

NAKED MINI® 4

PASCAL OPERATIONS MANUAL

93171-90

NOVEMBER 1981



NAKED MINI® Division

**Engineering Reference Number
90-93171-00A0**

**HERTFORD HOUSE
DENHAM WAY, MAPLE CROSS
RICKMANSWORTH, HERTFORDSHIRE WD3 2XD**

Telephone Rickmansworth 71211

Telex 922654

Printed in U.S.A.

REVISION HISTORY

<u>Revision</u>	<u>Issue Date</u>	<u>Comments</u>
A0	Nov. 1981	Original issue.

Computer Automation, Inc., reserves the right to make changes in specifications at any time and without notice. The information furnished by Computer Automation, Inc., in this publication is believed to be accurate and reliable. However, no responsibility is assumed by Computer Automation, Inc., for its use; nor for any infringements of patents or other rights of third parties resulting from its use. No license is granted under any patents or patent rights of Computer Automation, Inc.

Computer Automation, Inc., welcomes comments and suggestions for improvements to our manuals. Should difficulty be encountered with the information in the manual or if you have a suggestion for manual improvement, please send the comments or suggestions to Computer Automation, Inc., Technical Publications, Department 11-0770, 18651 Von Karman, Irvine, California, 92713. Please provide as much information as possible to help evaluate the comment or suggestion.

PREFACE

The purpose of this manual is to aid the PASCAL programmer in compiling, linking, loading, and executing programs on the NAKED MINI® 4 series of computers.

This manual is organized to follow the normal sequence of operations in developing and executing a PASCAL program. The compiler operation is described first, followed by library structure and linking procedures, and finally runtime procedures. The generation of compatible assembly language functions and use of PASCAL system files are then described.

The document refers frequently to procedures performed in the OS4 system and in the RTX4/IOS4 system. The reader should therefore refer to the following documents for further information on these subjects.

	<u>Order Number</u>
• Cherry, <u>Pascal Programming Structures</u>	93170-90
• <u>OS4 System Reference Manual</u>	93460-90

CONTENTS

SECTION 1	INTRODUCTION	1-1
1.1	PASCAL DEVELOPMENT CYCLE	1-1
1.2	OPERATING ENVIRONMENT	1-1
SECTION 2	NM4 PASCAL COMPILER	2-1
2.1	NM4 PASCAL COMPILER OPERATION	2-1
2.2	INVOKING THE COMPILER	2-1
2.2.1	PASCAL Command Line	2-1
2.2.2	PASCAL Command Options	2-3
2.2.3	Command Line Examples	2-5
2.3	NM4 PASCAL DIRECTIVES	2-6
2.4	COMPILER ERROR REPORTS	2-7
SECTION 3	POSTCOMPILATION PROCESSES	3-1
3.1	ASSEMBLY	3-1
3.2	LINKING	3-1
3.3	LINKING FOR EXECUTION UNDER OS4	3-1
3.4	LINKING FOR EXECUTION UNDER RTX4	3-2
SECTION 4	RUNTIME	4-1
4.1	LOADING AND EXECUTION UNDER OS4	4-1
4.2	LOADING AND EXECUTION UNDER RTX4	4-2
SECTION 5	INTERFACING TO ASSEMBLY LANGUAGE	5-1
5.1	ASSEMBLY LANGUAGE FUNCTIONS CALLED FROM PASCAL	5-1
5.1.1	Entering and Returning from Functions	5-1
5.1.2	Parameters of Functions	5-2
5.1.3	Returning Results	5-2
5.2	PASCAL FUNCTIONS CALLED FROM ASSEMBLY LANGUAGE	5-2
SECTION 6	REAL-TIME SERVICES	6-1
6.1	TASK SERVICES	6-1
6.1.1	Task Control	6-1
6.1.2	Semaphore	6-3
6.1.3	Mailbox	6-4
6.1.4	Clock	6-4
6.1.5	Date and Time	6-5
6.1.6	Console Interrupt	6-7
SECTION 7	SYSTEM FILES	7-1

APPENDIX

A	GENERAL ROUTINES	A-1
---	----------------------------	-----

FIGURES

<u>Figure</u>		<u>Page</u>
2-1	Sample Listing of PASCAL Program DEMO.PSC	2-2
2-2	Output Resulting from DEMO.PSC	2-8
2-3	Sample Listing of PASCAL Program with Error Printout ERROR.PSC	2-8
6-1	PASCAL Functions for RTX4 Services	6-2
A-1	General RTX4 Interface Routines.	A-1

TABLES

<u>Table</u>		
2-1	NM4 PASCAL Command Options	2-4
2-2	NM4 PASCAL Directives	2-7

SECTION 1

INTRODUCTION

The design, development, and compilation of NM4 PASCAL application software takes place under the OS4 operating system. Following the compilation and checkout of an application, it can be run in a production mode under either OS4 or RTX4.

1.1 PASCAL DEVELOPMENT CYCLE

Developing a PASCAL program is a series of basic steps:

1. The programmer designs and codes the NM4 PASCAL source program and transfers the code to a suitable input medium, usually a disk storage device.
2. The NM4 PASCAL compiler translates the source program into assembly language, producing a listing.
3. The NAKED MINI® 4 macro assembler takes the assembly code produced by the NM4 PASCAL compiler and produces object code as output.
4. The OS4 LINK utility links the object code output by the macro assembler with the needed library modules and any other required files.
5. The linked PASCAL program can then be loaded by one of the following methods:
 - By an OS4 command if execution is to take place under OS4
 - By the OS4 /AUTO command or hardware AutoLoad function if the program has been linked as an RTX application.

1.2 OPERATING ENVIRONMENT

The following hardware represents the minimum configuration required to compile, link, and execute a PASCAL program:

- NAKED MINI® 4 Family processor with 64K words of read/write memory
- Dual quad floppy drives for loading the NM4 PASCAL and OS4 product diskettes
- A diskette labelled using the OS4 LABEL utility
- I/O distributor capacity as required by input/output (I/O) channels used
- A suitable console terminal.

This minimum configuration can be augmented for improved performance. The system assumed for examples in this manual includes these items plus a CRT and a line printer. All character devices included in the user's configuration should be able to output the lowercase character set or transpose lowercase to uppercase. The NM4 PASCAL compiler outputs compiler and runtime error messages in lowercase.

The software product diskettes required to compile and link an NM4 PASCAL program are:

	<u>Diskette ID</u>	<u>Model Number</u>
OS4 Utilities Diskette # 1	F46001	96460-01
NM4 PASCAL Diskette	F17001	93170-01
OS4 LINK Diskette	F46004	93460-04

SECTION 2

NM4 PASCAL COMPILER

After an NM4 PASCAL program (Figures 2-1 and 2-2) has been created or modified with Edit and stored on a suitable medium, it is converted to assembly language by the NM4 PASCAL compiler.

2.1 NM4 PASCAL COMPILER OPERATION

The operations performed by the compiler are:

1. Inputting the PASCAL source program from the input medium
2. Producing assembly language source output on the specified output medium.

Before a compiled program can be executed, it must be macroassembled, linked to required library routines and converted into loadable format by the LINK Editor, and finally loaded for execution.

2.2 INVOKING THE COMPILER

NM4 PASCAL is invoked by entering the command PASCAL to the OS4 operating system from the command input (CI) device. The compiler is then loaded and executed in response to PASCAL command lines.

2.2.1 PASCAL Command Line

The PASCAL command instructs OS4 to load and execute the compiler. The programmer makes three specifications about the compilation to be performed in the PASCAL command:

1. Identification of the source medium or media, i.e., the file containing the PASCAL text to be compiled
2. Specification of any compiler options to be invoked
3. Definition of the destination media, i.e., the file or device to which the assembler source output code is to be written and the file or device to which the listing is to be written.

The programmer has considerable flexibility in how he supplies this information. Various ways of issuing the PASCAL command are described on the following pages, and the following rules apply to all forms of the PASCAL command, except where stated otherwise:

```
PASCALDEMO.PSC 1981 Jul 21 08:53:37.50
1  program listprimes(output) ;
2  {this program calculates and prints all prime
3  numbers from 1 to 100.}
4
5  type positiveinteger = 1..maxint ;
6
7  var i : positiveinteger ;
8
9  function prime (p : positiveinteger) : boolean ;
10 {prime determines whether p is prime or not}
11   var root, divisor : positiveinteger ;
12   begin
13     if p<4 then prime := p>1 else
14     if not odd(p) then prime := false else
15     begin
16       root := round (sqrt(p)) ;
17       divisor := 3 ;
18       while (divisor<=root) and (p mod divisor <> 0)
19       do divisor := divisor + 2 ;
20       prime := divisor>root
21     end
22   end ; {prime}
23
24   begin
25     for i := 1 to 100 do
26       if prime(i)
27       then writeln (i, ' is a prime number') ;
28   end.
( 0) 0 fatals 0 warnings
```

Figure 2-1. Sample Listing of PASCAL Program DEMO.PSC

```
2 is a prime number
3 is a prime number
5 is a prime number
7 is a prime number
11 is a prime number
13 is a prime number
17 is a prime number
19 is a prime number
23 is a prime number
29 is a prime number
31 is a prime number
37 is a prime number
41 is a prime number
43 is a prime number
47 is a prime number
53 is a prime number
59 is a prime number
61 is a prime number
67 is a prime number
71 is a prime number
73 is a prime number
79 is a prime number
83 is a prime number
89 is a prime number
97 is a prime number
```

Figure 2-2. Output Resulting from DEMO.PSC
- 2-2 -

1. If the PASCAL command line ends with an equals sign (=), or when the line ends on parenthesized data (i.e., data preceded by a left parenthesis but not followed by a right one), the compiler generates the prompt

PAS..

to the OC unit to indicate that it has completed the command sequence specified so far and is ready to accept additional input. The programmer can then enter a continuation of the PASCAL command from the CI unit.

2. The filename extension is optional in a file identifier specified as a source medium. If the extension is omitted, the compiler searches for the .PSC version. If that version is not found, an error occurs.
3. The filename extension is also optional in a file identifier specified as the destination medium. If the extension is omitted, the compiler writes the output to the .ASM version of the file.
4. If the source medium is a file, the filename is also optional in a filename identifier specified as the destination medium. That is, if the destination medium is a file-type device, the compiler writes the object code to a file on that device. This file has the same filename as the source file and the .ASM extension.
5. If a destination medium is a file, the command overwrites any contents of that file. The command has no effect on source media.

2.2.2 PASCAL Command Options

The programmer can specify any of the following options (Table 2-1). An option must be separated by commas and enclosed in parentheses in the PASCAL command line. If conflicting options are specified in a PASCAL command (e.g., TRACE and NOTRACE), the compiler recognizes the last option. The underlined characters in the names of the options listed below are the minimum number required to uniquely identify a particular option.

An asterisk preceding an option in the following list indicates that the option can also be specified within the source text of the program itself. This method of specifying options as directives is described in Section 2.3.

Table 2-1. NM4 PASCAL Command Options (Page 1 of 2)

OPTION	MEANING
* <u>ASSERT</u>	Default option. Code is included at runtime to test assertions and produce a runtime error if the assertion is false.
* <u>NOASSERT</u>	Code to test assertions is omitted. The assert source statement is still recognized and processed by the compiler, but it has no effect during program execution.
<u>ERRORSONLY</u>	Includes only error lines in the compilation listing.
<u>NOERRORSONLY</u>	Default option. All source lines are output to listing, unless the overriding NOLISTING option is used.
* <u>INTEGERSET</u> =nnnn	This option allows the programmer to define the size of the set of integers in terms of bits. The default size is 16, which defines the limit of integers to 0 to 65535 (unsigned), or -32768 to +32767 (signed).
* <u>NOLINENUM</u>	Default option. Runtime code to keep track of the source line number is omitted.
<u>LISTING</u> =[file]	Generates a compiler listing. If only the option word appears, the listing is generated to the L0 unit (default). If an equals sign also appears, the listing is generated to the .LIS version of the source medium, which must be a file. If a file identifier also appears, the listing is generated to that file. (In this case, the file name extension is optional; by default, the listing is generated to the .LIS version of the file.
<u>NOLISTING</u>	Does not generate a compiler listing. Note: This option overrides the subordinate option of ERRORSONLY.
<u>MACHINE</u> =ID	Specifies the machine that the program is to be executed on. ID is one of the following: 410 NAKED MINI@ 4/10 410A optioned NAKED MINI@ 4/10 430 NAKED MINI@ 4/30 430A optioned NAKED MINI@ 4/30 490 NAKED MINI@ 4/90 490A NAKED MINI@ 4/90 with Floating Point Option If the MACHINE option is not included, the default machine is the NAKED MINI@ 4/10.

Table 2-1. NM4 PASCAL Command Options (Page 2 of 2)

OPTION	MEANING
<u>NOMACHINE</u>	Signifies that the program is not specified for any particular machine. This is equivalent to MACHINE = 490A.
<u>OBJECT</u>	Default option. Object code (in the form of assembler source) is output.
<u>NOOBJECT</u>	The compilation does not create output code. Useful during debugging the PASCAL source program when it is known that the output will not be executable.
* <u>RANGE</u> CHECK	Default option. Code is inserted to check subrange variables against upper and lower subrange limits.
* <u>NORANGE</u> CHECK	Code to check subrange limits is not included.
* <u>STANDARD</u>	The compiler generates a warning to the listing whenever it encounters a nonstandard feature.
* <u>NOSTANDARD</u>	Default option. Nonstandard features are not flagged.
* <u>TRACE</u>	This option is checked by the compiler whenever a procedure is entered and exited. If the option is requested, a call to a system routine is inserted. This routine outputs a trace message at runtime.
* <u>NOTRACE</u>	Default option. Trace calls are not generated.

2.2.3 Command Line Examples

The following examples illustrate some of the parameter combinations that can be called by a PASCAL command.

Example 1:

Format: PAS medium(options)
 Example: PAS PROGRAM(TRACE)

Example 1 causes PASCAL source code to be input from and assembly language source to be output to the same device using the same filename.

This example causes source code to be read from the file named PROGRAM. Since no unit or device is specified, and since the file extension is not given, the system searches the UF device for the file with an extension of .PSC. (If none is found, an error occurs.) Assembler source is output to file UF.PROGRAM.ASM. The compilation listing is output to the L0 device.

Example 2:

Format: PAS medium1=medium2(options)
Example: PAS DM1.TESTOUT=TESTIN(E)

This format is used when the output file is to reside on a different medium from the input file and/or it is desired that the name of the output file differ from that of the input file. The input is from UF.TESTIN.PSC (with listing of errors only) and the output is to DM1.TESTOUT.ASM.

2.3 NM4 PASCAL DIRECTIVES

NM4 PASCAL provides the capability to set or reset some of the compiler options listed in Table 2-1 (specifically, those marked with an asterisk) by specifying them in the source code of the program being compiled. These options are shown in Table 2-2. Except for INTEGERSET and STANDARD/NOSTANDARD, these options may be set or reset within the program at any point. INTEGERSET and STANDARD/NOSTANDARD, if specified, must appear prior to the 'program' or 'task' keyword. Options specified in the source program are enclosed by metabraces {}.

The form of specification is the first letter of the option, preceded by a dollar sign (\$) and appended with a plus sign (+) to invoke the option or a minus sign (-) to invoke no option. INTEGERSET requires a specific integer value.

Example:

{ \$r+,s-,i16 }

This example tells the program to do the following:

- Do rangechecking (r+)
- Nonstandard (with extensions) (s-)
- Allow for 16 bit integers (i16).

These options precede the "program" statement.

Options not listed in Table 2-1 can only be invoked in the command line and cannot be embedded in the program text.

2.4 COMPILER ERROR REPORTS

The error reports produced by the NM4 PASCAL compiler are output in English and are accompanied by the local context. An example of a PASCAL program with errors is shown in Figure 2-3.

The programmer should be aware that, because of the block structured nature of the language, a coding error at a particular point in a program may not be detectable until many further statements have been processed. Hence, an error can be flagged at a point distant in the program from the original coding error.

Table 2-2. NM4 PASCAL Directives

COMMAND LINE	IN-LINE
ASSERT	{\$a+};
NOASSERT	{\$a-};
INTEGERSET=16	{\$i16}; (prior to 'program' only)
LINENUM	{\$l+};
NOLINENUM	{\$l-};
RANGECHECK	{\$r+};
NORANGECHECK	{\$r-};
STANDARD	{\$s+}; (prior to 'program' only)
NOSTANDARD	{\$s-}; (prior to 'program' only)
TRACE	{\$t+};
NOTRACE	{\$t-};

```
ERROR.PSC 1981 Jun 17 18:14:26.25
1      program listprimes(output) ;
2      {this program calculates and prints all prime
3        numbers from 1 to 100.}
4
5      type positiveinteger = 1..maxint ;
6
7
8      function prime (p : positiveinteger) : boolean ;
9      {prime determines whether p is prime or not}
10     var root, divisor : positiveinteger ;
11     begin
12         if p<4 then prime := p>1 else
13         if not odd(p) then prime := false else
14         begin
15             root := round (sqrt(p)) ;
16             divisor := 3 ;
17             while (divisor<=root) and (p mod divisor <> 0)
18             do divisor := divisor + 2
19             prime := divisor>root
( 0)      ↑
term: multiplying operator or end of term expected
20         end
( 19)      ↑
assignment: left and right hand side incompatible
21         end ; {prime}
22
23     begin
24         for i := 1 to 100 do
( 20)      ↑
identifier 'i' not declared
25         if prime(i)
( 24)      ↑
identifier 'i' not declared
26         then writeln (i, ' is a prime number') ;
( 25)      ↑
identifier 'i' not declared
27     end.
( 26) 5 fatals 0 warnings
```

Figure 2-3. Sample Listing of PASCAL Program with Error Printout ERROR.PSC

SECTION 3

POSTCOMPILATION PROCESSES

A successful PASCAL compilation results in an output file in the form of assembler source code. To produce an executable PASCAL program, two additional steps are required: assembly and linking.

3.1 ASSEMBLY

In an assembly, source code is input to MACRO, the NM4 macro assembler; the output is in the form of NM4 object code. The rules for invoking the assembler are described in Section 3 of the OS4 System Reference Manual, but in general the command

MACRO USERFILE (options)

inputs UF.USERFILE.ASM and output UF.USERFILE.OBJ, for example.

Since the assembler source is produced by the PASCAL compiler rather than by the programmer, many options (such as D= to denote definition files) are not needed.

It is a good idea to generate a listing of the assembly for debugging purposes. This is done by using the NOERRORONLY! option in MACRO. Once debugging is complete, the assembly can be run with the ERRORONLY option to verify that no assembly errors occurred.

3.2 LINKING

Not all the code needed to run the user's program is generated during compilation. Many operations, such as the standard I/O and file management routines, reside as subroutines on CA-supplied object libraries, and are merely called, or referenced, by the compiler (and assembler) output. In addition, since NM4 PASCAL supports separate compilations as well as interfacing to user-supplied assembly language routines, these various modules must be connected in some way before execution of the final program. The NM4 Linker performs this function. A full description of linker usage and options is found in the LINK Editor Utility Reference Manual.

3.3 LINKING FOR EXECUTION UNDER OS4

To link an NM4 PASCAL program for execution under OS4 the user enters the LINK command, based on the following example, which can be used to link a NM4 PASCAL program for execution under OS4:

LINK OS4PASCPRG=USERPROG+SUB1(UL)+PASCAL.LIB+OS4.LIB

This command produces an executable PASCAL program on UF called OS4PASCPR0G.BIN from PASCAL program modules in files USERPR0G.OBJ, SUB1.OBJ, PASCAL library routines in the PASCAL.LIB file, and system routines in OS4.LIB.

1. The primary input file is USERPR0G.OBJ, is presented to the linker first.
2. Any additional user PASCAL and assembler object files are unconditionally linked after the primary input file.
3. The standard file PASCAL.LIB is then linked.
4. The file OS4.LIB is the final input file.

The resulting linked program can be loaded under any OS4 system; relinking is not necessary.

3.4 LINKING FOR EXECUTION UNDER RTX4

To link an NM4 PASCAL program for execution under RTX4, the user enters the LINK command, based on the following example:

```
LINK  RTXPA SCPR0G=USERPR0G+PASCALRTX+SFM+IOS+RTX
```

SECTION 4

RUNTIME

4.1 LOADING AND EXECUTION UNDER OS4

A compiled and linked PASCAL program can be loaded and executed by typing the following syntax:

```
UF/PROG  out  = in (file1,file2,....,filen)
```

where: out = SO (also the standard PASCAL output file)
 in = SI (also the standard PASCAL input file)

PROG = the name of the PASCAL program

file1,file2,...,filen match the files specified in the PASCAL source program statement.

4.2 EXECUTION UNDER RTX4

A compiled and linked PASCAL/RTX program is loaded with the AUToload command. The RTX Unit Assignment Table must contain entries for 'SF', 'SO', 'SL', and 'OV'

where:

'SI' will be read whenever INPUT is read
'SO' will be written whenever OUTPUT is written
'SL' will be written on when errors occur
'SF' must contain the file PAS:RTERR.SDT
'OV' is used as a temporary LUN by I/O runtime.

See Appendix A for instructions on how to access a file by name.

SECTION 5

INTERFACING TO ASSEMBLY LANGUAGE

This section provides information on the interface between NM4 PASCAL and assembly language in the case when a user wishes to:

- Code an assembly language function and call it from an NM4 PASCAL program
- Call an NM4 PASCAL function from assembly language.

5.1 ASSEMBLY LANGUAGE FUNCTIONS CALLED FROM PASCAL

Assembly language functions called from PASCAL are discussed in the following paragraphs.

5.1.1 Entering and Returning from Functions

When a NAKED MINI[®] 4 PASCAL program calls a function, it does so by the following code:

ADD	=n,Y	SET STACK FRAME LEVEL
JSK	ROUT	CALL THE FUNCTION
SUB	=n,Y	RESTORE STACK FRAME

where ROUT is the address of the function

The sequence for an assembly language version of ROUT is:

ROUT	EQU	\$	LINK WORD
------	-----	----	-----------

A suitable return could be:

RSK

5.1.2 Parameters of Functions

On entry, function parameters are positioned as follows:

1. The first parameter is in the A register
2. The second parameter is in the Q register
3. For subsequent parameters, parameter n is in location n+1 of the stack frame (i.e., at n+1(Y)).

If the A and Q registers are not required immediately (or will be required again later), they can be stored in 2(Y) and 3(Y), respectively. If a function has n parameters, then locations n+2 and above of the current stack frame are available for local variables and temporary storage.

5.1.3 Returning Results

A function can return only one result. The result of a function is expected in the A register on return.

5.2 PASCAL FUNCTIONS CALLED FROM ASSEMBLY LANGUAGE

An assembly language program can call a NAKED MINI@ 4 PASCAL function by setting up the necessary parameters and calling the appropriate function via a JSK instruction.

A function with four parameters, P1 to P4, and name ROUT could be called by the following code:

COPY	P3,A	
COPY	A,n+4(Y)	STORE PARAMETER 3 ON STACK
COPY	P4,A	
COPY	A,n+5(Y)	STORE PARAMETER 4 ON STACK
COPY	Y,n+1(Y)	
COPY	P1,A	LOAD PARAMETER 1 IN A REGISTER
COPY	P2,Q	LOAD PARAMETER 2 IN Q REGISTER
ADD	=n,Y	SET STACK FRAME LEVEL
JSK	ROUT	CALL THE FUNCTION
SUB	=n,Y	RESTORE STACK FRAME

It is assumed here that the calling function has N-2 local variables and parameters. Note that in this code, location N+4 of the stack frame is location 4 of the called function's stack frame, which corresponds to parameter 3. On return from this call to function ROUT, the A register contains the result (if any).

SECTION 6

REAL-TIME SERVICES

This section describes RTX4 TASK declarations and lists the TASK services. See Appendix A for a list of procedures to manipulate I/O assignments with RTX.

6.1 TASK SERVICES

The RTX4 services described in this section may be called from NM4 PASCAL when operating in the RTX4 environment. The name of each service to be called from the PASCAL program must first be defined as a PASCAL function or procedure. Figure 6-1 lists the full set of service functions that can be specified. Only the specific services to be used in the program need be defined.

When a service is called in the body of the PASCAL program, the completion status for the service is returned as a value in the calling function. If the returned function equals zero, the service has successfully completed; if it is a non-zero value, an error has occurred. The user can thus test the value of the returned function for appropriate error routing. For example, a call to signal a semaphore and conditionally branch to an error routine could be coded in one statement as follows:

```
if p_sig (<semaphore name>)<>0 then error;
```

Another method would be to assign the function value to a variable for later evaluation as follows:

```
errorcd :=p_sig (<semaphore name>);
```

If the user does not require a service completion error test, the service name can be defined as a procedure rather than as a function.

6.1.1 Task Control

BEGIN AN ACTIVITY (P_BGIN)

This enables the user to begin an activity of a task from PASCAL. The format is:

```
P_BGIN (tdb, priority, a, q, x)
```

where: tdb denotes a Task Descriptor Block.

priority is an integer value or variable containing the activity's priority, as described in the Real-Time Executive RTX4 Reference Manual.

```

PASCAL RTX.PSC 1931 Nov 12 14:32:05.25
1  {id+}
2  {id+}
3  type
4  byte = 0..255;
5  pridesc = integer; { >=0 absolute, <0 relative }
6  semaphore = array[1..4] of integer;
7  mailbox = array[1..12] of integer;
8  datetime = packed array[1..14] of char;
9
10 function p_sdb(var sem: semaphore; initial: byte): integer; extern;
11 function p_slg(var sem: semaphore): integer; extern;
12 function p_wait(var sem: semaphore): integer; extern;
13
14 function p_gpri(var pri: pridesc): integer; extern;
15 function p_spri(pri: pridesc): integer; extern;
16
17 function p_mdb(var mail: mailbox): integer; extern;
18 function p_send(msg: long; var mail: mailbox): integer; extern;
19 function p_recv(var msg: long; var mail: mailbox): integer; extern;
20
21 function p_itlc(id: integer; var sem: semaphore; count: integer): integer; extern;
22 function p_mtlic(id: integer; var sem: semaphore; count: integer): integer; extern;
23 function p_ctlic(id: integer): integer; extern;
24 function p_awal(id: integer; var sem: semaphore; count: long): integer; extern;
25 function p_iwal(id: integer; var sem: semaphore; count: long): integer; extern;
26 function p_cwal(id: integer): integer; extern;
27 function p_gtcd(var time: long): integer; extern;
28 function p_stcd(time: long): integer; extern;
29 function p_sstd(var time: datetime): integer; extern;
30 function p_setcd(var time: datetime): integer; extern;
31
32 function p_clnt: integer; extern;
33 function p_paus(pri: pridesc): integer; extern;
34
35 function p_pgin(procedure tdb; pri: pridesc; areg, qreg, xreg: integer): integer;
    extern;
36 { actually tdb is a task descriptor block, however it must be declared
37   as an external procedure so that we can pass its address to p_pgin }

```

Figure 6-1. PASCAL Functions for RTX4 Services
- 6-2 -

a, q, x are integer values or variables containing the initial values of the A, Q, and X registers respectively, upon execution of an assembly language task, or the areg, qreg, and xreg integer values of a PASCAL task.

GET CURRENT TASK PRIORITY (P_GPRI)

P_GPRI (priority)

where: priority is the name of an integer variable into which the priority of the PASCAL activity is stored.

SET CURRENT ACTIVITY PRIORITY (P_SPRI)

P_SPRI (priority)

where: priority is an integer value or variable containing the desired priority of the PASCAL activity, as described in the Real-Time Executive RTX4 Reference Manual.

ROUND ROBIN (ACTIVITY PAUSE) SCHEDULING (P_PAUS)

P_PAUS (priority)

where: priority is an integer value or variable, containing the priority of the calling program.

6.1.2 Semaphore

CREATE A SEMAPHORE DESCRIPTOR BLOCK (P_SDB)

This sets up the needed semaphore table structure required by other service calls that deal with semaphores. The format is:

P_SDB (semaphore name, initial value)

where: semaphore name is the name of a four-word integer array which serves as the Semaphore Descriptor Block (SDB).

initial value is an integer value or variable name containing an initial semaphore value from 0 to 255, as described in the Real-Time Executive RTX4 Reference Manual.

SIGNAL A SEMAPHORE (P_SIG)

P_SIG (semaphore name)

where: semaphore name is the name of a four-word integer array which serves as the SDB.

WAIT ON A SEMAPHORE (P_WAIT)

P_WAIT (semaphore name)

where: semaphore name is the name of a four-word integer array which serves as the SDB.

6.1.3 Mailbox

CREATE A MAILBOX DESCRIPTOR BLOCK (P_MDB)

This call sets up the needed mailbox table structure required by mailbox manipulation calls listed below. The format is:

P_MDB(mailbox name)

where: mailbox name is the name of a 12-word integer array which serves as the Mailbox Descriptor Block (MDB).

SEND A MESSAGE (P_SEND)

P_SEND (mailbox name, message)

where: mailbox name is the name of a 12-word integer array which serves as the MDB.

RECEIVE A MESSAGE (P_RECV)

P_RECV (mailbox name, message)

where: mailbox name is the name of a 12-word integer array which serves as the MDB.

6.1.4 Clock

INITIALIZE TICK CLOCK (P_ITIC)

P_ITIC (ID, semaphore name, tick count)

where: ID is an integer expression which serves as the timer identifier, as described in the Real-Time Executive RTX4 Reference Manual.

semaphore name is the name of a four-word integer array which serves as the SDB.

tick count is an integer value or variable containing the number of ticks that must elapse before the semaphore is signaled.

MODIFY TICK CLOCK (P_MTIC)

P_MTIC (ID, semaphore name, tick count)

where: All parameters the same as P_ITIC.

CANCEL TICK CLOCK (P_CTIC)

P_CTIC (ID)

where: ID is an integer expression which serves as the timer identifier (ID), as described in the Real-Time Executive RTX4 Reference Manual.

6.1.5 Date and Time

WALL CLOCK ABSOLUTE TIMER (P_AWAL)

P_AWAL (ID, semaphore name, time)

where: ID is an integer expression which serves as the timer identifier, as described in the Real-Time Executive RTX4 Reference Manual.

semaphore name is the name of the third element of a four-word integer array which serves as the SDB.

time is a two-word variable (either a long integer or two-word integer array) that contains the absolute wall clock time in binary, as described in the Real-Time Executive RTX4 Reference Manual.

WALL CLOCK INTERVAL TIMER (P_IWAL)

P_IWAL (ID, semaphore name, time)

where: ID is an integer expression which serves as the time identifier, as described in the Real-Time Executive RTX4 Reference Manual.

semaphore name is the name of the third element of a four-word integer array which serves as the SDB.

time Same as P_AWAL, but representing a time interval, as described in the Real-Time Executive RTX4 Reference Manual.

CANCEL WALL CLOCK TIMER (D_CWAL)

P_CWAL (ID)

where: ID is an integer expression which serves as the timer identifier, as described in the Real-Time Executive RTX4 Reference Manual.

SET TIME OF DAY (P_STOD)

P_STOD (time)

where: time is a two-word variable (e.g., long integer) containing the time of day value. This variable takes the place of the AQ register contents described in the Real-Time Executive Reference Manual.

GET TIME OF DAY (P_GTOD)

P_GTOD (time)

where: time is a two-word variable, as described in the P_STOD call, into which the time of day value is stored.

SET ASCII TIME OF DAY (P_SATD)

P_SATD (ASCII time)

where: ASCII time is a seven-word integer array. The contents of this array are as described in the Real-Time Executive RTX4 Reference Manual.

GET ASCII TIME OF DAY (P_GATD)

P_GATD (ASCII time)

where: ASCII is a seven-word integer array. The ASCII time is stored in this array in the format described in the Real-Time Executive RTX4 Reference Manual.

6.1.6 Console Interrupt

SERVICE CONSOLE INTERRUPT REQUEST (P_CINT)

P_CINT

This request has no parameters.

SECTION 7

SYSTEM FILES

The following files must be on SF to compile, link, and execute a PASCAL program.

- PAS.SYS
- PAS:OPT.SYS
- PAS:LIST.SYS
- PAS:TRAN.SYS
- CG4.SYS
- PAS:ERRORS.SDT
- PAS:TABLES.SDT
- PASCAL.LIB
- PAS:RTERR.SDT

APPENDIX A

GENERAL ROUTINES

The routines shown in Figure A-1 permit the user to access files when interfacing with RTX4.

```
{5c+}
const      bufsize = ?;
type      br1 = 1..bufsize;
          br2 = 0..bufsize;
          buf = packed array(br1) of char;
```

{all routines must be declared extern}

```
function   argc:integer;
function   argv(i:integer):string;
procedure  argshift;
procedure  notext(var f:text);
procedure  diag(var f:text);
procedure  pcreat(var f:text; s:string);
procedure  popen(var f:text; s:string);
function   strbuf(var b:buf):string;
function   strtobuf(s:string; var b:buf; len:br1):br2;
function   strlen(s:string):integer;
```

- | | |
|----------|---|
| argc | Gives the number of arguments provided when the program is called. |
| argv | Selects the specified argument from the argument list and returns a pointer to it. This pointer is nil if the index is out of bounds (<0 or >=argc). |
| argshift | Effectively deletes the first argument from the argument list. argv[2] becomes argv[1], argv[3] becomes argv[2], etc. It is a useful procedure to skip optional flag arguments. Note that the matching of arguments and files is done at the time a file is opened by a call to reset or rewrite. |
| notext | Only useful for input files. End of line characters are not replaced by a space and character codes out of the ASCII range (0..127) do not cause an error message. |
| diag | Initialize a file for output on the diagnostic output stream ('SL'). Output is not buffered. |
| pcreat | The same as rewrite(f), except that you must provide the filename yourself. The name must be zero terminated. Only text files are allowed. |
| popen | The same as reset(f), except that you must provide the filename yourself. The name must be zero terminated. Only text files are allowed. |

Figure A-1. General RTX4 Interface Routines (Page 1 of 2)

`strbuf` type conversion from character array to string. It is your own responsibility that the string is zero terminated.

`strtoout` copy string into buffer until the string terminating zero byte is found or until the buffer is full, whatever comes first. The zero byte is also copied. The number of copied characters, excluding the zero byte, is returned. So if the result is equal to the buffer length, then the end of buffer is reached before the end of string.

`strlen` Returns the string length excluding the terminating zero byte.

Figure A-1. General RTX4 Interface Routines (Page 2 of 2)