilege.
-11:	Function not allowed before the PIOC is started.
-24:	Illegal function code (outside range 0 to 7).
-25:	The slot is occupied by another process.
-26:	Illegal slot number (outside range 1 to 8).
-27:	The slot is not reserved by you. You must reserve first it.
-30:	The mailbox is not empty, and can not receive info.
-31:	The mailbox <u>is</u> empty. No message to fetch.
-32:	Illegal LDN.
-33:	The PIOC is not initiated.
-34:	The PIOC memory is not all fixed.
41:	Space not available.
42:	Illegal segment.
43:	Segment not loaded.
44:	Attempt to fix demand segment.
45:	Attempt to fix too many pages.
46:	Segment already fixed at different address.

C) Other parameters:

INFO This is the 2 bytes of information, the message itself.

Other There are a few other parameters used for some functions. They are explained in the appropriate function description.

In the following pages, each monitor call function is discussed in greater detail.

RES_SLOT - reserve the ND-100 side of a slot

<u>NPL:</u>
```
X:=LDN; A:=SLOT; T:=0; *MON PIOC
T=:RETVAL
```

<u>PLANC:</u> You may use the following PIOC-N100LIB routine:
```
ROUTINE VOID,INTEGER (INTEGER,INTEGER): RES_SLOT &
                     (LDN,    SLOT   )
```

Example 3. Reserve the ND-100 side of a slot.

The ND-100 side of the SLOT will be reserved by the calling process in ND-100.

REL_SLOT - release the ND-100 side of a slot

<u>NPL:</u>
```
X:=LDN; A:=SLOT; T:=1; *MON PIOC
T=:RETVAL
```

<u>PLANC:</u> You may use the following PIOC-N100LIB routine:
```
ROUTINE VOID,INTEGER (INTEGER,INTEGER): REL_SLOT &
                     (LDN,    SLOT   )
```

Example 4. Release the ND-100 side of a slot.

The ND-100 side of the SLOT will be released if it is reserved by the calling ND-100 process. All kicks sent for this slot (when released) will be neglected.

KICK - send information to a PIOC process

<u>NPL:</u>
```
X:=LDN; A:=INFO=:D; A=:SLOT; T:=2; *MON PIOC
T=:RETVAL
```

<u>PLANC:</u> You may use the following PIOC-N100LIB routine:
```
ROUTINE VOID,INTEGER (INTEGER,INTEGER,INTEGER):KICK &
                     (LDN,    SLOT,   INFO   )
```

Example 5. Send information to a PIOC process.

The effect of this function depends on the contents of the INFO parameter. If the INFO parameter is 0 (zero), a kick (resulting in an event for the PIOC process) is generated. If the INFO parameter is other than 0, the effect of the function depends on the contents of the ND-100 to PIOC mailbox. If the mailbox is empty, the INFO parameter will be put in the mailbox and a kick will be generated for the PIOC process. If, however, the mailbox is full, the call will have no effect.

FETCH - get information from a PIOC process

```
NPL:    X:=LDN; A:=SLOT; T:=3; *MON PIOC
        T=:RETVAL; D=:A=:INFO

PLANC: You may use the following PIOC-N100LIB routine:
ROUTINE VOID,INTEGER (INTEGER,INTEGER,INTEGER WRITE):FETCH &
                     (LDN,    SLOT,   INFO         )
```

Example 6. Get information from a PIOC process.

The ND-100 process uses this function to read the contents of the PIOC to the ND-100 mailbox in the specified slot, whether the mailbox has any contents or not.

SEGLOAD - load a segment into PIOC memory and fix it

```
NPL:    X:=LDN; A:=PAGE=:D; A:=SEGNO; T:=4; *MON PIOC
        T=:RETVAL

PLANC: You may use the following PIOC-N100LIB routine:
ROUTINE VOID,INTEGER (INTEGER,INTEGER,INTEGER):SEGLOAD &
                     (LDN,    SEGNO,  PAGE   )
```

Example 7. Load and fix a segment into PIOC memory.

This function may be used to load a ND-100 segment generated by the RT-LOADER into the PIOC memory.

LDN is the PIOC number to which the loading relates to.

SEGNO is the actual segment number to be loaded. (Corresponds to the SINTRAN III segment number defined by the RT-LOADER, as explained in chapter 7.)

PAGE is the actual page number the loading is to begin at within the appropriate PIOC.

For PIOC/128Kb this number must be within the range 0 - 77 (octal) inclusive.

For PIOC/512Kb this number must be within the range 0 - 377 (octal) inclusive.

The illustration shows the situation if the PIOC memory is located at ND-100 physical memory from page 200, and you have loaded segment no 121 into memory from PIOC page no 26:

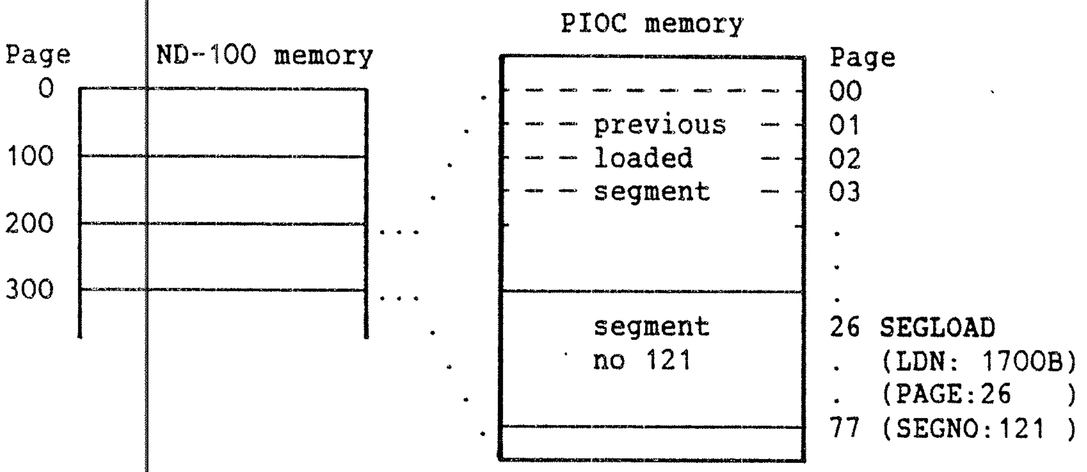


Figure 22. Relationship between ND-100 and PIOC memory.

Configuring the PIOC memory as a part of the ND-100 memory is controlled by thumbwheel switches 7J and 9J on the PIOC module. The switches may be set to define which memory pages both the PIOC and the ND-100 should address. Beware that when the PIOC is not active this memory is used by the SINTRAN III operating system as swapping area. SEGLOAD will, when called, reserve the memory area by issuing a FIXC (fix contiguous) monitor call. Further details on configuring the PIOC/ND-100 memory can be found in the PIOC Reference Manual, ND-02.003, chapter 2.

UNLOAD_PIOC - remove all fixed segments from PIOC memory

NPL:
```
X:=LDN; T:=5; *MON PIOC
T=:RETVAL
```

PLANC: You may use the following PIOC-N100LIB routine:
```
ROUTINE VOID,INTEGER (INTEGER): UNLOAD_PIOC &
                      (LDN    )
```

Example 8. Remove all fixed segments from PIOC memory.

This function unloads all fixed segments from the PIOC memory. It performs the opposite function of the previously executed SEGLOAD's.

START_PIOC - start the PIOC

<u>NPL:</u>

```
X:=LDN; A:=ST_ADDR; T:=6; *MON PIOC
A=:FAULT_PAGE; T=:RETVAL
```

<u>PLANC:</u> You may use the following PIOC-N100LIB routine:

```
ROUTINE VOID,INTEGER(INTEGER,INTEGER4,INTEGER WRITE):START_PIOC &
                    (LDN,    ST_ADDR, FAULT_PAGE   )
```

Example 9. Start PIOC.

Before PIOC can be started, the <u>whole</u> PIOC memory of 128/512 Kbytes must be fixed, either as one segment, or as many smaller segments in a contiguous area. The function SEGLOAD loads a segment generated by the RT-LOADER into the PIOC memory and fixes it. In addition, the rest of the PIOC memory must be fixed. The START_PIOC function starts the PIOC with the given LDN.

When the ST_ADDR is 0 (zero), PIOCOS is initiated and started, then a jump to the start address defined as 'AUTO_START' is performed. The process will run at priority level 1. The 'AUTO_START' symbol must be defined as a global entry when you link the :NRF-files using the Linkage Loader.

If the ST_ADDR is not zero, the processor starts execution at the specified address but no initialization or jump to 'AUTO_START' is performed. It is not recommended to start PIOCOS this way.

If the PIOC memory is not completely fixed, FAULT_PAGE receives the number of the first page that is not fixed.

Please note that PIOC process(es) may NOT do any input from or output to your terminal if PIOC is started with this monitor call function. This is only achieved if you start it from the PIOC monitor, and only as long as the PIOC-MONITOR is running on the PIOC.

STOP_PIOC - stop the PIOC

<u>NPL:</u>

```
X:=LDN; T:=7; *MON PIOC
T=:RETVAL
```

<u>PLANC:</u> You may use the following PIOC-N100LIB routine:

```
ROUTINE VOID,INTEGER (INTEGER): STOP_PIOC &
                     (LDN    )
```

Example 10. Stop PIOC.

The PIOC with the logical device number LDN will stop running.

5.5 The ND-100 Calls in PIOC

The ND-100 calls are used by PIOC processes to communicate with RT-programs in SINTRAN III. The following calls are available:

FRES_SLOT - reserve the PIOC side of a slot

```
                D0 : FRES_SLOT

        A0 ----> SLOT              % 2 bytes
                 EVENT BITS        % 4 bytes
```

The PIOC side of the given SLOT will be reserved by the calling process. The specified EVENT BITS are sent to the PIOC process if a kick is sent to it from an ND-100 process.

On return, D0 contains the status of the operation.

FREL_SLOT - release the PIOC side of a SLOT

```
                D0 : FREL_SLOT

        A0 ----> SLOT              % 2 bytes
```

The PIOC side of the given SLOT will be released from the calling PIOC process. Kicks sent for this slot will have no effect on any PIOC process.

On return, D0 contains the status of the operation.

FKICK - send a kick to a ND-100 process

```
                D0 : FKICK

        A0 ----> SLOT              % 2 bytes
                 INFO              % 2 bytes
```

The effect of this function depends on the contents of the INFO parameter. If the INFO parameter is 0, a KICK (resulting in a 'RT' for the ND-100 process which may be in the 'RT-WAIT' state) is generated.

If the INFO parameter is other than 0, the effect of the function depends on the status of the PIOC to ND-100 mailbox. If the mailbox is empty, the INFO parameter is put in the mailbox, and a kick is generated for the ND-100 process. If however, the mailbox is full, an error indication is set in the D0 register on return from PIOCOS, but the call will have no effect: NO kick is sent to the corresponding process in the ND-100.

FFETCH - get info from a ND-100 process

```
                    DO : FFETCH

          AO ----> SLOT            % 2 bytes
                   =INFO           % 2 bytes
```

The PIOC process uses this function to read the contents of the ND-100 to PIOC mailbox in the specified slot, whether the mailbox has contents or not. After the reading, the contents of the mailbox are set to zero.

On return, DO contains the status of the operation.

5.6 The ND-100 XMSG System from PIOC

The "remote" XMSG system running in the ND-100 may be used by processes in PIOCOS through the following system call. In this way processes in PIOC may communicate with tasks (processes and drivers) in the ND-100 or other PIOC's.

Processes running within the same PIOC, should use the "local" XMSG calls as described on page 32, as this is considerably faster.

FNXMSG - PIOC communication with the ND-100 XMSG SYSTEM.

```
                    DO : FNXMSG

          AO ----> T-reg           % 2 bytes
                   A-reg           % 2 bytes
                   D-reg           % 2 bytes
                   X-reg           % 2 bytes
                   PADDR           % 4 bytes
```

T A D X -registers: These parameters should be set according to the specification for the corresponding registers as described in the SINTRAN III Communication Guide (ND-60.134) and the COSMOS PROGRAMMING GUIDE (ND-60.164).

PADDR: This parameter is only used in the functions READ and WRITE, and gives the buffer address in the PIOC. In this case the A-reg parameter is not applicable.

If the XFWTF flag is set the function parameter (see the Communication Guide ND-60.134). The process will be suspended waiting for the call to be completed. If the XFWAK flag is set, the process will continue whether the function is completed or not. Upon completion an event (BIT 31) will be generated for the appropriate process.

When the PIOC is unloaded (either by the PIOC monitor call or by the PIOC-MONITOR), all processes in the PIOC will be disconnected from XMSG.

A set of subroutines to carry out the communication with tasks running in the ND-100 are available in PIOCOS object code and can be found in the XMSG library further described in the COSMOS PROGRAMMING GUIDE (ND-60.164).

5.7 Global Variables in PIOCOS

There are some variables in PIOCOS which can be imported to application processes.

- REALTIME an INTEGER4 variable containing the time in 5msec-units since the PIOCOS is started
- N100_CPU an INTEGER constant containing the ND100-CPU number in which the PIOC reside
- PIOC_NUMBER an INTEGER constant containing the PIOC number in this ND100-system (0..11D)

Note that these variables reside in the PIOC's write-protected data area.

6 THE PIOC-MONITOR

The PIOC-MONITOR is a program that runs on the ND-100, for loading; supervising; and controlling the PIOC. The PIOC-MONITOR can only control one PIOC at a time.

You must be logged in as user SYSTEM in order to run the PIOC-MONITOR, since it is protected from unauthorized use.

When starting the PIOC-MONITOR, it asks the user to specify which PIOC to supervise. The PIOC numbers not reserved by other users are shown in parentheses. You may, for example, answer number 0 (zero):

```
@PIOC-MONITOR-C

PIOC-Monitor - Release : Cxx - <month> <day>, <year>

Give PIOC-number:   0  1 or EXIT : 0
PIOC started
PIOCOS - Release MARCH 22, 1985
The selected PIOCs address-range is 0B to    377777B

P-M:
```

Figure 23. Starting the PIOC-MONITOR-C.

Having come so far, the monitor prompts with P-M: . You are now free to use any of the PIOC-MONITOR commands. A full list of commands can be obtained by the HELP command, see page 81.

In the following sections, the commands are described in their logical order. Some aspects are common to all commands except the "@"-SINTRAN command and the subcommands in the LOOK-AT commands:

- All commands have a special environment: Some can always be executed, for some the selected PIOC must be loaded, for some the PIOC must be started.

  The HELP command shows the environment for the various commands.

- Some commands have an unlimited number of parameters. These are enclosed within parantheses in the HELP-list shown on page 81.

  Such parameters are not prompted and can only be given on the command-line or in connection with a previous parameter prompting.

- All numerical parameters are range-checked and have a default radix (octal or decimal) which is standard in other ND products.

However, you may change the radix by appending a "D" or "B" to the number, e.g. 10D or 12B.

And, if you type a number with the digits "8" or "9" where octal is default, the command-processor will change to decimal and write "Using DECIMAL.".

- All address parameters (except in the LOAD command) can be symbolic if you first execute the LOAD-ENTRY-LIST command.

But note: successful execution depends on the release of the LINKAGE-LOADER you used to load your PIOCOS. This PIOC basic software works with the G-release.

If PIOCOS already is started, a register name can also be given, except the SR-register. The contents of the register is used as the address.

If a symbol is given the same name as a register, the register name is used.

6.1 The EXIT and HELP Commands

EXIT - to leave the PIOC-MONITOR

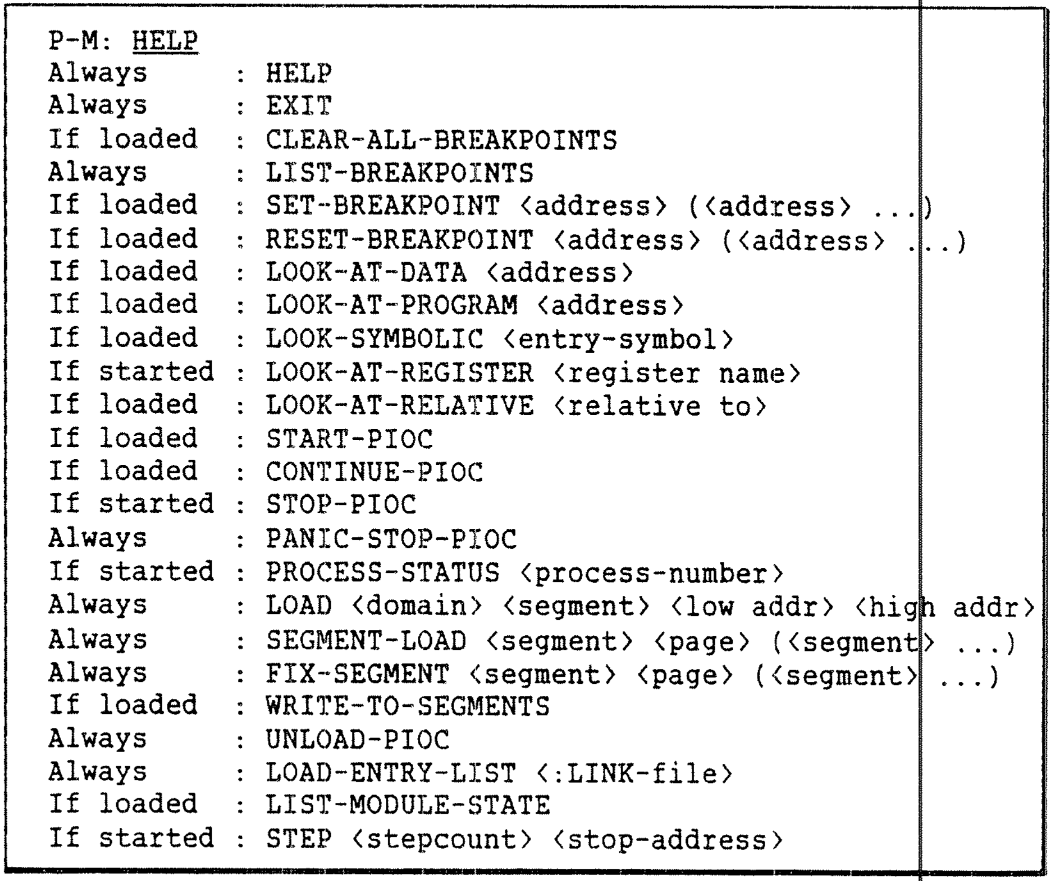
P-M: EXIT - (no parameters)

The EXIT command removes all breakpoints from the PIOC, releases the terminal-IO-slot and the PIOC as a device, writes an exit message to the terminal and returns to SINTRAN.

HELP - lists all the available commands with parameters

P-M: HELP (no parameters)

The HELP command writes out all available commands with the corresponding environment.



```
P-M: HELP
Always     : HELP
Always     : EXIT
If loaded  : CLEAR-ALL-BREAKPOINTS
Always     : LIST-BREAKPOINTS
If loaded  : SET-BREAKPOINT <address> (<address> ...)
If loaded  : RESET-BREAKPOINT <address> (<address> ...)
If loaded  : LOOK-AT-DATA <address>
If loaded  : LOOK-AT-PROGRAM <address>
If loaded  : LOOK-SYMBOLIC <entry-symbol>
If started : LOOK-AT-REGISTER <register name>
If loaded  : LOOK-AT-RELATIVE <relative to>
If loaded  : START-PIOC
If loaded  : CONTINUE-PIOC
If started : STOP-PIOC
Always     : PANIC-STOP-PIOC
If started : PROCESS-STATUS <process-number>
Always     : LOAD <domain> <segment> <low addr> <high addr>
Always     : SEGMENT-LOAD <segment> <page> (<segment> ...)
Always     : FIX-SEGMENT <segment> <page> (<segment> ...)
If loaded  : WRITE-TO-SEGMENTS
Always     : UNLOAD-PIOC
Always     : LOAD-ENTRY-LIST <:LINK-file>
If loaded  : LIST-MODULE-STATE
If started : STEP <stepcount> <stop-address>
```

Figure 24. List of commands in PIOC-MONITOR-C.

6.2 The LIST-MODULE-STATE and LOAD-ENTRY-LIST Commands

LIST-MODULE-STATE - list status information for modules

P-M: LIST-MODULE-STATE (no parameters)

This command lists the compilation date of all PIOCOS-modules except PIOC-TRAPV-C:NRF. This is only another tool for keeping track of different revisions of the PIOC Basic Software.

LOAD-ENTRY-LIST - load symbolic names

P-M: LOAD-ENTRY-LIST <file-name>

This command loads the :LINK file, which must belong to the system in the current PIOC.

```
P-M: LOAD-ENTR (PIOC-TEST)PIOC-0

Unsatisfied reference :          NONE_3
Unsatisfied reference :          NONE_2
220 symbols
```

Figure 25. Loading symbolic entry names.

After this command you may use symbolic names in most of the commands. The names which are allowed, are those you have exported from your PLANC modules.

For special debugging it is also possible to keep the PIOCOS entries available. Note that there is no command for listing the entries. You must make a printout from the LINKAGE-LOADER when building your system.

Note that the internal table for all the entries has a limited size. Therefore, an overflow may occur, and a message will be written to the terminal. However, all symbols which were read into the table before the overflow, will remain available.

If you use this command several times, you can append new entries to the table. Names already in the table will be ignored.

The same internal table is used for both the LOAD-ENTRY-LIST and the LOAD commands. All entries are therefore cleared after executing the LOAD command. (See section 6.3.)

6.3 Panel Commands

These commands are used for operational control of the PIOC processor. Panel commands are commands covering functions normally found on the operator panel of many computers.

LOAD - loads a linkage-loaded segment into PIOC memory

```
P-M: LOAD <domain-name>,<segment-name>,<low-addr>,<high-addr>
```

This command may be used to load a :PSEG segment generated by the Linkage Loader into the PIOC memory. This can be a single PIOC user process or several processes occupying the total PIOC memory.

The <domain-name> and the <segment-name> are the name of the domain file and the name of the segment as defined for the process by the commands COPY-DOMAIN and APPEND-SEGMENT in the Linkage Loader. (See page 103.)

The <low-addr> is the memory start address in PIOC where the segment shall be stored, while the <high-addr> is the maximum address location the segment may occupy in the PIOC memory. Symbolic addresses are not allowed here. Usually you have to use the default values for the address parameters.

When LOAD is used, it is assumed that a segment of sufficient size is defined by the RT-LOADER, and fixed in memory before issuing the command. The LOAD command simply copies the content of the :PSEG segment onto the physical memory of the PIOC.

To speed up the LOAD function, this command needs a buffer which has to be as large as possible. It therefore uses a buffer common with the LOAD-ENTRY-LIST command (see section 6.2). This means: All symbolic entries are cleared when executing this LOAD command.

This command is also described on page 111.

The FIX-SEGMENT command (described below) also loads and fixes RT-LOADER segments, but you can also use the @FIXC SINTRAN command before starting the PIOC-MONITOR and the LOAD command. @FIXC requires a segment number and a physical page number for the memory area shared between the PIOC and ND-100. The physical page number can be derived from the tables on page 141.

SEGMENT-LOAD - load segments made by the RT-LOADER

P-M: SEGMENT-LOAD <segment> <page> ((<segment> ...)

This command is very similar to LOAD, but it is used for segments generated by the RT-LOADER, not the Linkage-Loader. The SEGMENT-LOAD command brings the specified <segment> into the PIOC memory, starting at <page> which must be an octal number in the range 0 - 77/377. It also fixes the segment.

After the page number, you may specify more segments. They will be loaded following the previous segment.

The command performs the same function as the SINTRAN III monitor call PIOC (MON 255) function SEGLOAD. The command accepts also a segment name instead of a segment number.

This command is also described on page 113.

FIX-SEGMENT - load and fix segments

P-M: FIX-SEGMENT <segment> <page> ((<segment> ...)

Similar to the SEGMENT-LOAD command, a segment generated by the RT-LOADER is loaded and fixed in the memory. Several segments can be specified, and they will be loaded following the first. Page numbers can only be given for the first segment.

WRITE-TO-SEGMENT - write segments to the segment file

P-M: WRITE-TO-SEGMENT (no parameters)

Writes back all fixed segment(s) in the PIOC to the SINTRAN III's segment file(s). If the segment were originally loaded by the LOAD command, it may from now on be loaded with the SEGMENT-LOAD command. This command loads a segment faster, and does not require previous fixing of the segment(s). When defining the segment(s) by the RT-LOADER's NEW-SEGMENT or NEW-BACKGROUND-SEGMENT command, the parameter <WP/NP> must be set to WP.

START-PIOC - starting the PIOC

P-M: START-PIOC (no parameters)

The PIOC processor will start execution at the entry 'AUTO_START'.

The symbol 'AUTO_START' must be defined as a global label when linking the :NRF-files with the Linkage Loader.

The START-PIOC command brings the PIOC MONITOR into a transparent-mode, and before other monitor commands can be used (e.g. STOP-PIOC) it must be reset from this state with:;

CTRL + L

Note that typing CTRL+L does not execute a STOP-PIOC command, but gives the PIOC-MONITOR control over the terminal input. To really stop the process running in the PIOC, the STOP-PIOC command must be given explicitly.

STOP-PIOC - stopping the PIOC

P-M: STOP-PIOC (no parameters)

The PIOC processor is stopped. Registers and memory may be inspected and altered with the LOOK-AT commands. The CONTINUE-PIOC command starts the processor again.

PANIC-STOP-PIOC - panic stop of the PIOC

P-M: PANIC-STOP-PIOC (no parameters)

If the PIOC processor for any reason is looping on hardware priority level, i.e. 6 or 7, the normal STOP command has no effect. However, the PANIC command always stops the processor with a RESET trap. After a PANIC-STOP-PIOC command, memory and registers except for the program counter and system stack pointer may be examined.

Since the program counter and the system stack pointer are NOT saved when PANIC-STOP-PIOC is performed, it is NOT possible to resume execution with the CONTINUE-PIOC command.

CONTINUE-PIOC - continue after STOP-PIOC or a breakpoint

P-M: CONTINUE-PIOC (no parameters)

The PIOC processor will resume execution at the current program counter if it has been stopped with STOP-PIOC, but not with PANIC-STOP-PIOC.

The CONTINUE-PIOC command also brings the PIOC-MONITOR into transparent-mode, and before other monitor commands can be used (e.g. STOP-PIOC) it must be reset from this state with:;

CTRL + L

Note that typing CTRL+L does not execute a STOP-PIOC command, but gives the PIOC-MONITOR control over the terminal input. To really stop the process running in the PIOC, the STOP-PIOC command must be given explicitly.

UNLOAD-PIOC - unload the PIOC segments

P-M: UNLOAD-PIOC (no parameters)

This command unloads all PIOC segments previously loaded with the SEGMENT-LOAD or the FIX-SEGMENT commands.

6.4 Debugging Commands

SET-BREAKPOINT - setting a breakpoint

P-M: SET-BREAKPOINT <address> (<address> ...)

This command sets breakpoints at the specified addresses. When a breakpoint is reached, execution terminates and control is passed to the PIOC-MONITOR. A breakpoint will be reset (removed) when it is reached.

Execution may be resumed by using the STEP or CONTINUE-PIOC commands.

```
P-M: SET-BREAKPOINT PIOCTRAP,30000

Breakpoint 1 installed at 43012
Breakpoint 2 installed at 30000
```

Figure 26. Setting breakpoints.

RESET-BREAKPOINT - resetting a breakpoint

P-M: RESET-BREAKPOINT <address> (<address> ...)

The breakpoints in the specified addresses are removed.

CLEAR-ALL-BREAKPOINTS - resetting all breakpoints

P-M: CLEAR-ALL-BREAKPOINTS (no parameters)

All breakpoints set with the SET-BREAKPOINT command will be removed.

LIST-BREAKPOINTS - list all breakpoints

P-M: LIST-BREAKPOINTS (no parameters)

All breakpoints set in the PIOC will be listed.

STEP - execute some instructions

P-M: STEP <number of steps>,<stop-address>

The instruction pointed to by the program counter is disassembled and shown on the terminal. Then the instruction is executed and this is also written to the terminal. If it was a branch instruction, and the branch is executed, an extra line is written to the terminal. This procedure will be repeated until one of the following alternatives occurs:

- the number of steps given as the first parameter is executed (default is 1 step)
- the address given as the second parameter is reached (default: no address stop)
- an address in the range 0 to 1777B or an address higher than the highest possible memory address is reached (this memory area will never contain code)
- a STOP instruction is reached

After a STEP command a simple <CR> will cause a "STEP,,," to be executed. That means that after you have issued the STEP command once, it can be repeated by typing <CR> only.

6.5 LOOK-AT Commands

These commands make it possible to display and modify registers and locations in the PIOC memory. The PIOC must be loaded, and for the LOOK-AT-REGISTER command, the PIOC must be started also.

The LOOK-AT commands have a set of subcommands as follows:

CR Carriage Return causes the next item, register, instruction, or location to be displayed.
CODE Inserts MC68000 instructions.
EXIT Terminates the LOOK-AT command.
PERMIT-DEPOSIT Must be used before changes are allowed.
EXTRA-FORMAT Defines format of the displayed data.

Special notations used with slash (/, indirect) subcommands:

m = address (octal) or register name.
n = number of bytes.
CR = Carriage Return.

m/CR Take the value of *m* as the address and display this location. *m* may also be a register name.

/CR Take the contents of the current location as the next address and display this location. If the current location is a register, the monitor will start displaying the memory at the address which is contained in the register.

m,n/CR Take the value of *m* as the next address and display *n* bytes. Also *m* may be a register name.

xxx CR Memory or registers are modified by typing the new value in the current default format followed by a *CR*. You may use the desired format by typing B (oct.), H (hex.) or D (dec.) after the value prior to *CR*. For example: 37777B CR.

CODE .... CR Write a single MC68000 instruction to the current memory locations, and - depending on the size of the instruction - the following locations.

You are not allowed to use any symbolic addresses/offsets in the instructions, and you are responsible for any overwritten instruction.

EXIT To leave one of the LOOK-AT commands and go back to the monitor, you type EXIT.

PERMIT-DEPOSIT In order to avoid unintended modification to the content of a memory location or a register, this command must be typed before you can do any changes.

EXTRA-FORMAT Within the LOOK-AT commands you may extend the formats used to display the contents of memory locations or the registers. These formats are in addition to the main display formats, and is valid only for the current LOOK-AT command.

The format can be any combination of the following:

BYTE, HALFWORD, WORD, ASCII, SYMBOL,

the options may be typed in fully or abbreviated, given on separate lines, or on the same line separated by commas or blanks.

The BYTE, HALFWORD, and WORD displays the contents as an 8, 16 or 32 bit binary value. ASCII displays the contents as an 8 bit character.

The SYMBOL option displays the symbolic name of the contents of the location, provided an exported symbol exist for this address, and this symbol is found in file read by a the LOAD-ENTRY-LIST command.

An HELP subcommand shows all available format options.

You may choose among five LOOK-AT commands:

LOOK-AT-DATA - to examine the data part of the memory

P-M: LOOK-AT-DATA <address>

The command may be used to look at, and to modify the data in the memory.

LOOK-AT-PROGRAM - to examine the program

P-M: LOOK-AT-PROGRAM <address>

The command may be used to look at, and to modify the program code in the memory.

LOOK-AT-REGISTER - to examine registers

P-M: LOOK-AT-REGISTER <register name>

The command may be used to look at, and to modify the contents of the registers. The default register name is PC (program counter).

Starting at register DO, the following registers are displayed sequentially, for example by typing CR between each display:

DO ... D7, AO ... A6, USP, SSP, PC, SR.

LOOK-AT-RELATIVE - display the memory relative to an address

P-M: LOOK-AT-RELATIVE <relative to: >

The command has a similar function such as LOOK-AT-DATA, and in addition to the memory address there is also the offset to the originally specified address written out in front of the memory contents. This command is useful when looking at data records or arrays, of which the start address is known.

LOOK-SYMBOLIC

P-M: LOOK-SYMBOLIC <symbolic address>

This command only accepts a symbolic address. It then calls LOOK-AT-PROGRAM if the entry represents a program entry; otherwise it calls LOOK-AT-DATA.

To look at a different type while you are in one of the LOOK-AT commands, you only need to enter the type desired:

DATA, PROGRAM, REGISTER or RELATIVE.

Thus you do not need to type EXIT followed by LOOK-AT-REGISTER, you can just type the word REGISTER at once.

6.6 The PROCESS-STATUS Command

PROCESS-STATUS - list status information for processes

P-M: PROCESS-STATUS <process-number>

The command lists the status of one or all processes. See examples below.

If you enter a process number, a full status report is given for this process.

If no process number is given, a short status line is reported for all processes.

Note that if the PIOC is still running, you will usually not get a reliable snapshot of the process(es); you have to stop PIOCOS first. Note also that most values are undefined for all processes that are dormant.

```
P-M: PROCESS-STATUS,,

Process     Status    Prio.  Curr.Ev.     Wait.Ev. Program-Cnt   SR   Time used
1  "FREE"  dormant     5    ----------------- not defined --------------------
2  "RTC "  suspended  14           0           20       47526    404          1
3  "PRO1"  active      1           0            0       77454    400       6953


P-M: PROCESS-STATUS,3

Process   3   "PRO1" is active
Priority             :           1
Events waiting for   :          0B
Current events       :          0B
Time used            :       7011

PC:         77454        SR:          400
D0:             1        A0:       200324
D1:             0        A1:       200204
D2:             0        A2:       200250
D3:           101        A3:       200326
D4:             1        A4:        77442
D5:        377501        A5:        53702
D6:             0        A6:       200302
D7:             0        A7:       203702
```

Figure 27. Process status display.

7 COMPILING, LOADING AND EXECUTING PIOC PROCESSES

Compiling, loading, and execution of user applications for PIOC, consist of several steps:

Compilation — you have to write and compile your own application programs for the PIOC.

System-load — with the PIOCOS Basic Software you got the object modules to load a basic operating system for your configuration. You can use one basic system for several of your applications.

Application-load — the basic system and your application modules are loaded together to form the complete user application system.

Executing — to load the whole PIOC software into the PIOC memory, and start the execution, you can use either the PIOC monitor call in SINTRAN III or commands in the PIOC-MONITOR.

7.1 Compiling PIOC Programs

User programs for the PIOC are written in PLANC. To compile the source code (:SYMB-files) you use the PLANC-MC68000 compiler. This compiler generates object code for programs to run in PIOC.

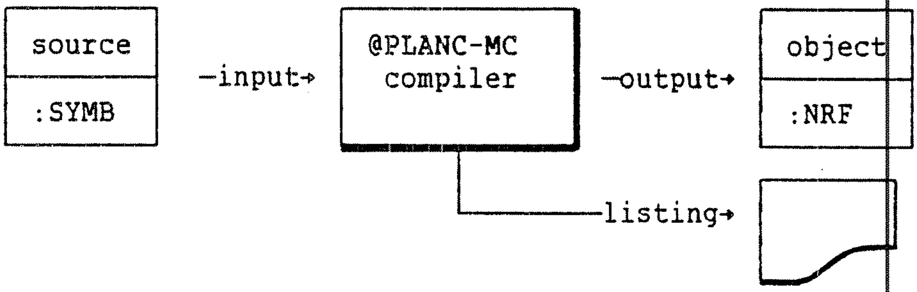


Figure 28. Compiling a program with the PLANC-MC68000 compiler.

The PLANC-MC68000 compiler is used just like any other language compiler on the ND-100/500 computers. You type

```
@PLANC-MC68000
```

or your installation's selected name, as a command to the operating system.

The compiler prompts with an asterisk (*), a HELP command lists the available commands and their parameters, the EXIT command returns control to the SINTRAN III operating system. You start compiling your source files with the command:

```
*COMPILE <source-file> <list-file> <object-file>
```

and several source files can be compiled by repeating the command. You should use the compiler command

```
$SEPARATE-DATA ON
```

to ensure that program code and data areas are separated. This will give you better protection and performance of your application. The command must be given before you start any compilation.

Furthermore, in your source file you should always use the $INCLUDE statement to incorporate the symbol definition files (:DEFS files) supplied with the PIOC Basic Software. Then you are able to make symbolic error checks, and your source file will remain up to date even if some values in PIOCOS may change.

The compiler produces an object file with the program in MC-68000 relocatable form (NRF: Nord Relocatable Format). The object file is made up of machine instructions, but without fixed addresses. It must be built into an executable program, a PIOC "domain", together with the PIOCOS operating system and additional user and system runtime libraries. This process, called linkage-loading is described on page 103.

If you write your own library with routines for the calls to PIOCOS, you should compile in library mode to get the most compact :NRF code. The LIBRARY-MODE command must be given before any COMPILE command. An example of such a library is shown on page 117. In the examples here, the user library here is called PIOCOS-LIB.

In the figure below all necessary commands are shown:

```
@PLANC-MC
*LIBRARY-MODE ON
*SEPARATE-DATA ON
*COMPILE USER-PROG,TERMINAL,"USER-PROG"
*COMPILE BUFFER-POOL,TERMINAL,"BUFFER-POOL"
*COMPILE PIOCOS-LIB,TERMINAL,"PIOCOS-LIB"
*EXIT
```

Example 11. Compiling a program with the PLANC-MC68000 compiler.

7.2 Global Labels AUTO_START, BUFFER_START and BUFFER_END

When loading user application programs to PIOCOS, some global labels will be defined: AUTO_START, BUFFER_START, and BUFFER_END. When starting the PIOC, PIOCOS will automatically begin execution at the address AUTO_START, a default in the START command. The process will run at priority level 1.

PIOCOS needs space in the PIOC memory for system tables and message space. This area is called the BUFFER_POOL and is limited by the global labels BUFFER_START and BUFFER_END.

The size of this area depends very much on your programs. You may define this area in a separate program module as shown below, and load it together with your program(s). In this way, it is easy to change the size of this area independent of the main program:

```
%  This module reserves 2000 words (16 bit)  buffer pool area
%  used by PIOCOS for system tables and messages.

MODULE BUFFER_POOL
EXPORT BUFFER_START,BUFFER_END
  INTEGER ARRAY: BUFFER_START (0:1999)
  INTEGER: BUFFER_END
ENDMODULE
$EOF
```

Example 12. Reserving a buffer pool for PIOCOS.

When loading PIOCOS modules there is an alternative way of defining the size of the buffer pool. See question 8 on page 99, and question 9 on page 105.

7.3 Loading a PIOCOS Basic System

Loading the PIOCOS Basic System is done by running the program

```
@PIOC-GENERAT-C
```

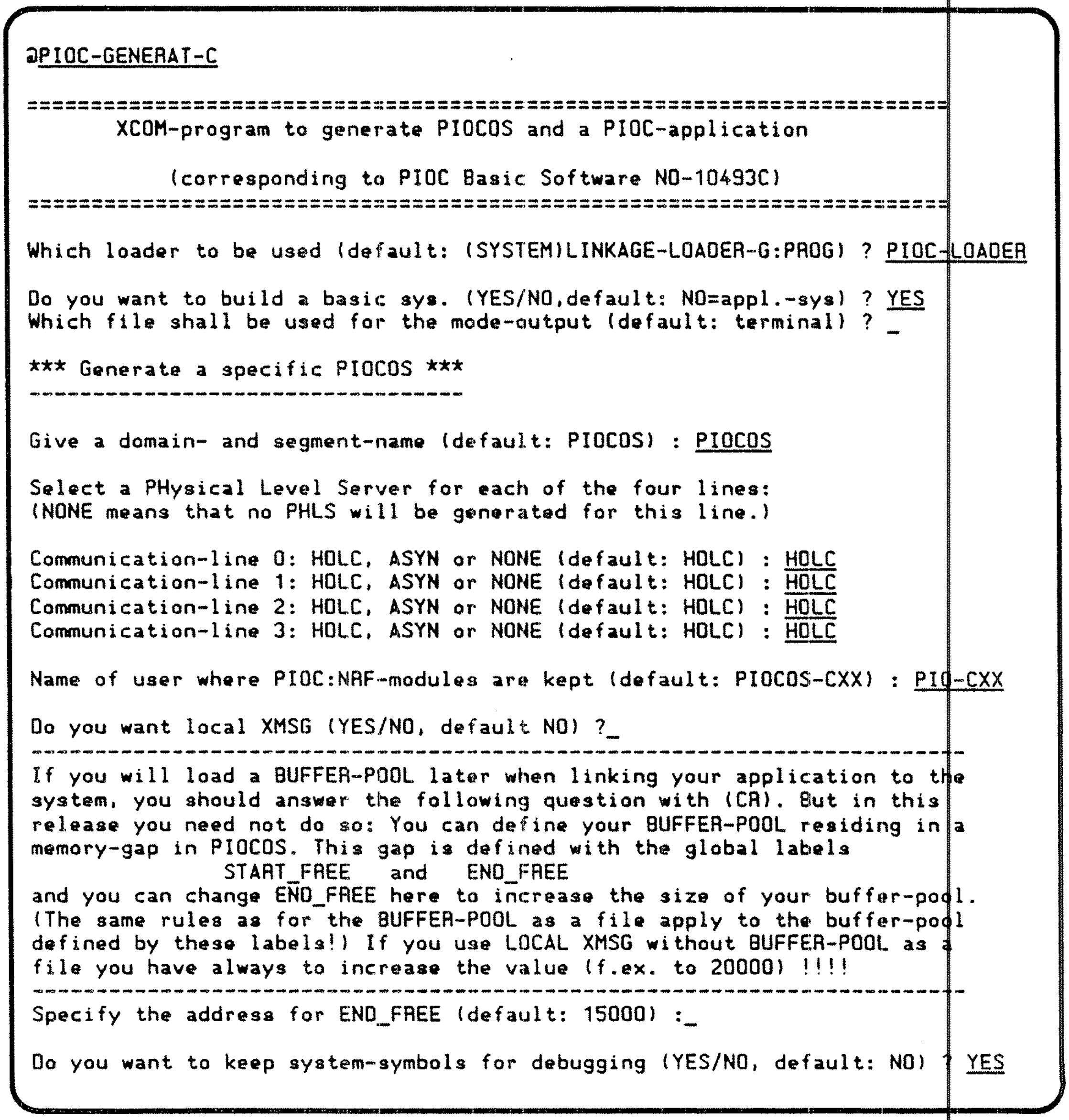
The program asks for special options and types of communication lines for the basic system you will generate, and produces a mode file which is started after the final question. When the mode job terminates, the file will be deleted.

For special purposes it is possible to produce the mode file without starting it. This can be done by giving the no-execution option on the command line:

```
@PIOC-GENERAT-C /NOEX
```

The name of the mode file will be written on the terminal. If required the file can be changed with an editor to fit special needs. But remember to write the changed file back to a new file. Otherwise, the next run of PIOC-GENERAT-C from the same terminal will overwrite your modifications.

On the next page, you find an example of a full dialog of such a PIOCOS Basic System Loading. The numbers refers to the questions and answers on the following pages. All user input is underlined.



```
  @PIOC-GENERAT-C

  ===================================================================
        XCOM-program to generate PIOCOS and a PIOC-application

             (corresponding to PIOC Basic Software ND-10493C)
  ===================================================================

1 Which loader to be used (default: (SYSTEM)LINKAGE-LOADER-G:PROG) ? PIOC-LOADER

2 Do you want to build a basic sys. (YES/NO,default: NO=appl.-sys) ? YES
3 Which file shall be used for the mode-output (default: terminal) ? _

  *** Generate a specific PIOCOS ***
  ----------------------------------

4 Give a domain- and segment-name (default: PIOCOS) : PIOCOS

  Select a PHysical Level Server for each of the four lines:
  (NONE means that no PHLS will be generated for this line.)

5 Communication-line 0: HDLC, ASYN or NONE (default: HDLC) : HDLC
  Communication-line 1: HDLC, ASYN or NONE (default: HDLC) : HDLC
  Communication-line 2: HDLC, ASYN or NONE (default: HDLC) : HDLC
  Communication-line 3: HDLC, ASYN or NONE (default: HDLC) : HDLC

6 Name of user where PIOC:NRF-modules are kept (default: PIOCOS-CXX) : PIO-CXX

7 Do you want local XMSG (YES/NO, default NO) ?_
  --------------------------------------------------------------------
  If you will load a BUFFER-POOL later when linking your application to the
  system, you should answer the following question with (CR). But in this
  release you need not do so: You can define your BUFFER-POOL residing in a
  memory-gap in PIOCOS. This gap is defined with the global labels
                    START_FREE   and   END_FREE
  and you can change END_FREE here to increase the size of your buffer-pool.
  (The same rules as for the BUFFER-POOL as a file apply to the buffer-pool
  defined by these labels!) If you use LOCAL XMSG without BUFFER-POOL as a
  file you have always to increase the value (f.ex. to 20000) !!!!
  --------------------------------------------------------------------
8 Specify the address for END_FREE (default: 15000) :_

9 Do you want to keep system-symbols for debugging (YES/NO, default: NO) ? YES
```

Example 13. Generating a PIOCOS Basic System using PIOC-GENERATE-C

The type of questions asked, and the type of answers you must give is shown on the next pages, followed by a listing from the execution of the mode file.

Question 1: Which loader to be used?

What to do: Give the name of the ND Linkage-Loader which should be used. Default is the G-version running on the ND-100.

Question 2: Do you want to build a basic system (YES, otherwise application-sys)?

What to do: Type Y and <CR>

Question 3: Which file shall be used for the output from the mode-job (Default: terminal) ?

What to do: When the question is asked, PIOC-GENERAT-C:PROG will start execution of the produced mode file. As in the @MODE command the output from the job can be written to a file instead of the terminal.

Type the name of the file (enclosed in ".." if it is a new file) where you want to save the listing of all symbolic entries, otherwise <CR>.

Question 4: Give a domain and segment name (default: PIOCOS):

What to do: Specify a suitable name for your basic system, for example HDLC-PIOCOS. The name you give will be used for both the domain and the segment file.

Question 5: Communication-line 0: HDLC, ASYN or NONE (default: HDLC):

What to do: This question will be repeated for each of the four communication lines, 0 to 3. You should select a type of driver for each line: HDLC (synchronous), ASYN (asynchronous), or NONE.

If you answer NONE, no driver will be included for this line, and a reference NONE_<x> (where <x> is the line number) will remain undefined in your system. This will not affect the operation of your PIOCOS basic system.

Note that the default type will be HDLC for line 0, but beginning with line 1 the default will be the type that you specified on the previous line.

Abbreviation of the type names are not allowed.

Question 6: Name of user where PIOC:NRF modules are kept (default: UTILITY):

What to do: You have to answer with the user name, where all the PIOC :NRF files are stored.

Question 7: Do you want local XMSG (YES, otherwise no local XMSG)?

What to do: If you want to use local XMSG for communication between the PIOC processes, type YES. If NO, or <CR> is typed, no local XMSG system will be included.

Note that the XMSG system for communicating with RT-programs on different computers can always be reached from PIOCOS, if it is available on the ND-100.

Question 8: Specify the address for END_FREE (Default: 15000B) :

What to do: Since code and data are separated in release C there is a memory gap between the end of PIOCOS data area and the start of the PIOCOS code.

The size of this gap can be increased by the user with the value of END_FREE. (START_FREE and END_FREE are global labels, which means that they are exported from the PIOCOS Basic System to the application loading.)

When loading a user application (as shown on page 105), you will be asked for the name of a buffer pool. If you there answer with only <CR>, the labels BUFFER_START and BUFFER_END will be defined with the values of START_FREE and END_FREE, respectively.

In this way, the memory gap will be used instead of a user defined buffer pool.

The value 15000B is chosen to make sure that under no circumstances the data will overlap the code.

But if you are going to generate a big PIOC (ASYN and HDLC, all communication lines used, local XMSG included, many processes) then this value will be too small for the gap/buffer pool and the PIOCOS will not work. Try a large value, for example 20000B.

Question 9: Do you want to keep system-symbols for debugging (Default: NO) ?

What to do: If you need all the PIOCOS system labels, you should answer with Y(es). Otherwise, <CR> will exclude such labels from the listing.

If you answer Y you can set symbolic breakpoints inside PIOCOS. Of course, this requires very good knowledge of the internal structure of PIOCOS or special support from ND.

With the answer NO or <CR> only a very few PIOCOS-labels are available after the application loading.

Also because of the limited size of the internal symbol table, you are recommended not to include the PIOCOS labels.

Beware also that if these symbols are kept, name collisions may occur.

The mode file generated and started by the PIOC-GENERAT-C program will produce similar listing as on the following pages:

```
@ENTER,,,20

@PIOC-LOADER

ND-Linkage-Loader - G          1. May        1984 Time:  0:00
Nll entered:                  23. April      1985 Time: 15:32
Nll: cc *** Generate a specific PIOCOS as PIOCOS   %
Nll: abort-batch-on-error off                      % terminate job if error
Nll: computer-mode PIOC                            % set PIOC link-load mode
Nll: release-domain PIOCOS                         % close domain if open
Nll: delete-domain PIOCOS                          % remove previous version

Nll: set-dom "PIOCOS",,                            % create new domain
Nll: define-entry NIL,0,D
Nll: open-segment "PIOCOS",,,                      % create new segment
Nll: local-trap-enable XX AO-Z                     % local trap enable
Nll: define-entry XX,0,P                           % define trap symbol
Nll: low-add  2000B,D                              % set load-address
Nll: low-add 15000B,P                              % set load-address
Nll: define-entr END_FREE,#PCLC,D
Nll: load (PIO-CXX)PIOC-NCOMM-C:NRF                % load first part of PIOCOS
Program:.......17344 P01   Data:...........2634 D01
Nll: prog-ref HDLC_0,0,p                           % define type of comm line 0
Nll: prog-ref HDLC_1,0,p                           % define type of comm line 1
Nll: prog-ref HDLC_2,0,p                           % define type of comm line 2
Nll: prog-ref HDLC_3,0,p                           % define type of comm line 3
Nll: prog-ref NO_LOCAL_X,0,p                       % include local xmsg if wanted
Nll: prog-ref NO_ETHERNE,0,p                       % define ethernet if wanted
Nll: load (PIO-CXX)PIOC-HDLC-C:NRF                 % load PHYSICAL LEVEL SERVER HDLC
Program:.......24712 P01   Data:...........4240 D01
Nll: cc                                            % depending on type selected
Nll: load (PIO-CXX)PIOC-ASYNC-C:NRF                % load PHYSICAL LEVEL SERVER ASYNC
Program:.......24712 P01   Data:...........4300 D01
Nll: cc                                            % depending on type selected
Nll: load (PIO-CXX)PIOC-XMSG-C:NRF                 % load XMSG part
Program:.......24712 P01   Data:...........4340 D01
Nll: load (PIO-CXX)PIOC-XROUT-C:NRF                % load XROUT part
Program:.......24712 P01   Data:...........4400 D01
Nll: define-entry CUR_D,#DCLC,D                    % define current load-address
Nll: low-add 0 0                                   % set load-address 0 (zero)
Nll: load (PIO-CXX)PIOC-TRAPV-C:NRF                % load trap handler vector
Program:.......25306 P01   Data:...........2000 D
Nll: low-add CUR_D,D                               % reset current load-address
Nll: load (PIO-CXX)PIOC-MAIN-C:NRF                 % load serv.call manager
Program:.......37342 P01   Data:...........7670 D01
Nll: load (PIO-CXX)PIOC-MEMA-C:NRF                 % load memory-management
Program:.......41150 P01   Data:...........7744 D01
Nll: load (PIO-CXX)PIOC-PHLS-C:NRF                 % load remaining part of PHLS
Program:.......42202 P01   Data:..........10104 D01
Nll: load (PIO-CXX)PIOC-CLOCK-C:NRF                % load clock handler
Program:.......45500 P01   Data:..........12734 D01
Nll: load (PIO-CXX)PIOC-SHORT-C:NRF                % load short-circuit modules
Program:.......45542 P01   Data:..........13004 D01
Nll: load (PIO-CXX)PLANC-MC-18:NRF                 % load PLANC runtime library
MOTOROLA-RUN-831027
Program:.......50404 P01   Data:..........13004 D01
Nll: define-entr START_FREE,#DCLC,D
Nll: define-entr END_SYSTEM,#PCLC,P                % define current load-address
```

```
N11: close-segment Y                                        % stop loading PIOCOS segment
END_PIOCOS.....30272 P01   AUTO_START.....30614 P01
BUFFER_END.....30342 P01   BUFFER_STA.....30364 P01
BUFFER_STA.....37356 P01   BUFFER_END.....37440 P01

Undefined entries on the last used segment

------------------------------------------------------------------------------
23. April      1985 Time: 15:33
Unsatisfied references :

END_PIOCOS.....30272 P01   AUTO_START.....30614 P01
BUFFER_END.....30342 P01   BUFFER_STA.....30364 P01
BUFFER_STA.....37356 P01   BUFFER_END.....37440 P01
------------------------------------------------------------------------------

Defined symbols :

XX.................0 P     UERRFATAL......15226 P
TESTFATAL......15312 P     TLEVEL_6.......15400 P     START..........16402 P
TLEVEL_7.......16734 P     TRAP_10........17004 P     TTRACE.........17074 P
SYN_COMMON.....17236 P     NO_ETHERNE.....17344 P
SI001A_INT.....17364 P     HDLC_0.........17364 P
SI002A_INT.....17400 P     SI003A_INT.....17414 P
SI010A_INT.....17430 P     SI011A_INT.....17444 P
SI001B_INT.....17460 P     HDLC_1.........17460 P
SI002B_INT.....17474 P     SI003B_INT.....17510 P
SI010B_INT.....17524 P     SI011B_INT.....17540 P
SI021A_INT.....17554 P     HDLC_2.........17554 P
SI022A_INT.....17570 P     SI023A_INT.....17604 P
SI031A_INT.....17620 P     SI030A_INT.....17634 P
SI021B_INT.....17650 P     HDLC_3.........17650 P
SI022B_INT.....17664 P     SI023B_INT.....17700 P
SI031B_INT.....17714 P     SI030B_INT.....17730 P
START_TRAN.....17744 P     HDLC_O_VO......20204 P
POSSIBLE_S.....20310 P     SERIN..........20352 P     CONNECT........20524 P
REC_DISCON.....20576 P     TRANSM_DIS.....20722 P
HDLC_DOTRA.....21010 P     HDLC_O_V1......21166 P
RESTART_RE.....21520 P     START_RECE.....21564 P
HDLC_COMMO.....21720 P     HDLC_I_V1......22136 P
HDLC_I_V2......22300 P     HDLC_I_V3......22370 P     INCT_37........22612 P
HDLC_DOREC.....23130 P     HDLC_DOCON.....23370 P     TBUS...........24712 P
TADDR..........24730 P     TILIN..........24746 P     TZERO..........24762 P
TCHECK.........24776 P     TTRAPV.........25012 P     TPRIV..........25026 P
T1010..........25042 P     T1111..........25056 P     T24............25072 P
T25............25106 P     T26............25122 P     T27............25136 P
T28............25152 P     T29............25166 P     T30............25202 P
T31............25216 P     TNOTUSED.......25232 P     INCT_30........25246 P
ASYNC_SCHE.....25306 P     N100XMSG.......25412 P     NXMSG..........25620 P
CHECK_SLOT.....26052 P     YKICK..........26530 P     INCT_34........27102 P
YSETEV.........27234 P     SYN_RETURN.....27374 P     BREAK..........27500 P
UINIT..........27652 P     UFIND_PNAM.....31034 P     UFIND_PO.......31346 P
UEND...........31474 P     UEVOK..........32254 P     UWAITEV........32416 P
SCHEDULER......32736 P     ST_USER........33102 P     YMON2..........33346 P
YCREATE........33752 P     YBEGIN.........34606 P     YKILL..........35040 P
YEND...........35176 P     YABORT.........35244 P     YNAME..........35506 P
YWAITEV........35664 P     YSELWAITEV.....35770 P     YREADEV........36074 P
PIOCTRAP.......36706 P     ZINIM..........37342 P     ZGETM..........37710 P
ZRELM..........40476 P     YPHLS..........41754 P
FIX_TIME_C.....42202 P     TIM_MOD........42260 P     ACCOUNT........42444 P
RTCDRIV........42464 P     RTCINI.........42650 P     AT_CLOCK.......43252 P
UCONNECT.......43456 P     UDISCONNEC.....44152 P     UKICK..........44314 P
UREM_EVQ.......44456 P     YINTEREV.......44562 P
YINTERDEL......45212 P     YRTKICK........45406 P     INCT_36........45500 P
LOCAL_XMSG.....45502 P     XRSTART........45502 P     XMINI..........45502 P
MFXMSG.........45506 P     NO_LOCAL_X.....45542 P     MON64..........50344 P
MON65..........50374 P     MONO...........50400 P
END_SYSTEM.....50404 P
NIL................0 D     TRAP_LOCS..........4 D     MBOX...........2176 D
WAKEBOX.........2202 D     PIOC_MAP........2206 D     CODE...........2300 D
```

```
TRAP_HANDL......2302 D     MAILBOX_ST......2574 D
O_0CHANNEL......2634 D     I_0CHANNEL......2724 D
C_S_0CHANN......3014 D     O_1CHANNEL......3044 D
I_1CHANNEL......3134 D     C_S_1CHANN......3224 D
O_2CHANNEL......3254 D     I_2CHANNEL......3344 D
C_S_2CHANN......3434 D     O_3CHANNEL......3464 D
I_3CHANNEL......3554 D     C_S_3CHANN......3644 D
HDLC_OINIT......3674 D     HDLC_IINIT......3716 D
HDLC_STATE......4200 D     ASYNC_STAT......4240 D
XMSG_STATE......4300 D     XROUT_STAT......4340 D
PIOC_NUMBE......4400 D     CUR_D...........4400 D
ND100_CPU.......4402 D     C_XCTPT.........4404 D     C_MOTOR.........4424 D
PROC_TABLE......6506 D     MKICK_TAB.......7156 D
MAIN_STATE......7630 D     MEMA_STATE......7704 D
PHLS_TABLE......7744 D     PHLS_STATE.....10044 D     REALTIME.......11304 D
CLOCK_STAT.....12674 D     SHORT_STAT.....12734 D     RTC_DIV........12774 D
WATCH_DOG......13000 D     START_FREE.....13004 D     END_FREE.......15000 D

Program:.......50404 P01   Data:..........13004 D01

"BPUN"-code is generated
Lower bound:            0
Number of words:    24203
N11: exit

@cc PIOCOS is now ready.
```

Example 14. Output from loading a PIOCOS Basic System

Note that there may remain some undefined references after the link. They will be resolved when loading the user application later.

If you do not generate driver for a communication line, the symbol NONE_<x> will remain undefined, to remind you that there is no driver loaded for that line. (The <x> refers to the line numbers 0,1, 2, or 3.)

Such references will remain undefined throughout all future application loadings, and will even be written out by the LOAD-ENTRY-LIST command. These particular symbols being undefined, will not affect the operation of PIOCOS.

7.4 Loading an Application System

After you have compiled your application programs and produced a suitable basic system, you have to combine both parts. This is done by the same program PIOC-GENERAT-C just like generating the basic system as explained in the previous section.

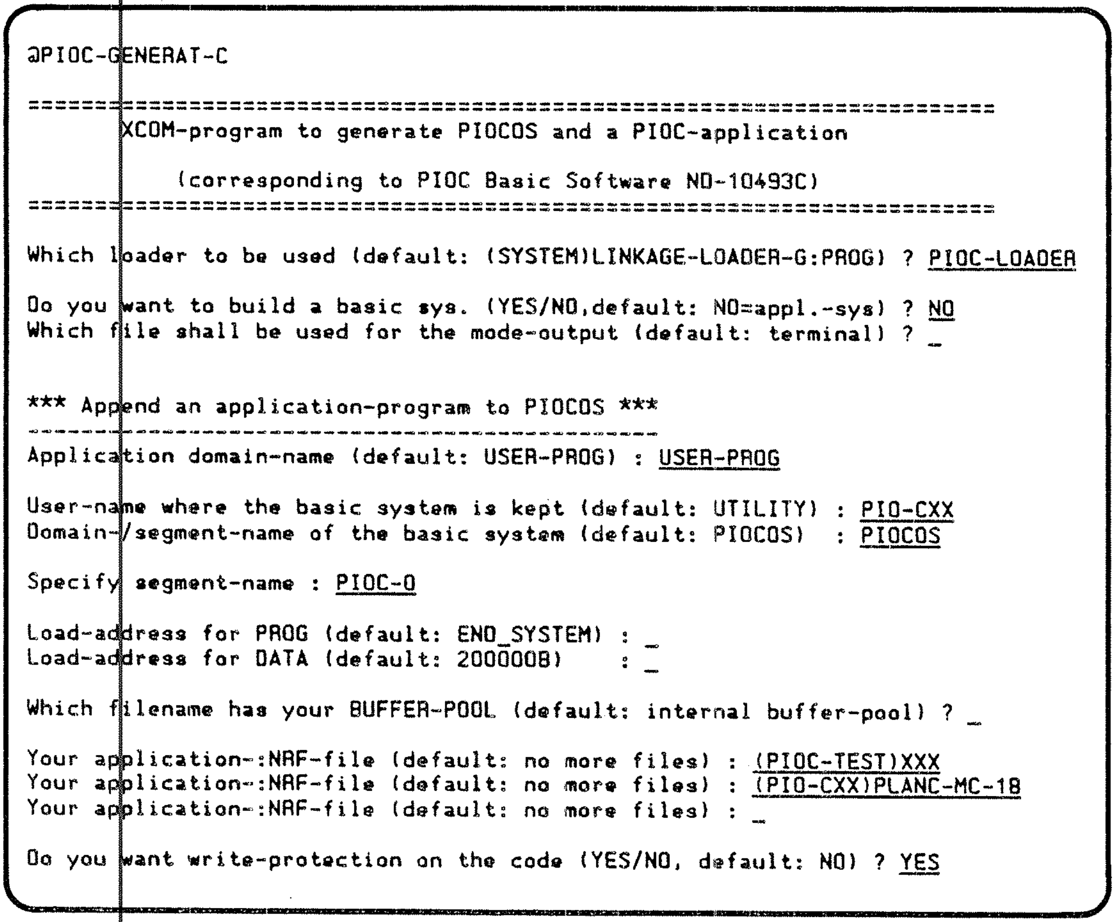
```
@PIOC-GENERAT-C
```

The program asks for the name of the basic system generated earlier, the name of your object files, and generates a mode file with all necessary commands. When the final question is answered, the program starts the execution of this mode file. Successful termination of the mode-job deletes the file.

You may inhibit the execution of the mode job by giving the non-execution option to the PIOC-GENERAT-C program:

```
@PIOC-GENERAT-C /NOEX
```

On the next page you will find a full dialog of such a PIOCOS application loading. The numbers refers to the questions/answers on the following pages. All user input is underlined.



```
     @PIOC-GENERAT-C

     ===================================================================
           XCOM-program to generate PIOCOS and a PIOC-application

               (corresponding to PIOC Basic Software ND-10493C)
     ===================================================================

1    Which loader to be used (default: (SYSTEM)LINKAGE-LOADER-G:PROG) ? PIOC-LOADER

2    Do you want to build a basic sys. (YES/NO,default: NO=appl.-sys) ? NO
3    Which file shall be used for the mode-output (default: terminal) ? _


     *** Append an application-program to PIOCOS ***
     -----------------------------------------------
4    Application domain-name (default: USER-PROG) : USER-PROG

5    User-name where the basic system is kept (default: UTILITY) : PIO-CXX
6    Domain-/segment-name of the basic system (default: PIOCOS)  : PIOCOS

7    Specify segment-name : PIOC-0

8    Load-address for PROG (default: END_SYSTEM) : _
9    Load-address for DATA (default: 200000B)    : _

10   Which filename has your BUFFER-POOL (default: internal buffer-pool) ? _

11   Your application-:NRF-file (default: no more files) : (PIOC-TEST)XXX
     Your application-:NRF-file (default: no more files) : (PIO-CXX)PLANC-MC-18
     Your application-:NRF-file (default: no more files) : _

12   Do you want write-protection on the code (YES/NO, default: NO) ? YES
```

Example 15. Loading a user application to the PIOCOS Basic System

The questions are explained on the following pages:

Question 1: Which loader to be used?

What to do: Give the name of the Linkage-Loader which should be used during the loading. Default is the G-version running on the ND-100.

Question 2: Do you want to build a basic system (YES, otherwise application sys)?

What to do: Answer NO and <CR>

Question 3: Which file shall be used for the mode output (default: terminal):

What to do: Give the name of a file to receive the result of the mode job.

Question 4: Application domain-name (default: USER-PROG):

What to do: Give a suitable name for the domain of your application, for example SNA-HASP-WS. The name you give will be used for both the domain and the segment.

Question 5: User name where the basic system is kept (default: UTILITY):

What to do: You have to answer with the name of the user which stores all the PIOC :NRF-files.

Question 6: Domain/segment name of the basic system (default: PIOCOS)

What to do: Give the name of the basic system.

Question 7: Specify a segment name.

What to do: Specify a name of your choice for the linkage-loader segment.

Question 8: Load address for PROG (default: END_SYSTEM) :

What to do: You have to specify the octal address, where to load the code of your application modules. Because the symbol END_SYSTEM is known from the basic system load, PIOC-GENERAT-C offers it as a default value. Choosing this value will give you the best memory utilization.

Question 9: Load address for DATA (default: 200000B):

What to do: Choose an address for your data area, not overlapping with the code. (You probably have to do a test run of the loading first, to find out a value that fits.) The default value points to the start of the second internal PIOC DMA-bank.

In the first version of the PIOC hardware there was no DMA possible from the first memory bank.

Question 10: Which file name has your BUFFER-POOL (Default: internal buffer pool) ?

What to do: If you want a special file for your buffer pool, you have to answer with the name of this :NRF file. Note that the entries BUFFER_STA and BUFFER_END must be exported from this file.

If you answer with <CR> only, the gap in PIOCOS is used instead (see question 7 in the section Loading a PIOCOS Basic System on page 99).

Question 10: Your application :NRF file (default: no more files) :

What to do: Give names, one by one, of the object file you want to load. The question is repeated until a single carriage return is typed.

Question 11: Do you want write-protection on the code (YES/NO, default: NO)?

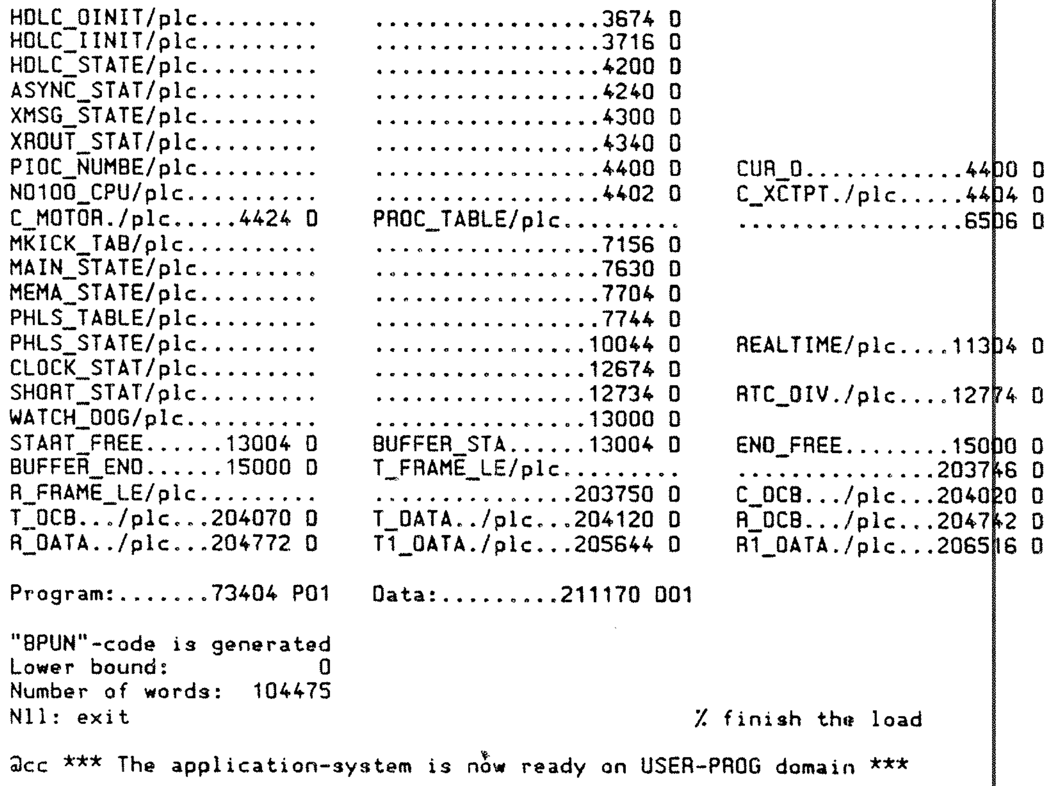
What to do: If your application program is compiled with the $SEPARATE-MODE ON, the code part is loaded just behind the PIOCOS code part. You can specify that this part is included in the write-protection area of PIOCOS.

If your program tries to modify its own code, an error message will be generated from the PIOC-MONITOR.

The following listing shows the result from the execution of the mode file:

```
@ENTER,,,20

@PIOC-LOADER

ND-Linkage-Loader -  G          1. May        1984 Time:  0:00
Nll entered:                  23. April      1985 Time: 15:42
Nll: cc *** Append an application-program to PIOCOS ***   %
Nll: abort-batch-on-error off                     % terminate job if error
Nll: computer-mode PIOC                           % set PIOC link-mode
Nll: release-domain USER-PROG                     % close domain if open
Nll: delete-domain USER-PROG                      % delete domain

Nll: copy-domain (PIO-CXX)PIOCOS,"USER-PROG"      % get standard PIOCOS domain
Nll: define-entry NIL,0,D
Nll: set-domain USER-PROG                         % set domain active
Nll: rename-segment PIOCOS PIOC-0                 % rename old seg.name to new
Nll: Nll: append-segment PIOC-0,,                 % prepare for further loading

WARNING: redefinition of NIL =           0 is ignored
Program:.......50404 P     Data:..........13004 D
Nll: low-address END_SYSTEM,P                     % set PROG address
Nll: low-address 200000B,D                        % set DATA address
Nll: define-entry BUFFER_STA,START_FREE,D
Nll: define-entry BUFFER_END,END_FREE,D
Nll: load (PIOC-TEST)XXX                          % load specified user file
Program:.......65420 P     Data:.........211165 D01
Nll: load (PIO-CXX)PL--1                          % load specified user file
MOTOROLA-RUN-831027
Program:.......73404 P01   Data:.........211170 D01
Nll: define-entry END_PIOCOS,#PCLC,P
Nll: system-entries ON                            % skip system.def.labels
Nll: close-segment Y
_______________________________________________________________________________
23. April      1985 Time: 15:43


Unsatisfied references :

None!
_______________________________________________________________________________

Defined symbols :

XX.................0 P   UERRFATAL/plc.........   .................15226 P
TESTFATAL/plc.........   ................15312 P   TLEVEL_6/plc....15400 P
START.../plc....16402 P   TLEVEL_7/plc....16734 P   TRAP_10./plc....17004 P
TTRACE../plc....17074 P   SYN_COMMON/plc.........   ................17236 P
NO_ETHERNE/plc.........   ................17344 P
SI001A_INT/plc.........   ................17364 P   HDLC_0../plc....17364 P
SI002A_INT/plc.........   ................17400 P
SI003A_INT/plc.........   ................17414 P
SI010A_INT/plc.........   ................17430 P
SI011A_INT/plc.........   ................17444 P
SI001B_INT/plc.........   ................17460 P   HDLC_1../plc....17460 P
SI002B_INT/plc.........   ................17474 P
SI003B_INT/plc.........   ................17510 P
SI010B_INT/plc.........   ................17524 P
SI011B_INT/plc.........   ................17540 P
SI021A_INT/plc.........   ................17554 P   HDLC_2../plc....17554 P
SI022A_INT/plc.........   ................17570 P
SI023A_INT/plc.........   ................17604 P
SI031A_INT/plc.........   ................17620 P
SI030A_INT/plc.........   ................17634 P
SI021B_INT/plc.........   ................17650 P   HDLC_3../plc....17650 P
SI022B_INT/plc.........   ................17664 P
SI023B_INT/plc.........   ................17700 P
SI031B_INT/plc.........   ................17714 P
SI030B_INT/plc.........   ................17730 P
```

```
START_TRAN/plc.........   ................17744 P
HDLC_O_V0/plc..........   ................20204 P
POSSIBLE_S/plc.........   ................20310 P   SERIN.../plc....20352 P
CONNECT./plc....20524 P   REC_DISCON/plc.........   ................20576 P
TRANSM_DIS/plc.........   ................20722 P
HDLC_DOTRA/plc.........   ................21010 P
HDLC_O_V1/plc..........   ................21166 P
RESTART_RE/plc.........   ................21520 P
START_RECE/plc.........   ................21564 P
HDLC_COMMO/plc.........   ................21720 P
HDLC_I_V1/plc..........   ................22136 P
HDLC_I_V2/plc..........   ................22300 P
HDLC_I_V3/plc..........   ................22370 P   INCT_37./plc....22612 P
HDLC_DOREC/plc.........   ................23130 P
HDLC_DOCON/plc.........   ................23370 P   TBUS..../plc....24712 P
TADDR.../plc....24730 P   TILIN.../plc....24746 P   TZERO.../plc....24762 P
TCHECK../plc....24776 P   TTRAPV../plc....25012 P   TPRIV.../plc....25026 P
T1010.../plc....25042 P   T1111.../plc....25056 P   T24...../plc....25072 P
T25...../plc....25106 P   T26...../plc....25122 P   T27...../plc....25136 P
T28...../plc....25152 P   T29...../plc....25166 P   T30...../plc....25202 P
T31...../plc....25216 P   TNOTUSED/plc....25232 P   INCT_30./plc....25246 P
ASYNC_SCHE/plc.........   ................25306 P   N100XMSG/plc....25412 P
NXMSG.../plc....25620 P   CHECK_SLOT/plc.........   ................26052 P
YKICK.../plc....26530 P   INCT_34./plc....27102 P   YSETEV../plc....27234 P
SYN_RETURN/plc.........   ................27374 P   BREAK.../plc....27500 P
UINIT.../plc....27652 P   UFIND_PNAM/plc.........   ................31034 P
UFIND_PO/plc....31346 P   UEND..../plc....31474 P   UEVOK.../plc....32254 P
UWAITEV./plc....32416 P   SCHEDULER/plc..........   ................32736 P
ST_USER./plc....33102 P   YMON2.../plc....33346 P   YCREATE./plc....33752 P
YBEGIN../plc....34606 P   YKILL.../plc....35040 P   YEND..../plc....35176 P
YABORT../plc....35244 P   YNAME.../plc....35506 P   YWAITEV./plc....35664 P
YSELWAITEV/plc.........   ................35770 P   YREADEV./plc....36074 P
PIOCTRAP/plc....36706 P   ZINIM.../plc....37342 P   ZGETM.../plc....37710 P
ZRELM.../plc....40476 P   YPHLS.../plc....41754 P
FIX_TIME_C/plc.........   ................42202 P   TIM_MOD./plc....42260 P
ACCOUNT./plc....42444 P   RTCDRIV./plc....42464 P   RTCINI../plc....42650 P
RT_CLOCK/plc....43252 P   UCONNECT/plc....43456 P
UDISCONNEC/plc.........   ................44152 P   UKICK.../plc....44314 P
UREM_EVQ/plc....44456 P   YINTEREV/plc....44562 P
YINTERDEL/plc..........   ................45212 P   YRTKICK./plc....45406 P
INCT_36./plc....45500 P   LOCAL_XMSG/plc.........   ................45502 P
XRSTART./plc....45502 P   XMINI.../plc....45502 P   MFXMSG../plc....45506 P
NO_LOCAL_X/plc.........   ................45542 P   #BCPC.../plc....45542 P
#MOVE.../plc....46202 P   #IMU..../plc....46662 P   #IDV..../plc....47020 P
#INSE.../plc....47256 P   #APPD.../plc....47274 P   #REMV.../plc....47326 P
#INIT.../plc....47372 P   #ENTR.../plc....47466 P   #LEAV.../plc....47530 P
#ERET.../plc....47550 P   #PRERR../plc....47730 P   MON64.../plc....50344 P
MON65.../plc....50374 P   #QUIT.../plc....50400 P   MON0..../plc....50400 P
INREAD../plc....50404 P   END_SYSTEM......50404 P
CALL_MANAG/plc.........   ................52434 P   FINISHED/plc....52746 P
CHECK.../plc....53722 P   SKRIVT../plc....54060 P   SKRIV.../plc....54350 P
AUTO_START/plc.........   ................56030 P   #INBY.../plc....65420 P
#UTBY.../plc....66220 P   #UTI4.../plc....67400 P   #SPASI../plc....72116 P
#ERROR../plc....72426 P   #GETNO../plc....72624 P   #OUTBYT./plc....73032 P
#INBYT../plc....73104 P   #IMOD.../plc....73176 P   MON1..../plc....73276 P
MON2..../plc....73342 P   END_PIOCOS......73404 P
NIL..................0 D   TRAP_LOCS/plc.........   ....................4 D
MBOX..../plc.....2176 D   WAKEBOX./plc.....2202 D   PIOC_MAP/plc.....2206 D
CODE..../plc.....2300 D   TRAP_HANDL/plc........   .................2302 D
MAILBOX_ST/plc.........   .................2574 D
O_0CHANNEL/plc.........   .................2634 D
I_0CHANNEL/plc.........   .................2724 D
C_S_0CHANN/plc.........   .................3014 D
O_1CHANNEL/plc.........   .................3044 D
I_1CHANNEL/plc.........   .................3134 D
C_S_1CHANN/plc.........   .................3224 D
O_2CHANNEL/plc.........   .................3254 D
I_2CHANNEL/plc.........   .................3344 D
C_S_2CHANN/plc.........   .................3434 D
O_3CHANNEL/plc.........   .................3464 D
I_3CHANNEL/plc.........   .................3554 D
C_S_3CHANN/plc.........   .................3644 D
```



```
HDLC_OINIT/plc.........   .................3674 D
HDLC_IINIT/plc.........   .................3716 D
HDLC_STATE/plc.........   .................4200 D
ASYNC_STAT/plc.........   .................4240 D
XMSG_STATE/plc.........   .................4300 D
XROUT_STAT/plc.........   .................4340 D
PIOC_NUMBE/plc.........   .................4400 D   CUR_D.............4400 D
ND100_CPU/plc..........   .................4402 D   C_XCTPT./plc.....4404 D
C_MOTOR./plc.....4424 D   PROC_TABLE/plc.........   .................6506 D
MKICK_TAB/plc..........   .................7156 D
MAIN_STATE/plc.........   .................7630 D
MEMA_STATE/plc.........   .................7704 D
PHLS_TABLE/plc.........   .................7744 D
PHLS_STATE/plc.........   ................10044 D   REALTIME/plc....11304 D
CLOCK_STAT/plc.........   ................12674 D
SHORT_STAT/plc.........   ................12734 D   RTC_DIV./plc....12774 D
WATCH_DOG/plc..........   ................13000 D
START_FREE......13004 D   BUFFER_STA......13004 D   END_FREE........15000 D
BUFFER_END......15000 D   T_FRAME_LE/plc.........   ...............203746 D
R_FRAME_LE/plc.........   ...............203750 D   C_DCB.../plc...204020 D
T_DCB.../plc...204070 D   T_DATA../plc...204120 D   R_DCB.../plc...204742 D
R_DATA../plc...204772 D   T1_DATA./plc...205644 D   R1_DATA./plc...206516 D

Program:.......73404 P01  Data:.........211170 D01

"BPUN"-code is generated
Lower bound:           0
Number of words:  104475
N11: exit                                         % finish the load

@cc *** The application-system is now ready on USER-PROG domain ***
```

Example 16. Output from loading a user application to PIOCOS

7.5 The Procedures of Fixing and Loading The PIOC Memory

There are three ways to load the PIOC memory with its final contents, shown as the paths A, B and C in the illustration below. The next three sections explain how to carry them out.

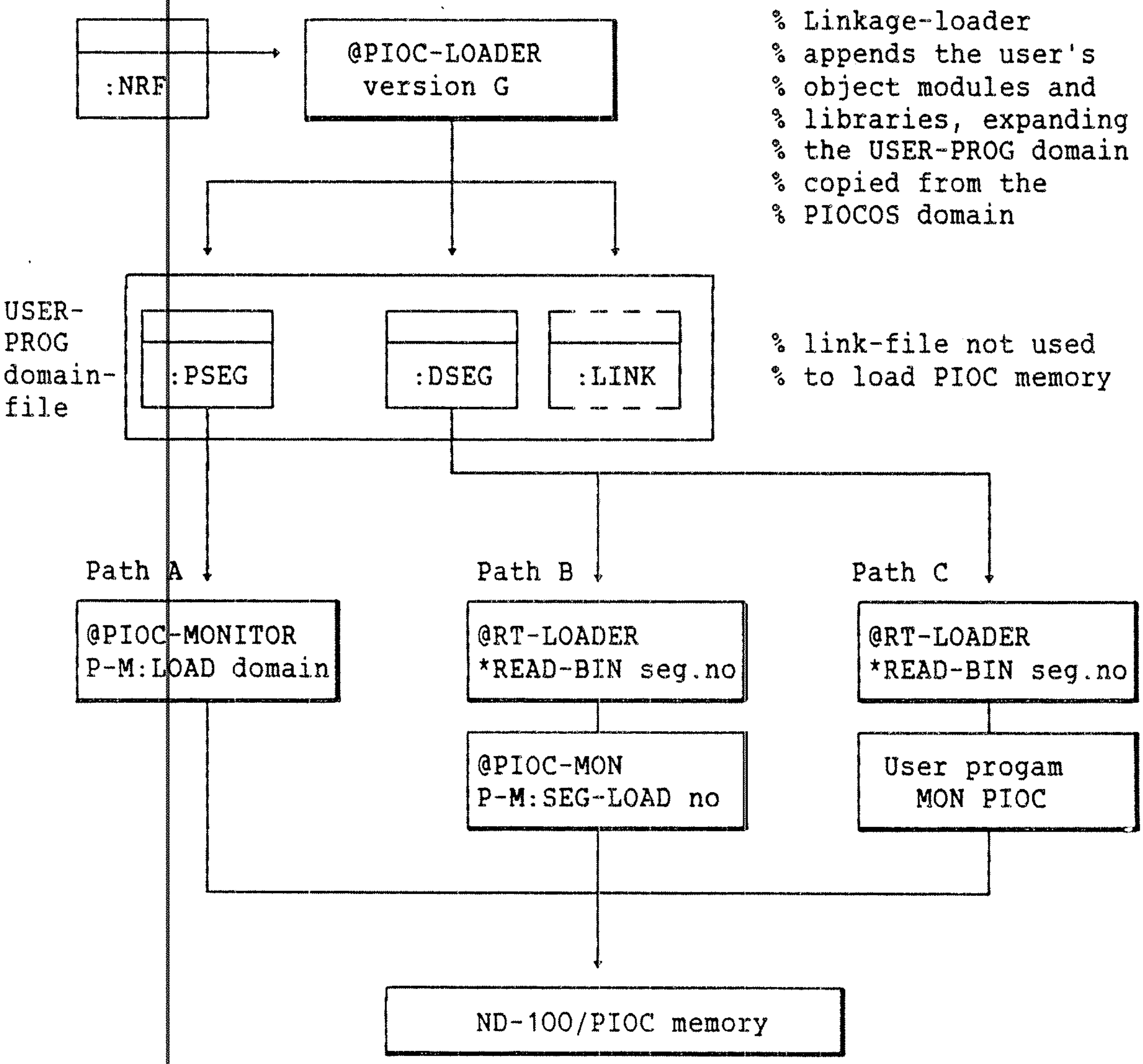


Figure 29. Several ways to load programs into PIOC memory.

If the PIOC has more physical memory than 128 Kbytes, the segments must cover the full memory area. A segment may be up to 64 pages (128 Kbytes). This is the maximum size that can be handled by SINTRAN III RT-LOADER.

Using the PIOC-MONITOR's SEGMENT-LOAD command, the additional segments will be requested until the PIOC memory is completely loaded.

The reason for this procedure is to prevent the SINTRAN III operating system from using unfixed memory pages for other ND-100 programs, as this may have undesired effects on the PIOC program.

7.5.1 Path A) Using the LOAD Command

This method assumes that you have created an empty segment of 128 Kbytes and that you fix it with the FIXC (fix contiguous) SINTRAN command. <1>

Remember to fix the segment beginning with the first physical PIOC page number (see the table on page 141). The reason for this is to prevent SINTRAN from using this area for swapping purposes.

You may then enter the PIOC-MONITOR and use the LOAD command reading the :PSEG file produced by the Linkage-Loader:

```
@PIOC-MONITOR-C

PIOC-Monitor - Release     : Cxx <month> <day> <year>

Give PIOC-number:   0  1 or EXIT : 0
PIOC started
PIOCOS - Release MARCH 22, 1985
The selected PIOCs address-range is 0B to    377777B

P-M:LOAD
Domain: USER-PROG
Segment: USER-PROG
Low-addr:
High-addr:
```

Example 17. Loading an application using the PIOC-MONITOR' LOAD command.

An empty segment can be defined in the RT-LOADER and fixed in memory, by using the following commands:

```
@RT-LOADER
REAL-TIME LOADER,  SINTRAN III  -  H

*NEW-SEGMENT,,,,,              (use default values)
NEW SEGMENT IS: 121            (segment number assigned)
*ALLOCATE-AREA,,177777,0       (defines area 128Kbytes)
*EXIT

@FIXC
SEGMENT: 121
PAGE: 200                      (see below)
```

Example 18. Defining an empty segment, and fixing it in memory

<1> Alternatively, use the FIX-SEGMENT command of the PIOC-MONITOR-C, in which case you do not need to know the physical page numbers.

The physical page number depends on the settings of the thumbwheel switches (7J and 9J) on the PIOC module.

Details on how to define the page numbers can be found in the PIOC Reference Manual, (ND-02.003), Chapter 2, and in the tables on page 141 of this manual.

7.5.2 Path B) Using the SEGMENT-LOAD Command

This method requires that you use the RT-LOADER for reading the :DSEG file into a free segment and force the segment to contain 64 pages. The loading session may look like this:

```
@RT-LOADER
REAL-TIME LOADER,  SINTRAN III  -  H

*NEW-SEGMENT,,,,,,                       (use default values)
NEW SEGMENT NO: 121
*READ-BINARY
INPUT-FILE: USER-PROG:DSEG
SEGMENT NO: 121
*SET-LOAD-ADDRESS
SEGMENT NO: 121           (force the segment to be 64 pages
LOAD ADDRESS: 177777      (You may also use several smaller)
                          (segments, e.g., 32+16+16 pages. )
*END-LOAD
*EXIT
```

Example 19. Using the RT-LOADER to create a PIOC memory segment.

If your system includes an RT-COMMON, you have to use the RT-LOADER's NEW-BACKGROUND-SEGMENT instead of the NEW-SEGMENT command.

You may then enter the PIOC-MONITOR and use the SEGMENT-LOAD command to bring this segment into the PIOC memory:

```
@PIOC-MONITOR-C

PIOC-Monitor - Release : Cxx - <month> <day> <year>

Give PIOC-number:   0  1 or EXIT : 0
PIOC started
PIOCOS - Release MARCH 22, 1985
The selected PIOCs address-range is 0B to    377777B

P-M:SEGMENT-LOAD
Segment (octal): 121
Page (octal): 0
```

Example 20. Loading an application from a ND-100 segment.

7.5.3 Path C) Using the MON PIOC Monitor Call in a ND-100 Program

This method also expects that you use the RT-LOADER, in the same way as explained in the previous section. But instead of using the monitor for loading, you must write an ND-100 real time program, calling the SEGLOAD function of the PIOC monitor calls (see page 71).

The program may look like this:

```
%        ***************************************************
%        *    RT-program to run in ND-100, loading the     *
%        *    PIOC-memory, using MON PIOC (255).           *
%        ***************************************************

MODULE LOADPIOC

% Import routine from the MON-PIOC-LIB library
% Remember to include the library when compiling this program

IMPORT (ROUTINE VOID,INTEGER&
    (INTEGER, INTEGER, INTEGER) : SEGFIX)

INTEGER ARRAY : STACK (0:100)
INTEGER : LDN :=1700B     % 1700 is LDN for PIOC no. 0
INTEGER : SEGNO :=121B
INTEGER : PAGE :=0
INTEGER : STATUS
INTEGER : DEVNO :=53D     % Messages to terminal no. 53
$INCLUDE PIOCOS-FUNCVAL-C:DEFS

PROGRAM : LOADPIOC

   INISTACK STACK

   SEGLOAD (LDN,SEGNO,PAGE) =: STATUS   %  LOAD
   IF STATUS >< U1OK THEN
      OUTPUT (DEVNO,'A','$*** AN ERROR HAS OCCURRED. ***')
      OUTPUT (DEVNO,'A','ERROR NUMBER: ')
      OUTPUT (DEVNO,'I6',STATUS)

      ELSE  OUTPUT (DEVNO,'A','$--- LOADING DONE ---')
   ENDIF

ENDROUTINE
ENDMODULE
$EOF
```

Example 21. Loading an application using MON PIOC from a PLANC program.

7.6 Object Modules Compiled with $SEPARATE-DATA OFF

If compiling with $SEPARATE-DATA OFF and importing PIOCOS objects (e.g., REALTIME), it may be necessary to define some symbols and addresses during loading of the object modules.

When the Linkage-Loader is in the computer mode PIOC, there is only one logical address area comprising both code and data. To "simulate" separation, the user must set load addresses explicitly before the loading begins:

```
        LOW-ADDRESS END_SYSTEM,P      % code area just behind PIOCOS
        LOW-ADDRESS 200000B, D        % data area on second bank
```

The addresses must be selected to match the real requirements, that is the code area chosen must be large enough to accomodate all of the user's object module(s) and run time libraries, without overlapping the data area.

So if compiling with $SEPARATE-DATA OFF, you have to define these symbols found on the "data" area of PIOCOS making them available as "program" references and vice versa:

```
DEFINE-ENTRY  REALTIME,     REALTIME, P
DEFINE-ENTRY  N100_CPU,     N100_CPU, P
DEFINE-ENTRY  PIOC_NUMBER,  PIOC_NUMBER, P
```

Figure 30. Defining global variables if using $SEPARATE-DATA OFF.

8 USING LIBRARIES - EXAMPLES

The ND-100 processes may communicate with PIOC processes through the monitor call PIOC (MON 255). This call offers several functions. By using the PIOC-N100LIB-C:BRF library supplied for MON PIOC it is fairly easy to add such calls to your program.

When writing PIOC programs you may also use library functions. Such a library is not delivered with the basic PIOC, but below you find a suggestion on how to write it.

```
MODULE PIOCOS_LIB

%    ***********************************************************
%    *        THIS LIBRARY IS USED FOR EXECUTING              *
%    *        CALLS FROM PIOC-PROGRAMS TO PIOCOS              *
%    ***********************************************************

EXPORT PLRES_SLOT
EXPORT PLREL_SLOT
EXPORT PLKICK
EXPORT PLFETCH
EXPORT PLWAITEV
%         .
%         .
%         .
%         .

$INCLUDE PIOCOS-FUNCVAL-C:DEFS

%    ***********************************************************
%    *      ROUTINE CALL_PIOCOS EXECUTE THE CALLS             *
%    ***********************************************************

ROUTINE VOID,VOID (FCODE,INTEGER POINTER,INTEGER WRITE):&
        CALL_PIOCOS (CALLNO,PARADDR,CSTATUS)

$*      MOVE.W      CALLNO,DO                        % DO:=CALLNO
$*      MOVEA.L     PARADDR,AO                       % AO:=PARADDR
$*      TRAP        #2                               % execute call
$*      MOVE.W      DO,CSTATUS                       % DO=:CSTATUS

        IF CSTATUS >< U10K THEN
           OUTPUT (1,'A','$ERROR : ')
           OUTPUT (1,'I3',CSTATUS)
        ENDIF

ENDROUTINE
```

```
%    ***************************************************
%    *                THE STANDARD ROUTINES            *
%    ***************************************************

%    ***    PLRES_SLOT    *******************************

ROUTINE VOID,VOID   (INTEGER, INTEGER4, INTEGER WRITE):&
        PLRES_SLOT  (SLOT,    EVENT,    ISTAT)

        CALL_PIOCOS (FRES_SLOT,ADDR SLOT,ISTAT)
ENDROUTINE

%    ***    PLREL_SLOT    *******************************

ROUTINE VOID,VOID   (INTEGER, INTEGER WRITE):&
        PLREL_SLOT  (SLOT,    ISTAT)

        CALL_PIOCOS (FREL_SLOT,ADDR_SLOT,ISTAT)
ENDROUTINE

%    ***    PLKICK        *********************************

ROUTINE VOID,VOID   (INTEGER, INTEGER,INTEGER WRITE):&
        PLKICK      (SLOT,    INFO,   ISTAT)

        CALL_PIOCOS (FKICK,ADDR_SLOT,ISTAT)
ENDROUTINE

%    ***    PLFETCH       *********************************

ROUTINE VOID,VOID   (INTEGER, INTEGER WRITE, INTEGER WRITE):&
        PLFETCH     (SLOT,    INFO,          ISTAT)

        CALL_PIOCOS (FFETCH,ADDR_SLOT,ISTAT)
ENDROUTINE

%    ***    PLWAITEV      *********************************

ROUTINE VOID,VOID   (INTEGER4 WRITE, INTEGER4, INTEGER WRITE):&
        PLWAITEV    (CURREV,         EVENT,    ISTAT)

        CALL_PIOCOS &
            (FWAITEV,(ADDR CURREV) CONVERT INTEGER POINTER,ISTAT)
ENDROUTINE

%              .
%              .
%              .
%              .


ENDMODULE
$EOF
```

Example 22. User library in PLANC.

On the next pages you find listings of two PLANC programs:

SPIOC: PIOC program which makes use of the library described earlier in this chapter (PIOCOS_LIB).

RPIOC: ND-100 real time program using the supplied library PIOC-N100LIB-C:BRF.

First prepare RPIOC as a usual ND-100 RT-program and SPIOC as a PIOC application. Remember to load the necessary libraries also.

Then enter the PIOC-MONITOR-C and load the process SPIOC by LOAD-DOMAIN or SEGMENT-LOAD. Start the program on PIOC and it will come to an INPUT-statement.

```
    P-M:LOAD SPIOC,SPIOC,,,, CR
```

Go to another terminal, log in as SYSTEM or RT and start the real time program RPIOC.

```
    @RT RPIOC CR
    @LOG CR          (You must log out, since the program
                     reserves the terminal you started it from.)
```

The program will write out something on this terminal if you have modified the initialization of the DEVNO variable in RPIOC according to your terminal number.

Go back to the PIOC-MONITOR-C, type any number and <CR>. SPIOC will now kick RPIOC and both will write out a protocol of this action on their terminals.

```
    P-M:START-PIOC,, CR
```

If SPIOC had been loaded and started from RPIOC (with the monitor call PIOC), SPIOC could not have done simple output to terminal. This is the reason why this example shows loading and starting of SPIOC from the PIOC-MONITOR.

The two programs then start to synchronize each others activities, and this is what happens:

Program running in the PIOC		Program running in the ND-100
SPIOC waits for input from the user	↓	
		RPIOC reserves a terminal.
	t	RPIOC reserves a slot.
	i	RPIOC enters 'RT-WAIT'.
SPIOC reserves a slot.	m	
SPIOC 'kicks' RPIOC.	e	
SPIOC enters SUSPENDED state.		RPIOC leaves 'RT-WAIT'.
waiting for 'kick'.		RPIOC fetches info from SPIOC.
		RPIOC 'kicks' SPIOC.
SPIOC enters ACTIVE state.		RPIOC releases its slot.
SPIOC releases its slot.		RPIOC releases the terminal.
SPIOC stops.	↓	RPIOC stops.

Figure 31. Programs synchronizing their activity using kicks.

```
%        **************************************************
%        *    SPIOC                                       *
%        *    Program to run in PIOC. It sends a kick to  *
%        *    ND-100, transferring a number (7) via the   *
%        *    mailbox to ND-100 ( RPIOC ).                *
%        *                                                *
%        **************************************************

MODULE SPIOC                                 % global declarations
EXPORT AUTO_START
IMPORT (ROUTINE VOID,VOID   &
       (INTEGER,INTEGER4,INTEGER WRITE) : PLRES_SLOT)
IMPORT (ROUTINE VOID,VOID   &
       (INTEGER,INTEGER WRITE) : PLREL_SLOT)
IMPORT (ROUTINE VOID,VOID   &
       (INTEGER,INTEGER,INTEGER WRITE) : PLKICK)
IMPORT (ROUTINE VOID,VOID   &
       (INTEGER4 WRITE,INTEGER4,INTEGER WRITE) : PLWAITEV)
INTEGER ARRAY : STACK(0:500)
INTEGER : STATUS
%        ==================================================
%             Main program : AUTO_START
%        ==================================================
PROGRAM  : AUTO_START
INTEGER4 : EVENT,CURREV
INTEGER  : SLOT,INFO, HELP

       INISTACK   STACK
       OUTPUT (1,'A','$ > THIS IS SPIOC RUNNING KICK-TEST TO RPIOC <')
       3 =: SLOT
       2 =: EVENT
       INPUT(1,'I',HELP)                          % Wait for input
       PLRES_SLOT (SLOT,EVENT,STATUS)            % Reserve slot 3
       OUTPUT (1,'A','$ SLOT 3 IS RESERVED ')
       OUTPUT (1,'A','$ I KICK RPIOC ')

       7=: INFO
       PLKICK (SLOT,INFO,STATUS)                 % Kick RPIOC, send 7

       OUTPUT (1,'A','$ I AM WAITING FOR EVENT')
       PLWAITEV (CURREV,EVENT,STATUS)            % Wait for event, go
                                                 % suspended
%      Started up on kick from ND-100
       OUTPUT (1,'A','$ I AM BACK TO LIFE')
       PLREL_SLOT (SLOT,STATUS)
       OUTPUT (1,'A','$ > SPIOC PROGRAM STOPS NOW < $')
ENDROUTINE
ENDMODULE
$EOF
```

Example 23. Program SPIOC running in PIOC (PLANC).

```
%         ***************************************************
%         *   RPIOC                                         *
%         *   RT-program to run on ND-100, receiving info   *
%         *   from PIOC by using 'kicks'.                   *
%         *                                                 *
%         ***************************************************

MODULE RPIOC
IMPORT (ROUTINE VOID,INTEGER (INTEGER, INTEGER, INTEGER) : MN122)
IMPORT (ROUTINE VOID,VOID (INTEGER,INTEGER) : MN123)

IMPORT (ROUTINE VOID,INTEGER (INTEGER, INTEGER ) : RES_SLOT)
IMPORT (ROUTINE VOID,INTEGER (INTEGER, INTEGER ) : REL_SLOT)
IMPORT (ROUTINE VOID,INTEGER (INTEGER, INTEGER ,INTEGER ) : KICK)
IMPORT (ROUTINE VOID,INTEGER (INTEGER, INTEGER ,INTEGER WRITE):FETCH)

INTEGER ARRAY   : STACK (0:100)
INTEGER : LDN    :=1700B                      % LDN to PIOC no 0
INTEGER : SLOT   :=3                          % Slot no 3
INTEGER : DEVNO  :=37D                        % Output to term no 37
INTEGER : IOFLAG:=1                           % Reserve the term outp.
INTEGER : IRET   :=0                          %
INTEGER : U1OK   :=1                          % Return status 1 if OK
INTEGER : STATUS, INFO
%        ===================================================
%            Main program : RPIOC
%        ===================================================
PROGRAM : RPIOC
       INISTACK STACK

       MN122 (DEVNO,IOFLAG,IRET)              %  RESERVE TERMINAL
       OUTPUT (DEVNO,'A','$ > RPIOC RUNNING < ')

       REL_SLOT (LDN,SLOT) =: STATUS          %  RELEASE SLOT
       IF STATUS >< U1OK THEN
             OUTPUT (DEVNO,'A','$ **** ERROR IN ROUTINE: REL_SLOT')
             OUTPUT (DEVNO,'I6',STATUS)
       ENDIF

       RES_SLOT (LDN,SLOT) =: STATUS          %  RESERVE SLOT
       IF STATUS >< U1OK THEN
             OUTPUT (DEVNO,'A','$ **** ERROR IN ROUTINE: RES_SLOT')
             OUTPUT (DEVNO,'I6',STATUS)
        ELSE OUTPUT (DEVNO,'A','$ SLOT RESERVED')
             OUTPUT (DEVNO,'A','$ RPIOC WAITING FOR KICK')
       ENDIF

$*     MON 135                                %  CALL RT-WAIT
```

```
          OUTPUT (DEVNO,'A','$ RPIOC RECEIVED KICK')
          FETCH (LDN,SLOT,INFO) =: STATUS          %  GET INFO FROM SLOT
          IF STATUS >< U1OK THEN
                OUTPUT (DEVNO,'A','$ **** ERROR IN ROUTINE: FETCH')
                OUTPUT (DEVNO,'I6',STATUS)
           ELSE OUTPUT (DEVNO,'A','$ RECEIVED INFO = ')
                OUTPUT (DEVNO,'I6',INFO)
          ENDIF

          0=:INFO
          KICK (LDN,SLOT,INFO) =: STATUS           %  KICK TO SLOT
          IF STATUS >< U1OK THEN
                OUTPUT (DEVNO,'A','$ **** ERROR IN ROUTINE: KICK')
                OUTPUT (DEVNO,'I6',STATUS)
          ELSE  OUTPUT (DEVNO,'A','$ I HAVE SENT A KICK')
          ENDIF

          REL_SLOT (LDN,SLOT) =: STATUS            %  RELEASE SLOT
          IF STATUS >< U1OK THEN
                OUTPUT (DEVNO,'A','$ **** ERROR IN ROUTINE: REL_SLOT')
                OUTPUT (DEVNO,'I6',STATUS)
          ELSE  OUTPUT (DEVNO,'A','$ SLOT IS RELEASED')
          ENDIF

          OUTPUT (DEVNO,'A','$ RPIOC PROGRAM STOPS NOW')
          MN123 (DEVNO,IOFLAG)                     %  RELEASE TERMINAL

ENDROUTINE
ENDMODULE
$EOF
```

Example 24. Program RPIOC running in ND-100 (PLANC).

APPENDIX A

Symbolic Names for PIOCOS System Calls and Status Codes

This appendix lists the PIOC-FUNCVAL-COO:DEFS file supplied with the PIOC Basic Software. If you are using a newer version, the file may have been expanded or changed.

```
%*****************************************************************
%
%   PIOC-FUNCVAL-COO:DEFS
%   ---------------------
%   definitions of the symbolic names for calls to PIOCOS
%   and return codes from PIOCOS.
%*****************************************************************
%
%   PIOCOS system calls
%   -------------------
TYPE FCODE = ENUMERATION(FMONO,FMON1,FMON2,FCREATE,FBEGIN,FKILL, &
       FABORT,FPROSNO,FPRNAME,FSETEV,FWAITEV,FSELWAITEV,FREADEV, &
       FINTEREV,FINTERDEL,FCRDRV,FTRAPH,FWHOAMI, FRES_SLOT,      &
       FREL_SLOT,FKICK,FFETCH, FXMSG,FPHLS,FRTKICK,FNXMSG,FEND,  &
       FSYN_RET,FXDRV)
%
%   system call return values (set in DO by PIOCOS)
%   -----------------------------------------------
CONSTANT U1OK                =   1   % operation successfully completed
CONSTANT UNOTCOMPL           =   0   % operation not completed
CONSTANT UNOEXIST            =  -1   % process does not exist
CONSTANT U1NOSP              =  -2   % no space in buffer pool
CONSTANT U1EXIST             =  -3   % process already exists
CONSTANT U1ILPRI             =  -4   % illegal priority
CONSTANT UQFULL              =  -5   % timer queue full
CONSTANT UEVNOEX             =  -6   % event not found
CONSTANT UILVEC              = -10   % illegal vector address
CONSTANT UILCAL              = -11   % call not implemented

%   error codes used in KICK monitor call
CONSTANT UPILF               = -20   % illegal function
CONSTANT UPSLBS              = -21   % slot occupied
CONSTANT UPILSL              = -22   % illegal slot (not existing)
CONSTANT UPNOTY              = -23   % slot not reserved by you
CONSTANT UPFULL              = -24   % box not empty
CONSTANT UPNOME              = -25   % box empty

%   error codes used in PHLS
CONSTANT UILL_SERVICE        = -31   % illegal service requested
CONSTANT UIL_PHLS            = -32   % illegal PHLS number
CONSTANT UIL_SERVICE_POINT   = -33   % no such service point
CONSTANT UIL_PHLS_TYPE       = -34   % illegal PHLS type (HDLC,ASYNC)

%   utilities
CONSTANT YYNOPROS            = -50   % no process with this name
CONSTANT YYNOFREE            = -51   % no free entry for new process
```

Table 7. PIOCOS FUNCTION VALUES, Part 1.

```
%   memory manager

CONSTANT MM_OK              =    0  % successful completion
CONSTANT MM_NOBUF           = -100  % no vacant buffer
CONSTANT MM_INCONSISTENT    = -101  % inconsistency
CONSTANT MM_ILADDR          = -102  % illegal buffer address
%
%   description of the system call parameters for PHLS
%   --------------------------------------------------------

TYPE SERP      = ENUMERATION (FCONTROL,FRECEIVER,FTRANSMITTER)
TYPE PHLS_TYPE = ENUMERATION (HDLC,ASYNCHRONOUS)

%   DCB-NAME values
%   ---------------

CONSTANT TRAN_REQUEST       =  1    % transmit DCB request
CONSTANT TRAN_RESPONSE      = -1    % transmit DCB response
CONSTANT RECV_REQUEST       =  2    % receive  DCB request
CONSTANT RECV_RESPONSE      = -2    % receive  DCB response
CONSTANT CONN_REQUEST       =  3    % connect      request
CONSTANT CONN_RESPONSE      = -3    % connect      response
CONSTANT DIS_REQUEST        =  4    % disconnect   request
CONSTANT DIS_RESPONSE       = -4    % disconnect   response
CONSTANT INIT_REQUEST       =  5    % init         request
CONSTANT INIT_RESPONSE      = -5    % init         response

%   HDLC         status values
%   --------------------------

CONSTANT E_OK               = 0     % okay
CONSTANT E_UNDERRUN         = 1     % underrun
CONSTANT E_ILL_NAME         = 2     % illegal message name
CONSTANT E_ABORT            = 3     % frame aborted
CONSTANT E_OVERRUN_CRC      = 4     % overrun or crc error
CONSTANT E_WP_VIOLATION     = 5     % write protect violation
CONSTANT E_SMAL_BUFF        = 6     % overflow in receive message
CONSTANT E_ILL_ADDR         = 7     % illegal address for data message
CONSTANT E_NOT_CONNECTED    = 8     % not connected

%   ASYNCHRONOUS status values
%   --------------------------

CONSTANT A_OK               = 0     % okay
CONSTANT A_EFRAM_ERROR      = 1     % framing error
CONSTANT A_EILL_NAME        = 2     % illegal message name
CONSTANT A_EPARITY          = 3     % parity error
CONSTANT A_EOVERRUN         = 4     % overrunn
CONSTANT A_ENOT_CONNECTED   = 8     % not connected
$EOF
```

Table 8. PIOCOS FUNCTION VALUES, Part 2.

APPENDIX B

Symbolic Names for Functions and Error Codes in XMSG

This appendix contains the PIOC-XFUNCVAL-COO:DEFS file supplied with the PIOC BASIC Software. If you are using a newer version, the file may have been expanded or changed.

```
%*****************************************************************
%
%   PIOC-XFUNVAL-COO:DEFS
%   ---------------------
%   Defines the symbols for functions and error-codes
%
%*****************************************************************

%   function values
%   ---------------

CONSTANT XFDUM =   0B   % dummy function
CONSTANT XFDCT =   1B   % disconnect from message system
CONSTANT XFGET =   2B   % get message space
CONSTANT XFREL =   3B   % release message space
CONSTANT XFRHD =   4B   % release message space (6 bytes)
CONSTANT XFWHD =   5B   % write header to a message (6 bytes)
CONSTANT XFREA =   6B   % read from message to user buffer
CONSTANT XFWRI =   7B   % write from user to message
CONSTANT XFSCM =  10B   % set current message
CONSTANT XFMST =  11B   % get message status
CONSTANT XFOPN =  12B   % open port
CONSTANT XFCLS =  13B   % close port
CONSTANT XFSND =  14B   % send message to a remote port
CONSTANT XFRCV =  15B   % receive a message on a given port
CONSTANT XFPST =  16B   % get local port status
CONSTANT XFGST =  17B   % general status or wait

%   service functions

CONSTANT XFSIN =  20B   % service initialization function
CONSTANT XFSRL =  21B   % service release function
CONSTANT XFABR =  22B   % absolute read block from POF area
CONSTANT XFABW =  23B   % absolute write block to POF area
CONSTANT XFMLK =  24B   % lock message system
CONSTANT XFMUL =  25B   % unlock message system
CONSTANT XFM2P =  26B   % magic number to port id.
CONSTANT XFP2M =  27B   % port to magic number
CONSTANT XFRIN =  30B   % routing initialize (called by XROUT)
CONSTANT XFCRD =  31B   % create driver with context
CONSTANT XFSTD =  32B   % start driver
```

Table 9. XMSG-FUNCTION-VALUES, Part 1.

```
%   indirect buffer handling functions

CONSTANT XFDIB =  33B   % define indirect buffer
CONSTANT XFRIB =  34B   % read from indirect buffer
CONSTANT XFWIB =  35B   % write to indirect buffer
CONSTANT XFDUB =  36B   % define user buffer for current message
CONSTANT XFMX1 =  37B   % end marker ** leave me here please
%   bit values in function codes
%   ---------------------------

CONSTANT XFWTF =  17B   % set then wait if operation not terminated
CONSTANT XFWAK =  16B   % RCV/PST/GST: do RTENTRY on status change
CONSTANT XFPRM =  15B   % XFOPN: permanent open required
CONSTANT XFWOK =  15B   % XFDIB: allow write access to indirect buffer
CONSTANT XFHIP =  15B   % XFSND: high-priority message
CONSTANT XFBNC =  14B   % XFSND: bounce message
CONSTANT XFFWD =  13B   % XFSND: forward message
CONSTANT XFROU =  12B   % XFSND: message to be sent to routing process
CONSTANT XFSEC =  11B   % XFSND: secure msg (return if not delivered)

%   message types: returned as successful status from XFRCV
%   -------------------------------------------------------

CONSTANT XMTNO =  1B    % normal message
CONSTANT XMROU =  2B    % routed message (via XROUT)
CONSTANT XMTHI =  3B    % high priority message
CONSTANT XMTRE =  4B    % return message (abnormal condition)
CONSTANT XMTPS =  5B    % pseudo message
```

Table 10. XMSG-FUNCTION-VALUES, Part 2.

```
%   error symbols used by XMSG
%   --------------------------
%   implies that the error is probably internal to the XMSG system
%   and so leads to a call to ZCRAS, and the error is not returned
%   to the user.

CONSTANT XEOK  =   1    % operation successfully completed

CONSTANT XENOT =  -1B   % No more XT-blocks free
CONSTANT XEIRM =  -2B   % Non-local remote port illegal here
CONSTANT XETMM =  -4B   % Task is not allowed any more memory
CONSTANT XENIM =  -5B   % Facility not yet implemented
CONSTANT XEIBP =  -6B
CONSTANT XEBNY =  -7B   % Message buffer not yours
CONSTANT XEISP = -10B   % Illegal service program calling
CONSTANT XENOP = -11B   % No more ports available
CONSTANT XEIDR = -12B   % Function not available to drivers
CONSTANT XENDM = -13B   % No default message
CONSTANT, XEMCH = -14B  % Message is already chained
CONSTANT XEBFC = -15B   % Message is in a queue
CONSTANT XERAL = -16B   % Routing port already defined
CONSTANT XECRA = -17B   % XMSG crash (Info in Basefield)
CONSTANT XEWNA = -20B   % Write Not Allowed (Indirect buffer)
CONSTANT XENVI = -21B   % No Valid Indirect buffer defined
CONSTANT XEILF = -22B   % Illegal function code in monitor call
CONSTANT XEIMA = -23B   % Invalid magic number
CONSTANT XEMFL = -24B   % Message space full
CONSTANT XEILM = -25B   % Illegal message size
CONSTANT XEIPN = -26B   % Illegal port number
CONSTANT XEXBF = -33B   % Message already has a buffer

CONSTANT XXEIE =   1B   % * illegal (COMTAB) entry ptr to XCRMG
CONSTANT XXIOW =   2B   % * illegal owner of buffer
CONSTANT XXBIN =   3B   % * memory allocn. inconsistency
CONSTANT XXMCE =   4B   % * message queue length inconsistency
CONSTANT XXIEN =   5B   % * ZRALL gave port not in XQTAB
CONSTANT XXIFL =   6B   % * INIT: ZFUNC function >XFMX1
CONSTANT XXIRT =   7B   % * illegal rt-description add used.
CONSTANT XXNBF =  10B   % * INIT: no buffer space available
CONSTANT XXRIN =  11B   % * inconsistency in resource allocation
CONSTANT XXNMM =  12B   % * more memory released than owned
CONSTANT XXNIM =  13B   % * not implemented (cannot recover)
CONSTANT XXCLS =  14B   % * inconsistency in port chain in close
CONSTANT XXCHE =  15B   % * double chaining attempted
CONSTANT XXNOR =  16B   % * no XMSG-resident found by MFINI (POF)
CONSTANT XXICM =  17B   % * inconsistency in XMPRT/XPCMS pair
```

Table 11. XMSG ERROR SYMBOLS, Part 1.

```
%   XROUT service values
%   --------------------

CONSTANT XSNUL =  1     % null command returns 0 status to sender
CONSTANT XSLET =  2     % send a letter
CONSTANT XSNAM =  3     % give name to this port
CONSTANT XSCNM =  4     % clear name of this port
CONSTANT XSGNM =  5     % get name of port (param: magno)
CONSTANT XSGNI =  6     % get name (param: mc/portno)
CONSTANT XSMAX =  XSGNI % maximum legal service value

%   XROUT errors
%   ------------
%   the following are error values returned as results from a
%   service request in byte 0 of the message.

CONSTANT XRISN =   1    % illegal service number
CONSTANT XRUNN =   2    % no open port has this name
CONSTANT XRDDF =   3    % another port already has this name
CONSTANT XRNSP =   4    % no space left for names
CONSTANT XRIPT =   5    % illegal parameter type
CONSTANT XRMMP =   6    % missing mandatory parameter
CONSTANT XRUNM =   7    % unknown magic number
CONSTANT XRMTL =  10    % resulting message too long
$EOF
```

Table 12. XMSG ERROR SYMBOLS, Part 2.

APPENDIX C

PIOC EXCEPTION VECTOR ASSIGNMENTS

Vectors not listed are not used - they use a dummy exception handler - or their use will be changed in the future.

No.	Used for		Handler
0	Reset (Initial supervisor stack-pointer)		
1	Reset (Initial program-counter)		
2	Bus-error		TBUS
3	Address-error (word-access to odd address)		TADDR
4	Illegal instruction		TILIN
5	Divide by zero		TZERO
6	Trap according to CHECK-instruction		TCHECK
7	Trap according to TRAPV-instruction		TTRAPV
8	Privilege violation		TPRIV
9	Tracing-mode		TTRACE
10	1010-Emulator		T1010
11	1111-Emulator		T1111
24	Spurious interrupt		T24
26	Serial output		T26
28	Serial input		T28
29	Memory error		T29
30	OPCOM interrupt from ND-100		TLEVEL_6
31	Power-fail		TLEVEL_7
34	TRAP #2, MONITOR-call to PIOCOS		PIOCTRAP
35	Special breakpoint-handling		BREAK
42	TRAP #10, reserved for breakpoints		TRAP_10
64	write-protect-violation		INCT_30
68	ND-100 calling		INCT_34
69	RT-clock (dynamically assigned during startup)		RTCDRIV
70	Output-DMA-error		INCT_36
71	Input-DMA-error		INCT_37
81	Line 1	HDLC	SI010B_INT
82	Line 1	HDLC,Async	SI001B_INT
83	Line 1	HDLC	SI011B_INT
84	Line 1	HDLC,Async	SI002B_INT
86	Line 1	HDLC,Async	SI003B_INT
89	Line 0	HDLC	SI010A_INT
90	Line 0	HDLC,Async	SI001A_INT
91	Line 0	HDLC	SI011A_INT
92	Line 0	HDLC,Async	SI002A_INT
94	Line 0	HDLC,Async	SI003A_INT
97	Line 3	HDLC	SI030B_INT
98	Line 3	HDLC,Async	SI021B_INT
99	Line 3	HDLC	SI031B_INT
100	Line 3	HDLC,Async	SI022B_INT
102	Line 3	HDLC,Async	SI023B_INT
105	Line 2	HDLC	SI030A_INT
106	Line 2	HDLC,Async	SI021A_INT
107	Line 2	HDLC	SI031A_INT
108	Line 2	HDLC,Async	SI022A_INT
110	Line 2	HDLC,Async	SI023A_INT

Table 13. PIOC EXCEPTION VECTOR ASSIGNMENTS

APPENDIX D

PIOC Physical Memory Page Numbers

The thumbwheel switches 7J and 9J on the PIOC-module are used to select the memory area shared between the PIOC and the ND-100.

On the PIOC/128Kb:

7J	9J	ND-100 pages:
0	0	0 - 77 *
.	1	100 - 177
.	2	200 - 277
.	3	300 - 377
.	.	
0	15	1700 - 1777
1	0	2000 - 2077
.	1	2100 - 2177
.	2	2200 - 2277

* The first 77 octal pages are reserved by SINTRAN III opr.sys.

Table 14. PIOC/128KB Physical Page Numbers.

On the PIOC/512Kb:
9J ODD number = small window, 128Kbytes:

7J	9J	ND-100 pages:
0	1	0 - 177 *
.	3	200 - 377
.	5	400 - 577
.	.	
0	15	1600 - 1777
1	1	2000 - 2177
.	3	2200 - 2377

* The first 77 octal pages are reserved by SINTRAN III opr.sys.

Table 15. PIOC/512KB Physical Page Numbers, Small Windows.

On the PIOC/512Kb:
9J EVEN number = large window, 256Kbytes:

7J	9J	ND-100 pages:
0	0	0 - 377 *
(0	2	0 - 377)
0	4	400 - 777
(0	6	400 - 777)
.	.	
0	14	3000 - 3377
1	0	3400 - 3777
.	2	4000 - 4377

* The first 77 octal pages are reserved by SINTRAN III opr.sys.

Table 16. PIOC/512KB Physical Page Numbers, Large Windows.

The lower two bits of the thumbwheel 9J are ignored with the standard use of PIOC.

Index

*************** SEND US YOUR COMMENTS!!! ***************

Are you frustrated because of unclear information in this manual? Do you have trouble finding things? Why don't you join the Reader's Club and send us a note? You will receive a membership card — and an answer to your comments.

Please let us know if you
- find errors
- cannot understand information
- cannot find information
- find needless information

Do you think we could improve the manual by rearranging the contents? You could also tell us if you like the manual!

************* HELP YOURSELF BY HELPING US!! *************

Manual name: PIOC Software Guide

Manual number: ND-60.161.3 EN

What problems do you have? (use extra pages if needed) ____________________

____

____

____

____

Do you have suggestions for improving this manual ? ____________________

____

____

____

____

____

Your name: ____________________ Date: __________

Company: ____________________ Position: __________

Address: ____________________

____

What are you using this manual for ? ____________________

____

NOTE!

This form is primarily for documentation errors. Software and system errors should be reported on Customer System Reports.

Send to:

Norsk Data A.S
Documentation Department
P.O. Box 25, Bogerud
0621 Oslo 6, Norway

Norsk Data's answer will be found on reverse side →

Answer from Norsk Data: ______________________________

Answered by: ______________________ Date: ______________

Norsk Data A.S
Documentation Department
P.O. Box 25 BOGERUD
N - 0621 OSLO 6 - Norway

ND