RC International

Reference Manual

for RC3502

Real-Time Pascal

PN: 99001174

Keywords:
Systems programming language, real-time, parallel processes, data communications network, RC3502, RC8000.

Abstract:
The contents of this manual describe the Real-Time Pascal implementation for the RC3502 computer. The Real-Time Pascal programming language is a high level Pascal-like language, designed to express algorithms and their implementation as parallel cooperating processes, executing on a network of processing components. The Real-Time Pascal language is defined in: Reference Manual for the Programming Language Real-Time Pascal (PN: 99110141).
Information on how to use the Real-Time Pascal cross compiler and the associated utility programs are included in appendices.

Date:
January 1990.

Authors:
Bo Bagger
Jan Bardino

Disclaimer

RC International makes no representations or warranties with respect to the contents of this publication and specifically disclaims any implied warranties of merchantability or fitness for any particular purpose. Furthermore, RC International reserves the right to revise this publication and to make changes from time to time in the content hereof without obligation of RC International to notify any person of such revision or changes.

FOREWORD.

This edition of the RC3502 Real-Time Pascal Reference Manual is updated according to release 5 of the RC3502 operating system package and release 4 of the RTP3502 package. The package numbers are SW2001 and SW2111, respectively. RTP3502 is the RC8000 cross compiler implementation for RC3502 of the Real-Time Pascal language as defined in Reference Manual for the Programming Language Real-Time Pascal (PN: 99110141).

TABLE OF CONTENTS

Page

Appendices

1. INTRODUCTION

This manual describes the RC8000 cross compiler implementation of the programming language Real-Time Pascal for the RC3502 machine.

1.1 Organisation of this Manual

Chapter 2 contains a description of how the implementation dependent details of the language are realized. Information on deviations between the language definition and the actual implementation is supplied.

Chapter 3 describes all the available routines of the system libraries.

Chapter 4 describes the standard processes comprising the run time environment for RC3502 Real-Time Pascal programs.

Appendix A contains references.

Appendices B, C, and D describe the use of the RC3502 Real-Time Pascal compiler and the error messages.

Utilities for manipulation of source and object programs are described in appendices E and F.

Appendix G describes how a load file or autoload file is generated on an RC8000.

Appendix H contains a complete list of language symbols.

Appendices I and J contain a list of all predefined constants, types, and routines.

Appendix K contains a list of operator input/output type definitions and routine declarations.

Appendix L is a more comprehensive example using the OPERATOR input/output routines.

Appendix M contains a complete list of exception codes.

Appendix N contains the figure index and the catchword index. A : appended to a catchword indicates that the reference is for a syntax diagram. An = appended indicates that the reference is for a definition of a constant or type.

2. IMPLEMENTATION DESCRIPTION

This chapter contains descriptions of how implementation dependent details of the language are realized in the RTP3502 cross compiler. As far as possible, the ordering follows that of the language reference manual.
A reference to the appropriate section of the language reference manual /RTP/ is given where possible. The format is (cf. RTP-x.y) for a reference to section x.y.

2.1 Names

(cf RTP-2)
The danish letters æ, ø, å, Æ, Ø, and Å are accepted as members of the category 'letter'.

2.2 Keywords

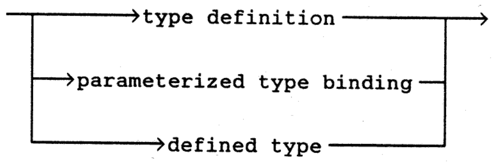
(cf. RTP-2.5.1)
In RTP3502 the list of keywords is extended with the following words. The total set of reserved words (and symbols) are given in appendix H.

Keyword	use
BEGINBODY	internal
CHANNEL	interrupt handling, cf. Driver Processes, section 4.3
EXPORT	internal
LABEL	label declarations, cf. 2.13.2
PREFIX	heading of separately compiled routine, cf. 2.16

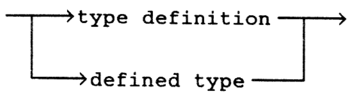
2.3 Specification of Types

(cf RTP-3.1)
The augmented form of type specification for formal parameter lists is not supported. The so-called open families are not implemented. The syntax diagrams for the current implementation are:

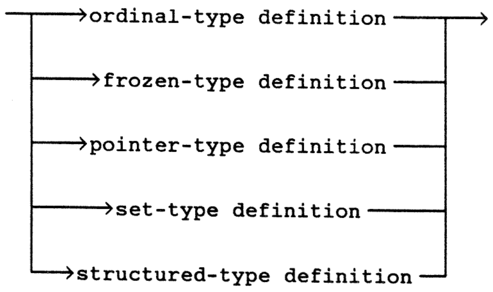
common type specification:



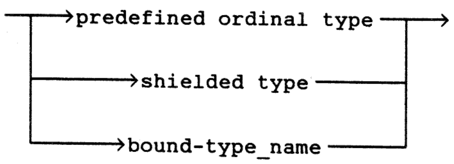
formal type specification:



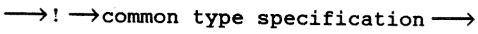
type definition:



defined type:



frozen-type definition:



The frozen type concept of earlier implementations of RTP3502 is still supported. An object of a frozen type cannot be changed and actual parameters for formal parameters of frozen type may be constants, even if the formal parameter is of kind VAR, i.e. frozen VAR parameters are equivalent to parameters of kind INSPECT.

2.4 The Type Integer

(cf. RTP-3.4.3)
The type double is not supported as a built in type with infix operators. But the type exists and operations on objects of type double are performed by means of routine calls. The necessary routines are defined in the context named doubleenv, which is part of the operating system package.

2.5 Set Types

(cf. RTP-3.5)

set-type definition:

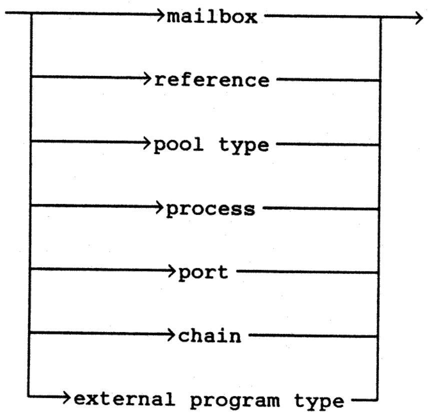
⟶SET ⟶OF ⟶common type specification ⟶

The common type specification specifies the element type which must be an ordinal type, and furthermore restricted to cover positive valued elements only. And set constants are further restricted only to contain elements inside the range 0..1023.

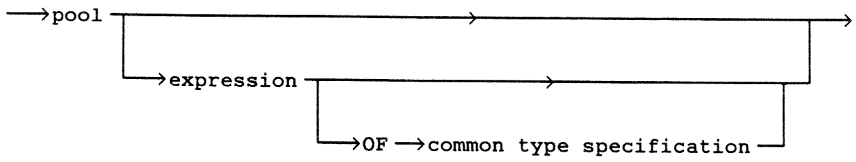
2.6 Pools

(cf. RTP-3.7)

shielded type:



pool type:



The value of expression of the pool type is the initial number of messages allocated to the pool object, i.e. the cardinality of the pool.
The type size of common type specification is the buffersize of the messages allocated to the pool object. If it is absent the messages are allocated with no message buffer. The non standard variants of pool type definition are equivalent to the simple definition followed by a call of allocpool, with the value of expression as 'no_of_messages' parameter and the value of typesize ('common type_specification') as 'bufsize' parameter. The value 0 is used if no type is specified. The size of the allocated buffer(s) is an integral number of words, i.e. an even number of bytes.
On the RC3502 there is only one kind of memory available for allocation of objects, therefore the function allocmempool and the type mem_type are not supported.

2.7 Message Buffer Attributes

(cf. RTP-3.7)
The attributes 'offset', 'top', and 'bytecount' are implemented by means of the first three words in the message buffer, representing the attributes named 'first', 'last', and 'next' in earlier versions of Real-Time Pascal systems for the RC3502.

The relations between the new and the old versions of these attributes during retrieval or update are:

offset(r)	: first
top(r)	: last+1; -- modulo arithmetic
bytecount(r)	: next-first; -- unsigned arithmetic
setoffset(r, val)	: savebytecount:= next-first; -- modulo
	: first:= val;
	: next:= val+savebytecount; -- modulo
settop(r, val)	: last:= val-1; -- modulo
setbytecount(r, val)	: next:= first+val; -- unsigned

Note: The 'next' attribute is updated as a sideeffect of 'setoffset', in order to obtain, that the 'bytecount' attribute is independent of the 'offset' attribute.

2.8 Structured Values

(cf. RTP-3.8.3)
Structured values are only supported as structured constants, i.e. all components must be constants and the types cannot be dynamic. Note that a bound parameterized type is dynamic, even if the parameters of the binding are constants. Furthermore, the result of calling the predefined functions abs, chr, ord, pred, and succ is treated as a variable, even if applied on a constant.

2.9 String Type

(cf. RTP-3.8.4)
The type string is not predefined and therefore not supported with built-in truncation and/or extension. But literal text strings are truncated/extended if assigned to a variable of a static character array type.

2.10 Representation and Layout of Variables

(cf. RTP-3.10)
In this chapter it is explained how values of the various types of Real-Time Pascal are represented in the RC3502 implementation and how memory is allocated and laid out to hold the values of the variables of a process.

Note: The object layout for the RC3502 differs from the prescribed layout of descriptive types.

2.10.1 Representation of Values

2.10.1.1 Enumeration Types

An enumeration type is defined as consisting of a finite, totally ordered set of values, corresponding to a set of ordinal values which is a subset of the integral numbers. The representation of a value of an enumeration type is the two's complement representation of the corresponding ordinal value. If a type includes negative ordinal values, the representation of values of the type is always in 16 bits (a word). If the type includes only non-negative ordinal values, and n is the largest of these, then the representation is in $\log_2(n+1)$ bits (at most 16). Examples:

- integer values (-32768..32767) are represented in 16 bits,
- boolean values (false, true) are represented in 1 bit,
- char values are represented in 8 bits,
- values of the subrange type -3..7 are represented in 16 bits,
- values of the scalar type (red, green, blue, orange, pink) are represented in 3 bits.

2.10.1.2 Shielded and Pointer Types

The representation of values of shielded and pointer types is not revealed.

2.10.1.3 Set Types

The representation of values of a set type uses a bit vector whose length depends on the base type. The base type must be an enumeration type which does not include negative ordinal values.

If the ordinal value set of the enumeration type T is the range m..n ($m \geq 0$) then values of the type set of T are represented in a bit vector with indices from 0 to n. The first m (possibly zero) of these bits are significant only in comparisons. If the element of T whose ordinal value is i ($m \leq i \leq n$) is a member of a particular value of type set of T, then bit i in the representation of that value is 1, otherwise it is 0. Example:

```
TYPE
      s_type= SET OF 3..5;

VAR
      s_var: s_type:= (.3, 5.);
```

The value of s_var is represented as shown:

bit	0	1	2	3	4	5
	▨	▨	▨	1	0	1

The shaded bits are significant only in comparisons.

2.10.1.4 Structured Types

Values of array or record types are vectors of values of enumeration, shielded, pointer and set types. The component values are represented as described for the component types.

2.10.1.5 Bit Size of Type

The bit size of a type T, denoted S(T), is the number of bits used to represent values of type T. The concept of size is only relevant for types which are affected by packing when used for components of structured types, and is therefore not defined for all types. For enumeration types S(T) is computed as described above.

Example: S(char)=8, S(integer)=16.

2.10.2 Memory Layout

The memory requirement of a type T, denoted M(T), is defined as the number of bytes which are allocated for a variable of type T. The value M(T) is the result of a call of typesize(T).

For an enumeration type T, M(T) depends on S(T), as follows:

```
1 <= S(T) <=  8 : M(T)=1
9 <= S(T) <= 16 : M(T)=2
```

For shielded and pointer types, M(T) is the following:

M(reference)=	7
M(mailbox)=	7
M(process)=	6
M(pointer)=	3
M(pool)=	7
M(chain)=	8
M(program_descriptor)=	25
M(port)=	55

Memory requirement for structured types is described in connection with memory layout for these types in the following subsections.

2.10.2.1 Alignment

The memory of the RC3502 machine consists of a sequence of eight bit bytes. Each byte has an address. Two consecutive bytes constitute a word. The most significant halfword is the byte with the lowest address.

Memory allocation for a variable of a shielded type, or of a stuctured type containing components of shielded type(s) is physically word-aligned, i.e. the allocated memory starts on a word boundary, where the first byte has an even address. All other variables are byte-aligned.

2.10.2.2 Stack Frame

Memory for variables declared in program or routine blocks is allocated in *stack frames* in the data structure of the process to which the variables belong. The general layout of a process stack is shown in fig. 2.1.

In the portion of a stack frame used for declared variables memory is allocated from low to high addresses in the order of declaration of the variables in the program text.

An integral number of bytes is allocated for each variable. The number of bytes is determined by the memory requirement of the type of the variables. Because of physical word-alignment, unused bytes of memory may be left between variables (in fact also inside structured variables).

When more memory is allocated for a variable of an enumeration type than indicated by the bit size of the type, the variable is placed in the least significant bits of the allocated byte or word.

Example: Corresponding to the declarations:

```
VAR
      a: char;
      b, c: 0..7;
      d: integer;
      e: reference;
```

Memory is allocated within a stack frame as illustrated in fig. 2.2.

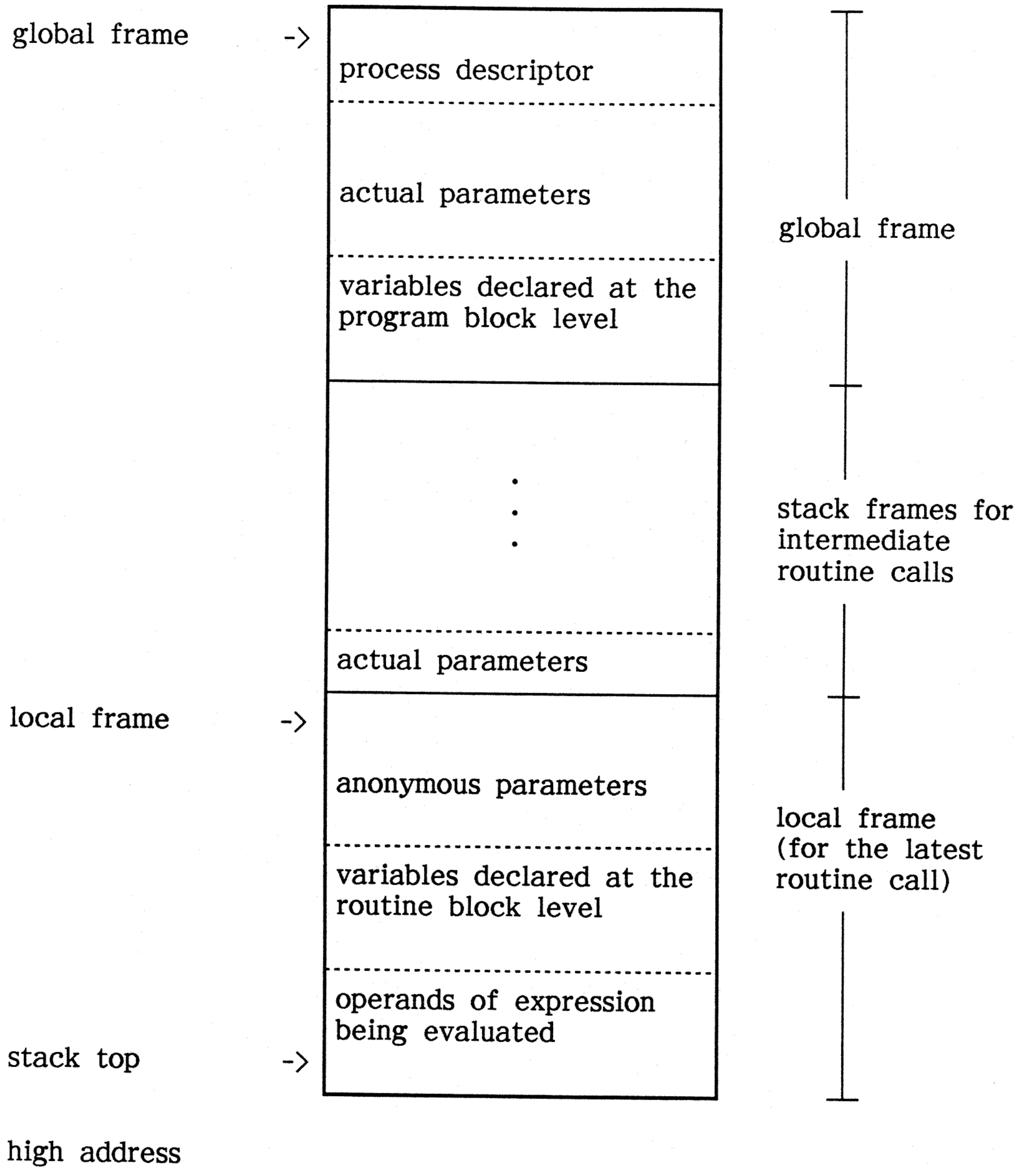
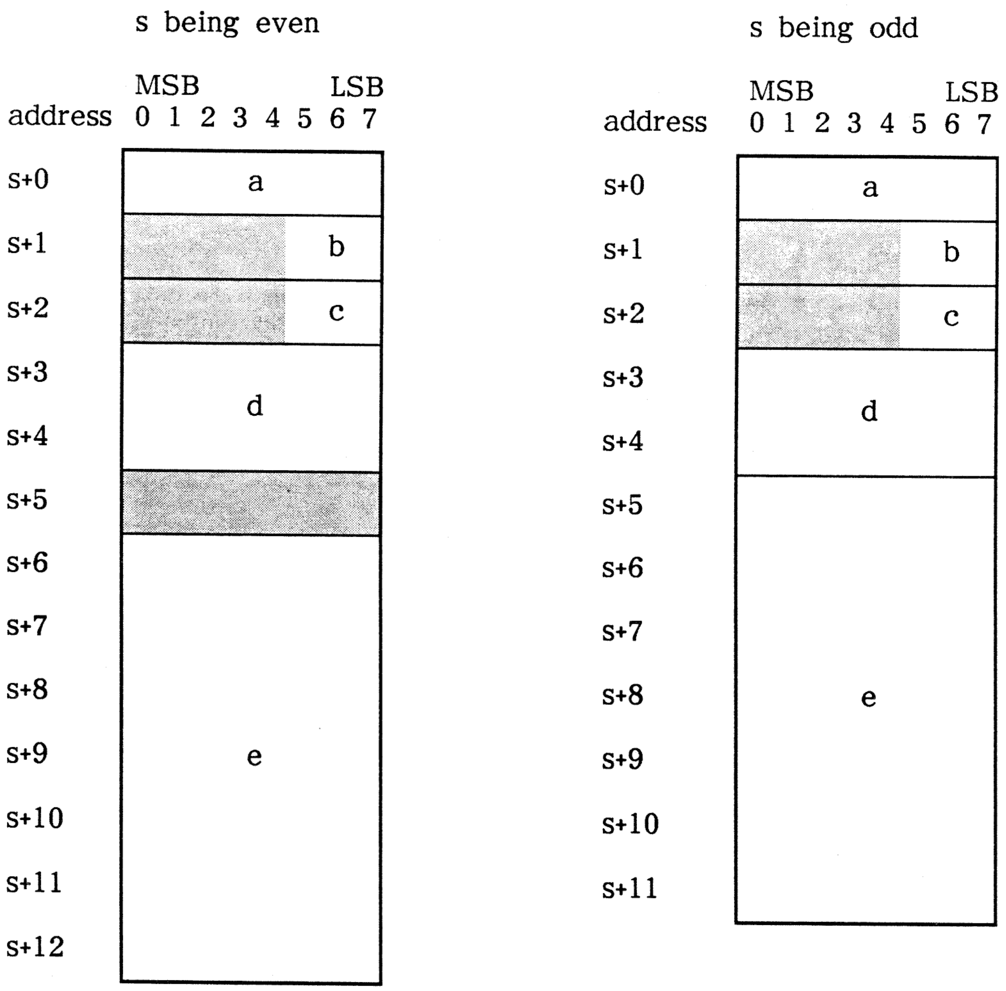

global frame ->
process descriptor
actual parameters
variables declared at the program block level
actual parameters
local frame ->
anonymous parameters
variables declared at the routine block level
operands of expression being evaluated
stack top ->
high address
global frame
stack frames for intermediate routine calls
local frame (for the latest routine call)

Fig. 2.1. Process Stack Layout (snapshot)



The shaded areas are unused.

Fig. 2.2. Memory Layout for Simple Variables in Stack Frame

The layout of actual parameters is similar to that of declared variables, with a minor modification: Since the smallest unit of memory pushed on the evaluation stack is a word, each actual parameter of a program or routine stack frame occupies an integral number of words. Similar to the case of declared variables, an actual parameter of an enumeration type of size less than 16 (bits) is placed in the least significant bits of the allocated word.

2.10.2.3 Structured Types

The memory requirement of a structured type is determined by the memory requirements of the component type(s) and by the necessary alignment.

The memory allocated for a variable of a structured type is sub-allocated for the components of the variable in a fashion which depends on whether the type is declared as packed or not.

For an array, memory is allocated from low to high addresses for the elements in index order, and for a record, memory is allocated from low to high addresses for the fields in the order they are declared in the record type definition.

2.10.2.4 Arrays and Records, Not Packed

For an array or record type which is not packed, memory allocation for the components takes place in precisely the same fashion as allocation of memory for declared variables in a stack frame, i.e.:

- from low to high addresses,
- an integral number of bytes per component,
- alignment (relative to the beginning of the array or record and by implication also in absolute memory) as described in subsection 2.10.2.1.
- right justification of each enumeration type component within the allocated byte or word.

Example: With the record type definition:

```
TYPE
      r=RECORD
          a: char;
          b, c: 0..7;
          d: integer;
          e: reference
        END;
```

Fig. 2.2 -s being even- shows the layout of a variable of type r. The record is word aligned because of the shielded component.

By summation (or multiplication) the memory requirement of an array or record type may be determined from the above rules for sub-allocation of memory for components. For the record type in the example M(r)=13, because the start address is even.

2.10.2.5 Packed Arrays and Records

In the memory allocated for a variable of a packed array or record type several consecutive components may be packed into a single byte or word. Only components of types with size less than 16 (bits) (for arrays: 6 bits) are candidates for packing. Packing of components always starts from bit 0 (MSB) of a byte, and each component is allocated as many bits as indicated by its size. When it is not possible to fit any more components without crossing two byte boundaries, packing stops and allocation is resumed from the next byte boundary, i.e. byte-alignment takes place. By this rule unused space may be left in the least significant bits of the byte, where byte-alignment took place.

Note: only components of ordinal types are candidates for packing.

The memory requirement of a packed record or array type is always at least 1 byte.

Example: The layout of variables of the following packed record type is shown in fig. 2.3.

```
TYPE
     q=PACKED RECORD
          a: char;
          b,c: 0..7;
          d: integer;
          e: reference
        END;
```

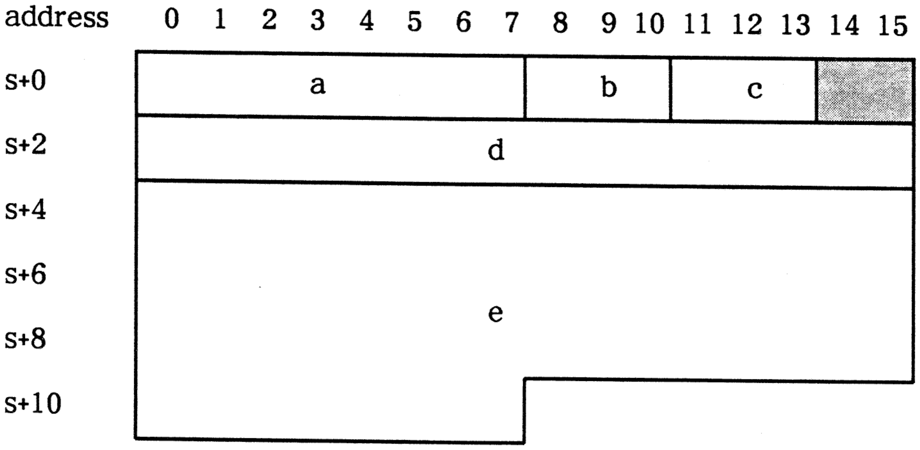


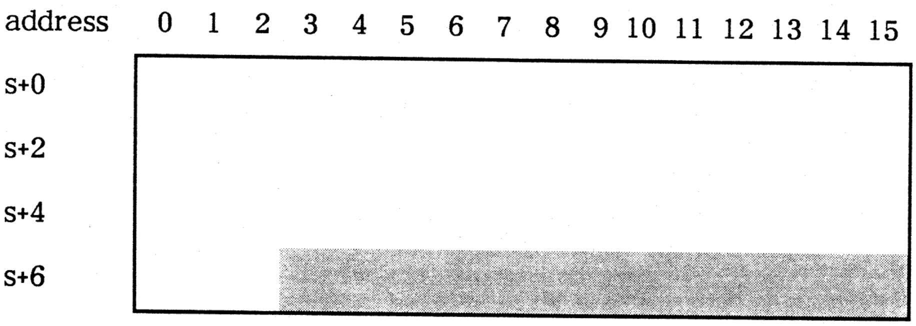
Fig. 2.3. Memory layout for packed record. M(q)=11

2.10.2.6 Set Types

The memory requirement of a set type is always an even number (of bytes), i.e. an integral number of words are allocated for each variable of a set type. The number of words used is the smallest number which will accomodate the bit vector used to represent values of the set type, cf. subsection 2.10.1.3. The bit vector is laid out with index 0 in the most significant bit of the first word. The last word may contain an unused portion in its least significant bit positions. Example:

```
    VAR set_var: SET OF 0..50
```

layout:



13 bits are unused (the shaded area). The memory requirement of the type is 8.

2.11 Evaluation of Expressions

(cf. RTP-4.1)
The short circuit evaluation of logical expressions with the operators OR and AND is not supported. All expressions are completely evaluated.

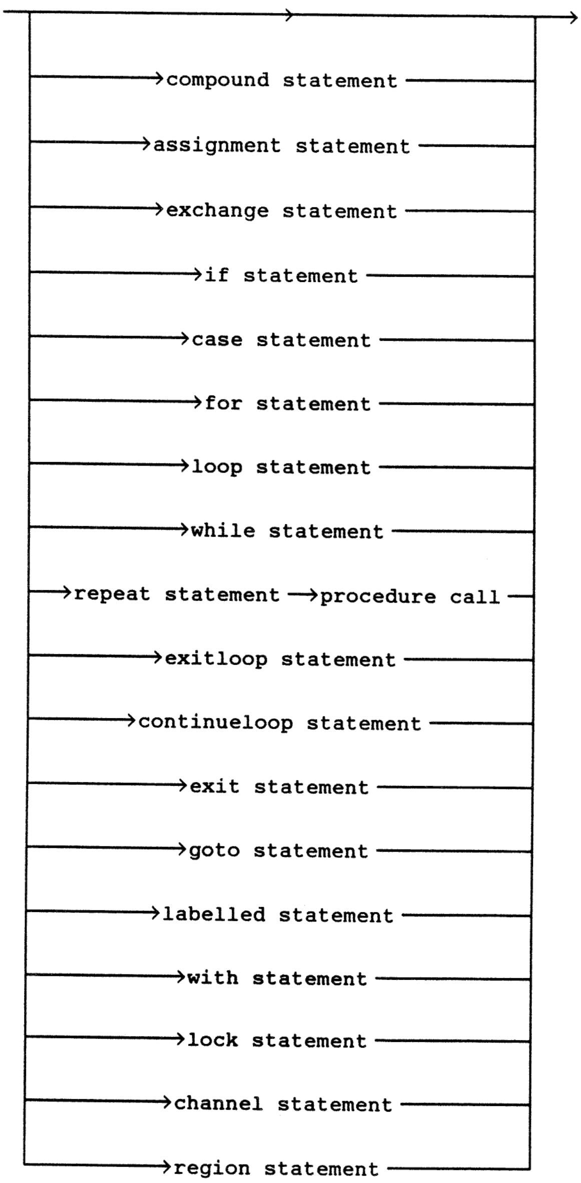
2.12 Constant Expressions

(cf. RTP-4.3)
Constant expressions are further restricted:
The predefined functions abs, chr, ord, pred, and succ are not allowed in a constant expression, not even if applied on a constant argument. Neither are typesize or varsize calls allowed.

2.13 Statement

(cf. RTP-5)

statement:



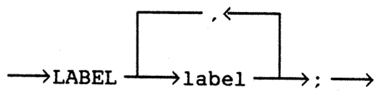
2.13.1 Exchange Statement

(cf. RTP-5.2.2)
In the RC3502 implementation exchange statement is restricted only to accept objects of type reference or of type process. The general exchange is not supported.

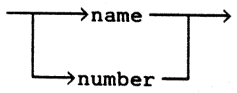
2.13.2 Labels

(cf. RTP-5.7.4)

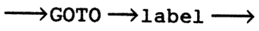
label declaration:



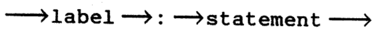
label:



goto statement:



labelled statement:



As in Standard Pascal label declaration parts are allowed among the other declarations. Labels are names or unsigned numbers, not prefixed by a radix specification. It is checked that declared labels are introduced in labelled statements.

2.13.3 Lock Statement

(cf. RTP-5.9)
The implicit types of lock statements are anonymous types. This means that the types are not compatible with other types defined in the program. Access to the data area must be preceded by an explicit typing in the lock statement or/and by means of a with statement with retyping.

If the lockword is LOCKDATA the accessible data area is the part of the message buffer bound by offset and top-1, inclusive. Offset and top are the values of the message attributes.

If the lockword is LOCKBUF the whole message buffer is accessible.

2.13.4 Channel Statement

The channel statement is the language construct which provides access to a device. The statement following do of the channel statement is executed in a special priority class and with a priority depending on the channel_object. For further details cf. Driver Processes in section 4.3.

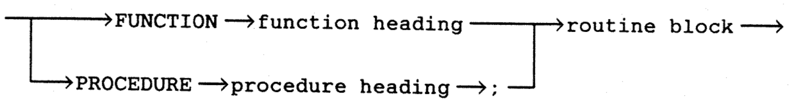
```
channel statement:

 ──→CHANNEL ──→object denotation ──→DO ──→statement ──→
```

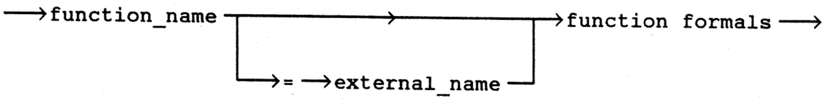
2.14 External Routine Declaration

(cf. RTP-6 and RTP-6.2.2)
If the routine block is specified by the word EXTERNAL it is possible to specify the external name different from the internal name. The external name is used by the linkage editor and the internal name is the one which is used when the routine is called. It is possible to have more bindings, even with different parameter lists, to one external routine. This is caused by a reduced amount of parameter checking between the 'formal parameters' of a separately compiled routine and the corresponding formal parameter specification of the separately compiled block. The implemented parameter checking is mainly based on parameter kind, i.e. VAR, SHARED, INSPECT, or value, and the size of the parameter object, though shielded components must match exactly.

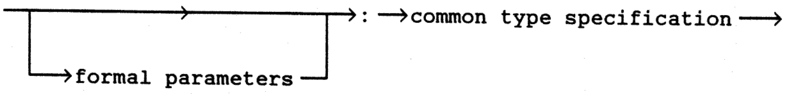
routine declaration:



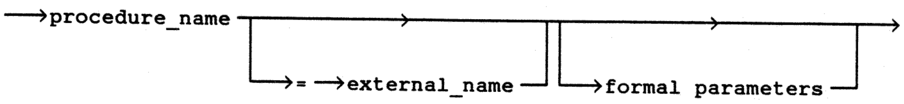
function heading:



function formals:



procedure heading:



Example:

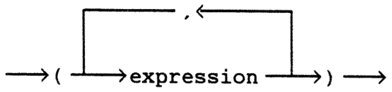
PROCEDURE trace=exception(code: integer); EXTERNAL

trace is the internal name and exception is the name of the external procedure which will be linked to.

2.15 Parameters

(cf. RTP-6.1)
The symbol ? is not allowed as actual parameter specification in the RTP3502 implementation.

actual parameters:



As a further restriction to process parameters, i.e. to the formal parameters which occur in a program heading, parameters of type pool are not supported.

2.16 Compilation Unit

(cf. RTP-8.2)
For backward compatibility reasons, separately compiled routines may be preceded by the reserved word PREFIX and a name, which must be the same as the routine name.

2.17 IMC Functions

(cf. RTP-11)
Notice that the IMC constants, types, and routines are not predefined. They are defined in a context file named imc3502env, which must be specified in the call of the compiler, if one or more IMC related functionalities are used in a program.

2.18 Compiler Directives

(cf. RTP-12)
This section contains a complete description of the supported compiler directives.
Directives to the Real-Time Pascal compiler may be regarded as lexical separators. They have the general form:
$ directive-name parameters end-of-line
given on a separate line. Alternatively directives may be supplied as parameters in the call of the compiler. Some of the directives must be specified before the first line of actual source text is met, viz. either in the compiler call or as $-directive lines in front of the outermost program/routine heading.

The following table lists the available directives. Non standard directives are marked as such beneath the directive name.

name	parameters	description
PAGELENGTH (default 45)	number	maximum number of lines per page of listing,
PAGEWIDTH (default 120)	number	maximum number of characters per line of listing,
EJECT	none	force the start of a new page of the listing
TITLE	"char. string"	the character string is placed in the title field of the header line of each page of the listing,
SUBTITLE	"char. string"	the character string is placed in the subtitle line of each page of the listing,
CODE	none	causes code generated for the following lines of source text to be listed,
NOCODE (default)	none	suppresses listing of generated code,
CREATESIZE	number	determines stack size of incarnations of the following program(s) if the value of the size_expression of the create call is 0, cf. section 3.1.2, (implementation dependent)
INDEXCHECK (blind)	none	causes checking of subrange constraints before indexing to be included in code generated for the following lines of source text. Note: it cannot be turned off because the check operations are part of the machine instructions.
NOINDEXCHECK (blind)	none	switches off index checking, Note: it cannot be turned off because the check operations are part of the machine instructions.
LIST	none	causes the following lines of source text to be listed,
NOLIST (default)	none	suppresses listing of source text,

name	parameters	description
LINENUMBER (non standard)	number	force line number of current line to be the value of the parameter expression,
ROUNDUPLINE (non standard)	none	line number of current line is rounded up to the next multiplum of 1000,
INCLUDE (non standard)	name	continue reading of program text in the specified file. Include may be specified in an included file.
VERSION (non standard)	number	assign the value of the version expression to the version attribute of the program
RANGECHECK (blind)	none	causes checking of subrange constraints before assignment to be included in code generated for the following lines of source text. Note: it cannot be turned off because the check operations are part of the machine instructions.
NORANGECHECK (blind)	none	switches off range checking, Note: it cannot be turned off because the check operations are part of the machine instructions.
ACCESSCHECK (blind)	none	causes code to be generated for every access to a formal parameter object which is not of kind value, to check the existence of the actual parameter (not specified as ?). The symbol ? is not accepted as parameter in RTP3502.
NOACCESSCHECK (blind)	none	switches off access checking.
SET	switch assignment list	see below,
DEFAULT (non standard)	switch assignment list	see below,

name	parameters	description
IF	expression	see below,
ENDIF	none	see below,
ELSE	none	see below,
ELSEIF	expression	see below,

Switches are compile-time variables which can - except from parameters of the transfer function getswitch (cf. RTP-3.12) - only be used in the expressions occurring in IF and ELSEIF directives. Only the first 12 characters of a switch name are significant. A switch assignment list consists of one or more switch assignments separated by commas. A switch assignment has the form (number must be integer, no radix allowed):

    name = number

A switch assignment either introduces and sets the value of a switch, or, if the switch has been introduced in a previous switch assignment (SET or DEFAULT directive or from the call of the compiler), merely changes the value. The directive word DEFAULT is used for introduction and setting the value of a switch. If the switch already existed, as a result of an earlier SET directive, the DEFAULT directive is blind.

The directives IF, ELSE, ELSEIF, and ENDIF provide the capability for conditional compilation. The expressions occurring in IF and ELSEIF directive must be boolean expressions obeying the standard rules of the language. The operands must be switches and integer numbers. The following operators may be used: <, <=, >=, >, =, <>, AND, OR, XOR, NOT.

The directives for conditional compilation may be used to selectively exclude blocks of lines of source text from compilation, i.e. to cause such lines to be treated as comments. When listed, each excluded line will be marked with -- appearing at the beginning of the line.

The directive IF and ENDIF must always be used in matching pairs. ELSE and ELSEIF may optionally be used in conjunction with IF and ENDIF. A use of conditional compilation takes the following form:

$IF expr$_1$

st$_1$

$ELSEIF expr$_2$

st$_2$

$ELSEIF expr$_3$

st$_3$

...

$ELSEIF expr$_n$

st$_n$

$ELSE

st$_{n+1}$

$ENDIF

The only mandatory parts are the IF and ENDIF directive lines and the source line(s) st$_1$. The effect is as follows: If the values of all the expressions are false then the source lines st$_1$, st$_2$, ..., st$_n$ are excluded from compilation. Otherwise let k be the smallest number such that the value of expr$_k$ is true. Then st$_1$, ..., st$_{k-1}$, st$_{k+1}$, ..., st$_n$, and st$_{n+1}$ (if present) are excluded from compilation.

Conditional compilation may be used at several nested levels. In the above terminology any of the st$_i$ may thus include repeated use of conditional compilation.

Notes:
Switches and variables in the program text belong to separate name spaces and cannot be confused. The same names may be used.

Directives occurring in comments are ignored, in particular those occurring in source lines excluded from compilation.

2.18.1 Predefined Switches

There are two predefined switches, target and compiler. In the RC8000 compiler named rtp3502 the value of target is rc3502, and the value of compiler is rtp3502. These switches may be utilized for conditional compilation of machine dependent parts of programs.

3. AVAILABLE ROUTINES

In the following sections all available routines are described. The routines are grouped together according to common functional characteristics:

Routines for Program Control,
Communication Routines,
Routines for Message Manipulation,
Conversion and Arithmetic Routines,
Miscellaneous Routines,
Routines for Operator Communication,
Driver Input/Output Routines.
Routines for backward compatibility.

In the descriptions one out of three attributes is listed:

Predefined: The routine is predefined and may be understood as part of the language.

Not predefined: The routine must be declared in the declaration part of a program or in a user defined environment.

Defined in env_name: The routine is declared in the context file named env_name, which may be used in the compilation for access of the routine. If env_name is rc3502env the declaration is automatically included. The routine is not part of the language definition, but part of the rc3502 specific extension of the language.

3.1 Routines for Program Control

This section describes the routines for control of programs and processes as incarnations of programs. Exception handling is described, and a primitive debug tool (trace).

3.1.1 Break

PROCEDURE break(VAR pr: process; excode: integer);

Defined in rc3502env.

- stops the process and starts it in the exception procedure (see EXCEPTION). If 0<=excode and excode<=break_by_father, the process will go into the exit state; otherwise the process will continue.

3.1.2 Create

```
FUNCTION create(INSPECT processname: alfa;
                program name(actual parameters); VAR proc: process;
                bytes: integer; priority: priotype): create_result;
```

Predefined.

- A new incarnation of the program linked to 'program name' is created. The 'bytes' parameter specifies the amount of storage for holding the runtime stack. The stack is initialized with the actual parameters and various administrative information. The procname field in the stack is initialized to 'processname' and the state to not stopped.

The function returns the following results:

Results	Meaning
create_ok,	Process created
create_process_not_nil	The process variable was not nil
create_program_not_linked	The program was not linked
create_no_memory	No storage or demanded size too small

The 'bytes' parameter is treated as an unsigned integer, allowing a process stack to take up to 32 K-1 words.

If 'bytes' equals zero, a compiler calculated default create size will be used.

Consult appendix B.1 option stack.<appetite> for calculation of the default create size.

Example 1. CREATE

```
PROGRAM example1(VAR mymbx: mailbox);

CONST
  size= 238;
  ok= 0;

VAR
  childmbx: mailbox;
  pr: process;

PROGRAM example11(VAR m1, m2 : mailbox);
BEGIN
    (* body of internal program *)
END;

BEGIN
  IF create('child', example11(mymbx, childmbx), pr, size, stdpriority)=create_ok THEN
    ....

END. (* example1 *)
```

3.1.3 Exception

PROCEDURE exception(excode: integer);

Predefined.

- A declaration of such a procedure causes the runtime system to call this procedure, if an exception occurs.

If the user has not declared an exception procedure, the standard exception procedure will be called. The standard procedure may also be called as the procedure TRACE.

The standard exception procedure activates the exception program which produces output with the format:

```
system clock
process name >> exception, excode=code: error text
gf= ....., top= ...., code= ...
called from: ....., ic= ...., line ...-... , date
     .
     .
     .
```

- 'system clock' is the current value of global date and time
- 'code' and 'error text' are the actual exception code and the meaning of the code; some of the texts include information about the operands which caused the exception.(The possible texts may be seen in appendix M).
- 'gf' and 'top' are stack references, and the number after 'code' is the instruction which caused the exception.
- The list of 'called from...' is the dynamic chain of routine activations.
- 'ic' and 'line ...-...' are an identification of the calls.
- 'date' is the compilation date of the modules in question.

Furthermore, if the program is external linked, printout of date, time, and version of the source is appended.

3.1.4 Link

FUNCTION link(external_name: alfa; program name): link_result;

Predefined.

- The program identified by 'external_name' is searched for in the LINKER catalogue. If found the program identified by 'ex ternal_name' is linked to 'program name'.

The function returns the following results:

Results	Meaning
link_ok	Program linked
link_not_found	Program name with 'external_name' was not found in the LINKER catalogue.
link_no_parameter_match	Program with name 'external_name' is in the LINKER catalogue, but the number of parameters or the type of parameters do no match.
link_already_linked	A program is already linked to program name.

Example 2. LINK

```
PROGRAM example2;

CONST
  def_size= 0;
  ok= 0;
VAR
  mbx: mailbox;
  pr: process;

PROGRAM example21(VAR m: mailbox); EXTERNAL;

BEGIN
  IF link('program21', example21)=link_ok THEN
    IF create('child', example21(mbx), pr, def_size, stdpriority)=create_ok THEN
      ....

END; (* example2 *)
```

3.1.5 Ownname

FUNCTION ownname(VAR name: alfa): byte;

Defined in rc3502env.

- Returns in 'name' the value of the procname field of the process descriptor. The field is initialized by create or an explicit call of setownname.

The function returns the number of characters in the name, apart from trailing blanks.

3.1.6 Ownprogramname

FUNCTION ownprogramname(VAR name: alfa): byte;

Defined in rc3502env.

- returns in 'name' the external name of the least surrounding external program.

The function returns the number of characters in the name, apart from trailing blanks.

3.1.7 Pi3 get

```
FUNCTION pi3_get(program name; pageno : integer;
                 VAR r: reference) : integer;
```

Not predefined.

- Page number 'pageno' is returned in the message in locations specified by the 'offset' and 'top' attributes. 'bytecount' is initialized to the actual number of bytes transferred.

Results	Meaning
0	OK
1	The program is not linked.
2	Illegal kind. The program is not linked by a call of PI3-LINK.
3	Last page. The message contains the last page of the program.
4	Illegal page. 'Pageno' is outside the range 1..lastpage.

3.1.8 Pi3 link

```
FUNCTION pi3_link(external_name : alfa; program name): integer;
```

Not predefined.

- The program of kind DATA identified by 'externalname' is searched for in the LINKER catalogue. If found, the program identified by 'externalname' is linked to 'program name'.

The function returns identical results as the predefined function LINK.

3.1.9 Remove

```
PROCEDURE remove(VAR pr: process);
```

Predefined.

- 'Remove' terminates execution of the process, which cannot be resumed. 'Remove' removes the whole sub-tree controlled by the process. All the resources of the process are deallocated.

The home and return mailboxes of messages in all declared pool variables are initialized to a deallocation mailbox controlled by the ALLOCATOR. All messages in reference, mailbox and pool variables are returned with u2=1 (not processed).

The process variable must refer to a program incarnation (process), otherwise an exception occurs.

After the call the process variable is nil.

3.1.10 Resume

PROCEDURE resume(VAR pr: process);

Predefined.

- if the child process is stopped a call of resume makes it not stopped; otherwise the call has no effect.

An exception occurs, if the process variable is nil.

3.1.11 Setownname

PROCEDURE setownname(INSPECT name: alfa);

Defined in rc3502env.

- Changes the value of the procname field of the process descriptor to the value of 'name'.

3.1.12 Setpriority

PROCEDURE setpriority(priority : integer);

Defined in rc3502env.

- changes the priority of the calling process inside the priority classes II and III running on interruption level 0. If the value of priority is outside the range minpriority..maxpriority the priority is rounded to the nearest priority inside the range.

An exception occurs if:

- called in a channel statement

3.1.13 Start

PROCEDURE start(VAR pr: process; priority: integer);

Predefined.

- The priority of the child process is changed to the value of priority, and if the child process is stopped a call of start makes it not stopped.

If 'priority'=0, the process enters the coroutine class (class II). If 'priority'<0, the child enters the time slice class (class III).

If the activation is actually a reactivation of the process, the process could have been waiting at a mailbox. In that case the WAIT statement will be repeated.

If the process was stopped at an interruption level greater than zero, the process is scheduled directly to the old interruption level and activated as if a timeout has occurred.

An exception occurs, if the process variable is nil.

3.1.14 Stop

PROCEDURE stop(VAR pr: process);

Predefined.

- stops a process. The associated subtree - if any - is not stopped.

If the process is already stopped, the procedure is dummy.

The child process is deactivated and possibly removed from the mailbox where the process is waiting.

If the child process is waiting for interrupt on an interruption level greater than zero, the process is removed from the interruption level.

If the process variable is nil, an exception occurs.

3.1.15 Trace

PROCEDURE trace(excode : integer);

Predefined.

- calls the standard exception routine and generates identical output on the console as the equivalent exception. The program always continues execution after the call, so 'trace' may be used for testoutput generation.

3.1.16 Unlink

FUNCTION unlink(program name): integer;

Predefined.

- The link to the program linked to program name is deleted, if there exists a link and no incarnations of the program exist.

The function returns the following results:

Results	Meaning
unlink_ok	Program unlinked successfully.
unlink_no_program_linked	No program was linked to program name.
unlink_program_busy	Incarnations of the program are existing.

Example 3. UNLINK

```
PROGRAM example3;

CONST
  def_size= 0;
  ok= 0;

VAR
  mbx: mailbox;
  pr: process;

PROGRAM example31(VAR m: mailbox); EXTERNAL;

BEGIN
  IF link('program31', example31)=link_ok THEN
  BEGIN
    IF create('child', example31(mbx), pr, def_size, stdpriority)=create_ok THEN
    BEGIN
      remove(pr);
      IF unlink(example31)=unlink_ok THEN;
    END;
  END;
END; (* example3 *)
```

3.2 Communications Routines

These routines are used for communication between processes, either as signal-wait communication or by means of IMC services. IMC stands for Inter Module Communication and is described in /DSA-IMC/ and /RTP/. The state of mailbox, pool and reference variables may be tested, etc. Attention should be paid to the fact, that the routines alloc and release in section 3.3 also may infer communications equivalent to wait, signal, or return.

3.2.1 Callremote

```
FUNCTION callremote(VAR r: reference; INSPECT m_name: alfa):
                integer;
```

Defined in rc3502env.

- signals the message designated by r to a catalogued mailbox with the specified name. Before return from the routine a call of wait is performed.

Results	Meaning
0	OK, the mailbox was found and the remote call terminated successfully.
1	The mailbox with the specified name was not found.
2	No resources, the call was incomplete because of lack of memory resources to perform the remote call.

3.2.2 Closeport

```
PROCEDURE closeport(VAR p: port);
```

Defined in imc3502env.

- Withdraws a port from the IMC network. If the port is already closed the call has no effect. Otherwise all connections on the port are removed with graceful completion of any feasible data transfers. Then all general receive messages are returned as dummy events, and finally the port_closed event occurs with reason_ok. The port is made unknown to the IMC network, and its state changes to closed.

3.2.3 Connect

```
PROCEDURE connect(VAR p: port; index: 1..maxint;
                  VAR compl, disc: reference;
                  INSPECT remote_name: alfa; service: conn_service);
```

Defined in imc3502env.

- Establishes a connection between a local connection end-point and a remote end-point, which has been made available by a call of getconnection. If the port is closed or closing the message designated by compl is returned as a dummy event and the message designated by disc as a disconnected event with reason_closed. Otherwise the call is only accepted if the state of the indicated end-point is free. The value of remote_name is the name of the remote port to which a connection is requested. The value of the parameter compl may be NIL, if it is present it will be used to generate a local_connect event when the actual connection has been successfully established. The message designated by disc is the disconnected event message. It will be outstanding until the state of the connection end-point eventually reverts to free or the port is closed. If the connection cannot be established the reason will be either reason_name, if a port with the specified name could not be found, or reason_resource, if the necessary resources were not available in the IMC network or at the remote port.

3.2.4 Definetimer

PROCEDURE definetimer(onoff: boolean);

Defined in rc3502env.

- 'definetimer(true)' includes the process in a chain, where count down of the timer field in the process descriptor is done once per second. The call 'definetimer(false)' removes the process from the chain, and count down is stopped.

Timeout can only take place after the call 'definetimer(true)'. At most one call of definetimer can be processed each 50 msec.

3.2.5 Delay

PROCEDURE delay(msecs: integer);

Predefined.

- waits for the expiration of a delay period.

The timer field of the process descriptor is initialized to the value of 'msecs' DIV 1000, and the wait is performed.

Note: Count down of the timer only takes place if the routine 'definetimer' has been called.

3.2.6 Deletemailbox

FUNCTION deletemailbox(INSPECT m_name: alfa): integer;

Defined in rc3502env.

- Searches the mailbox catalogue of the calling process, and deletes the entry, if found.

Results	Meaning
0	OK
1	Not found. No mailbox is catalogued under the specified name.

3.2.7 Disconnect

PROCEDURE disconnect(VAR p: port; index: 1..maxint);

Defined in imc3502env.

- The connection end-point (p, index) is freed. If the indicated connection is absent or if the state of the connection end-point is disconnecting the call has no effect. Otherwise if the connection end-point is engaged in a connection, i.e. if its state is connected or resetting, the connection will be gracefully removed. All data transfers for which credit is available are completed before the remaining outstanding messages are returned. At both ends of the connection all outstanding messages except the disconnected event messages are then returned as dummy events. While this takes place the state of the end-point is disconnecting. Finally the disconnected events occur with reason_ok, and the state becomes free.

3.2.8 Getconnection

```
PROCEDURE getconnection(VAR p: port; index: 1..maxint;
              VAR compl, disc: reference);
```

Defined in imc3502env.

- The connection end-point is made available for remotely initiated connection establishment. If the port is closed or closing the message designated by compl is returned as a dummy event and the message designated by disc as a disconnected event with reason_closed, and the call has no further effect. Otherwise a call is only permitted if the state of the end-point is free. After the call it is remote_accept. The message designated by compl will be returned as a remote_connect event when a connection has been established. The message designated by disc will be the disconnected event message of the connection end-point. It will be outstanding until the end-point becomes free again.

3.2.9 Getcredit

```
PROCEDURE getcredit(VAR p: port; index: 1..maxint;
              VAR credmes: reference);
```

Defined in imc3502env.

- If the general flow control feature is enabled for the port (cf. openport) the designated message becomes an outstanding credit message. It is returned as a credit event when there is one or more outstanding receive messages at the remote end of the connection for which credit has not been given previously. The number of such receive messages is passed as the message attribute credit count. However, a credit event can only occur provided there is at least one outstanding reset indication message at the connection end-point, cf. getreset.

3.2.10 Getreset

```
PROCEDURE getreset(VAR p: portindex: 1..maxint;
                   VAR indic: reference);
```

Defined in imc3502env.

- The message designated by indic becomes an outstanding reset_indication event message. That is, when a reset occurs at the remote end of the connection causing loss of credit this message is returned as a reset_indication with credit count equal to the number of credits that have been taken back. At least one reset_indication message must be outstanding before a credit event can occur, cf. getcredit. Therefore a call of getreset may trigger a credit event.

3.2.11 Getnames

```
FUNCTION getnames(VAR r: reference; imc_id: 0..maxint): integer;
```

Not predefined.

- Entries from the IMC Endpoint Port table identified by imc_id are returned as components in the data area of the designated message. The actual number of components are returned as the function result. The type of the component is:

```
RECORD
  name: alfa;
  scope: byte;
  state: byte;
  alias: integer;
  chainx: integer;
  next: integer;
  path: integer;
  sub: byte;
END
```

3.2.12 Hometest

```
FUNCTION hometest(VAR r: reference; VAR p: pool): boolean;
```

Predefined.

- returns the value true, if the message referenced by 'r' originates from the pool 'p', otherwise false.

An exception occurs if the reference variable is nil.

3.2.13 Imcexists

FUNCTION imcexists(imc_id: 0..maxint): boolean;

Defined in imc3502env.

- Returns the value true if the IMC net identified by imc_id is available, otherwise false. By convention IMC net 0 is the degenerated IMC net with no physical network. IMC net 1 is the network accessible via LAN controller number one, IMC net 2 is the network accessible via LAN controller number two, etc.

3.2.14 Initport

PROCEDURE initport(VAR p: port; imc_id: 0..maxint);

Defined in imc3502env.

- Initializes the port for communications via the IMC net identified by imc_id. The procedure is dummy if imc_id is greater than the constant max_imc_id.

The port must be closed, otherwise an exception occurs.

3.2.15 Locked

FUNCTION locked(VAR mbx: mailbox): boolean;

Predefined.

- returns the value true, if the mailbox is locked (cf. RTP-9.2.1), otherwise false.

3.2.16 Locktest

FUNCTION locktest(VAR r: reference): boolean;

Not predefined.

- returns the value true if the reference variable is locked, otherwise false.

3.2.17 Masknames

```
PROCEDURE masknames(VAR r: reference; VAR mask: !alfa3;
              imc_id: 0..maxint);
```

Defined in imc3502env.

- Ports with name attribute equal to mask as the last three characters are returned as components of type masknames_t in the data area of the designated message. The actual number of components may be calculated from bytecount. The components are ordered in alphabetic order. Only ports on the IMC net identified by imc_id are returned.

3.2.18 Namemailbox

```
FUNCTION namemailbox(VAR m: mailbox; INSPECT m_name: alfa):
             integer;
```

Defined in rc3502env.

- Catalogues the mailbox under the specified name.

Results	Meaning
0	OK, the mailbox is catalogued under the specified name.
1	Overlap, a mailbox is already catalogued under the specified name.
2	No resources, the mailbox cannot be catalogued because of lack of memory resources.

3.2.19 Nil

```
FUNCTION nil(VAR p: niltype): boolean;
```

Predefined.

- returns the value true, if the parameter p is nil, otherwise false. 'Niltype' may be the types reference, process, or any pointer type.

3.2.20 Open

```
FUNCTION open(VAR mbx: mailbox): boolean;
```

Predefined.

- returns the value true, if the mailbox is open (cf. RTP-9.2.1), otherwise false.

3.2.21 Openport

```
PROCEDURE openport(VAR p: port; VAR mes: reference;
                   INSPECT name: alfa; scope: scope_type;
                   no_of_conns: 0..maxint; cntrl: control_type);
```

Defined in imc3502env.

- The port is opened with the attributes set according to the parameter values. The parameter mes designates the port_closed event message which will be outstanding while the port is open. An exception occurs if the number of requested connection endpoints is greater than the maximum allowed (cf. maxconnections), or if the port is not closed before the call. The general receive feature and the general flow control feature is enabled or disabled according to the value of the cntrl parameter (cf. RTP-11).

3.2.22 Passive

```
FUNCTION passive(VAR mbx: mailbox): boolean;
```

Predefined.

- returns the value true, if the mailbox is passive (cf. RTP-9.2.1), otherwise false.

3.2.23 Receive

```
PROCEDURE receive(VAR p: port; index: 1..maxint;
                  VAR mes: reference);
```

Defined in imc3502env.

- The designated message is made an outstanding receive message for the specified connection (p, index). When a unit of data has been transferred from a remote send message to the receive message the latter is returned as a data_arrived or data_overrun event, depending on whether the received data unit had to be truncated to fit into the receive data area.

3.2.24 Receiveall

```
PROCEDURE receiveall(VAR p: port; VAR mes: reference);
```

Defined in imc3502env.

- If the general receive feature is enabled for the designated port (cf. the cntrl parameter of openport) the message designated by mes is delivered to the IMC as a general receive message for the whole port, not dedicated to a particular connection. If the port is closed or closing the message is returned as a dummy event. If the general receive feature is not enabled an exception occurs. The message will eventually be returned as data_arrived or data_overrun, depending on whether the received data unit had to be truncated to fit into the receive data area. The index of the connection end-point to which the data belong will be an attribute of the message.

3.2.25 Reset

PROCEDURE reset(VAR p: port; index: 1..maxint; VAR mes: reference);

Defined in imc3502env.

- All data and credit messages which are outstanding at an end-point of the specified connection (p, index) are taken back. All data transfers for which credit is available are carried out first. Then the remaining data and credit messages are returned as dummy events. Following a call of reset the state of the connection end-point will be resetting. The message designated by mes is the reset_completion event message. This event occurs after all outstanding data and credit messages have been returned. When it occurs the state of the end-point reverts to connected.

3.2.26 Resetevent

PROCEDURE resetevent(VAR r: reference);

Defined in imc3502env.

- If the value of r is NIL an exception occurs, otherwise the value of r's event kind attribute becomes not_event.

3.2.27 Return

PROCEDURE return(VAR r: reference);

Predefined.

- signals the message referenced by 'r' to the anonymous answer mailbox.

An exception occurs if:

- the reference variable is nil.
- the reference variable is locked.

3.2.28 Searchmailbox

FUNCTION searchmailbox(INSPECT m_name: alfa): ↑ mailbox;

Defined in rc3502env.

- Retrieves a pointer value of a catalogued mailbox. The retrieval is directed from the leaves to the root of the process tree, starting at the calling process. A nil value is returned, if no catalogued mailbox with the specified name is found.

3.2.29 Send

PROCEDURE send(VAR p: port; index: 1..maxint; VAR mes: reference);

Defined in imc3502env.

- The data unit designated by mes will be sent on the specified connection (p, index), if present, by the IMC network. The message becomes an outstanding send message. If a receive message is available at the remote end, or when one becomes available (call of receive or receiveall), the indicated data unit is transferred to the receive data area. Then the send message is returned as data_sent, and the receive message at the remote end as data_arrived (or data_overrun).

3.2.30 Sendlinker

PROCEDURE sendlinker(VAR r : reference);

Defined in rc3502env.

- signals the message referenced by 'r' to the LINKER process. No wait is performed in the procedure.

An exception occurs if:

- the reference variable is nil.
- the reference variable is locked.

User Fields	Message	Answer
u1	function	unchanged
u2	unused	result
u3	param	unchanged
u4	unused	unchanged

An unknown function is returned with result = 6.

Result = 6 is also returned, if the message cannot contain a proper result record.

3.2.30.1 Lookup name

User Fields	Message	Answer
u1	3	3
u2	unused	result
u3	unused	unchanged
u4	unused	unchanged

Message buffer
A record of the predefined type 'lookup_descriptor_segment' defined in rc3502env (section I.1).

The LINKER catalogue is searched for a program with name 'name'. Some values of 'name' have special effects, by sending output directly to the OPERATOR mailbox, before returning the request. If the name contains a wild character "*", the whole LINKER catalogue is scanned. Programs are listed if the name satisfies a 'wild character compare', cf. the routine wild_compare. If the name is 'PROGRAM' all programs of kind program are listed. If the name is 'PROCEDURE' or 'FUNCTION', all programs of kind procedure or function are listed.

Results	Meaning
0	OK. A program with name 'name' is found in the LINKER catalogue. In the answer, the remaining fields of the type 'lookup_descriptor_segment' are initialized.
1	No program with name 'name' is found in the LINKER catalogue.

3.2.30.2 Check

User Fields	Message	Answer
u1	7	7
u2	unused	result
u3	module	module
u4	unused	unchanged

Message buffer
A buffer of size 128 bytes (minimum).

A crc16 check is performed on the module.

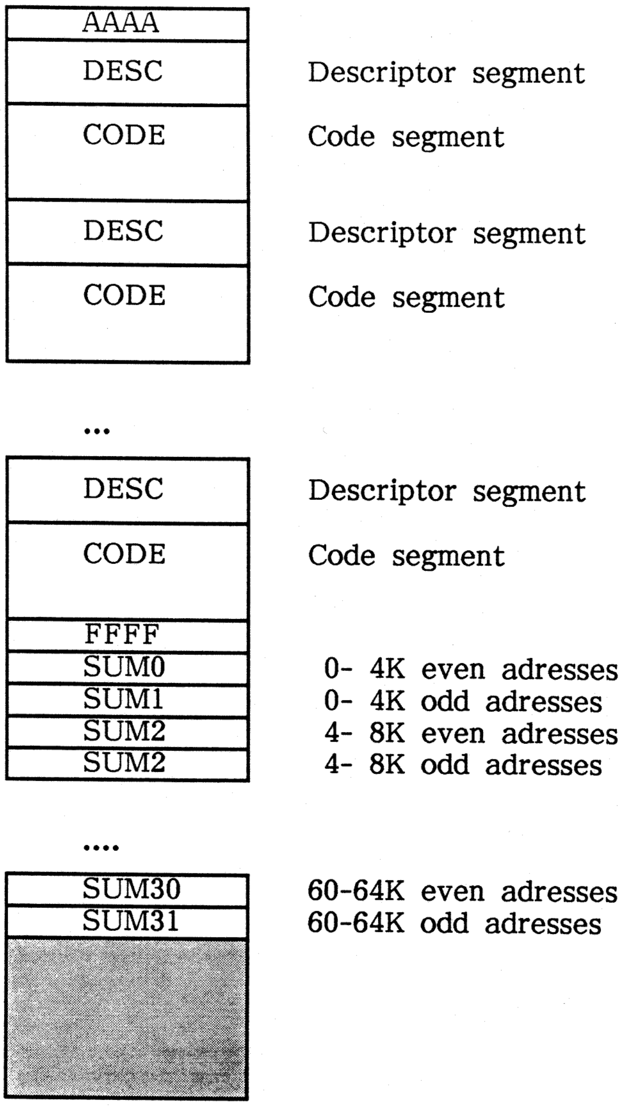
Note: The size of the message must be at least 128 bytes.

The polynomium used is:

$$x^{16}+x^{12}+x^{5}+1$$

with remainder = -1 as initial value.

The module must have the format:



The sums include the 'AAAA' and 'FFFF' words. The sum words themselves are not included.

The sum check in the module and the computed sum are delivered in the message as:

```
ARRAY (1..32) OF
   RECORD
      promsum,
      expected: integer;
   END;
```

Results	Meaning
0	The check sum values are returned in the data buffer.
1	The module is empty.

3.2.31 Sendtimer

PROCEDURE sendtimer(VAR r : reference);

Defined in rc3502env.

- signals the message referenced by 'r' to the TIMER process. No wait is performed in the procedure.

An exception occurs if:

- the reference variable is nil
- the reference variable is locked

If the buffer cannot contain a record of the predefined type 'delaytype' (section I.1) when demanded, the message is refused immediately with u2=4 (unintelligible). Undefined functions are processed as a get clock message.

The subrecord consisting of the fields 'prev_date', 'prev_time', and 'prev_secs' is called 'buffer time'.

3.2.31.1 Short Delay Message

User Fields	Message	Answer
u1	5 (1+4*1)	5
u2	delay1 (<>0)	result
u3	delay2	0
u4	unused	unchanged

Message buffer
Not used.

Function
Returns the message after delay1*2↑delay2 msec. The values of 'delay1' and 'delay2' must fulfil the relation delay1*2↑delay2<=65535.

Results	Meaning
0	OK.
1	Not processed. The message is regretted.

3.2.31.2 Long Absolute Delay Message

User Fields	Message	Answer
u1	13 (1+4*3)	13
u2	unused	result
u3	unused	0
u4	unused	unchanged

Message buffer

The buffer is supposed to hold a record of type 'delaytype' (section I.1).

Function
The buffer time is set to 'buffer time + inc'. If the new buffer time is before global time, the buffer time is set equal to global time and the message is returned immediately, otherwise the message is returned when global time passes the new buffer time.

Results	Meaning
0	OK.
1	Not processed. The message is regretted.

3.2.31.3 Set Clock Absolute Message

User Fields	Message	Answer
u1	2 (2+4*0)	2
u2	unused	0
u3	unused	0
u4	unused	unchanged

Message buffer
The buffer is supposed to hold a record of type 'delaytype'.

Function
Global time is assigned from the buffer time. ('inc' is not used). The queue of long delay messages is scanned for eventual returns.

The message is returned immediately.

3.2.31.4 Get Clock Message

User Fields	Message	Answer
u1	1 (1+4*0)	1
u2	unused	0
u3	unused	0
u4	unused	unchanged

Message buffer
A record of type 'delaytype'.

Function
Global time is assigned to the buffer time. ('inc' is not used).

The message is returned immediately.

3.2.31.5 Long Relative Delay Message

User Fields	Message	Answer
u1	9 (1+4*2)	9
u2	unused	result
u3	unused	0
u4	unused	unchanged

Message buffer
A record of type 'delaytype'.

Function
The buffer time is set to 'global time+inc'. The message is returned, when global time passes the new value of the buffer time.

Results	Meaning
0	OK.
1	Not processed. The message is regretted.

3.2.31.6 Set Clock Relative Message

User Fields	Messag	Answer
u1	6 (2+4*1)	6
u2	unused	0
u3	unused	0
u4	unused	unchanged

Message buffer
A record of type 'delaytype'.

Function
Global time is set to 'global time+inc', and the buffer time is set to this new value of global time. The message is returned immediately. The queue of long delay messages is scanned for possible returns.

3.2.31.7 Set Clock Interval Message

User Fields	Message	Answer
u1	4 (0+4*1)	4
u2	unused	0
u3	new interval	old interval
u4	unused	unchanged

Message buffer
Not used.

Function
Controls the speed of global time. For every Real Time Clock interrupt, global time is incremented by 'global interval' msec. Default = 50 msec. Only long delays are influenced by redefinitions of 'global

interval'. Short delays and the RTC interrupt frequency are not disturbed.

New interval
'Global interval' is set to 'new interval' (msec.).

Old interval
'Global interval' is returned, before set to 'new interval' for reestablishment purposes.

3.2.31.8 Regret Message

User Fields	Message	Answer
u1	12 (0+4*3)	12
u2	unused	0
u3	unused	unchanged
u4	unused	unchanged

Message buffer
Not used.

Function
The delay queues are scanned. If a message is found with matching 'home mailbox', it is returned with result=1 (not processed). The regret message is returned immediately. At most one message is regretted per 'regret message'.

3.2.31.9 Regret Stream Message

User Fields	Message	Answer
u1	8 (0+4*2)	8
u2	unused	0
u3	unused	unchanged
u4	stream no	unchanged

Message buffer
Not used.

Function
The delay queues are scanned. If a message is found with matching 'home mailbox' and stream number (u4 value), it is returned with result=1 (not processed). The regret message is returned immediately. At most one message is regretted per 'regret message'.

3.2.32 Signal

PROCEDURE signal(VAR r: reference; VAR mbx: mailbox);

Predefined.

- If the mailbox is passive or open, the message referred to by 'r' becomes the last element of the mailbox's sequence of messages. If the mailbox is locked, the message is handed over to the first process waiting on the mailbox, and this process is activated.

An exception occurs if:

- the reference variable is nil.
- the reference variable is locked.

3.2.33 Wait

PROCEDURE wait(VAR r: reference; VAR m: mailbox);

Predefined.

- If the mailbox is open, the first message is removed from the mailbox's sequence of messages. If the mailbox is passive or locked, the process waits and becomes the last element of the sequence of processes waiting on the mailbox. It is resumed by another process calling signal or return.

The reference parameter must be nil, otherwise an exception occurs. After a call of 'wait' it refers to a message.

3.2.34 Waitdelay

```
FUNCTION waitdelay(VAR r: reference; VAR m: mailbox;
                   msecs: integer): activation;
```

Predefined.

- waits for two kinds of event:

    1. The arrival of a message to the mailbox 'm' (a_mailbox)
    2. Expiration of a delay period (a_delay)

The timer field of the process descriptor is initialized to the value of 'msecs' DIV 1000 before the wait is performed.

An exception occurs if the reference 'r' is not nil.

Note: Count down of the timer only takes place, if 'definetimer' has been been called.

3.3 Routines for Message Manipulation

The routines in this section are used for allocation and deallocation of messages, besides general manipulation of message stacks and message attributes, such as the size of the message buffer, the u-attributes, and the message buffer attributes, offset, top, and bytecount. Routines for manipulation of the attributes first, last, and next are described, but observe that they are not part of the language (i.e. not predefined). Furthermore, direct attribute operations on the first three words in the buffer in lock statements should be avoided.

3.3.1 Alloc

PROCEDURE alloc(VAR r: reference; VAR p: pool; VAR m: mailbox);

Predefined.

- If the pool of messages is not empty, one of the messages is removed. If the pool is empty, the incarnation waits until a message is released to the pool by another process incarnation calling release. The answer mailbox of the allocated message becomes 'm'.

The reference variable must be nil, otherwise an exception occurs. After the call it refers to a message.

3.3.2 Allocdelay

```
FUNCTION allocdelay(VAR r: reference; VAR p: pool;
                    VAR m: mailbox; no_of_msecs: 0..maxint): activation;
```

Predefined.

- If the pool of messages is not empty, one of the messages is removed. If the pool is empty, the incarnation waits until a message is released to the pool by another process incarnation calling release or the expiration of the delay period, whichever happens first. If the function result is a_mailbox, the answer mailbox of the allocated message becomes 'm'. If the delay period expires before the process obtains a message, the call will return the result a_delay, and otherwise have no effect..

The reference variable must be nil, otherwise an exception occurs.

3.3.3 Allocpool

FUNCTION allocpool(VAR p: pool; number, bytes: integer): integer;

Predefined.

- supplies the pool variable 'p' with 'number' messages of size 'bytes'. If the value of 'bytes' is odd the size will be the value of 'bytes'+1. The result indicates the actual number of messages obtained from the ALLOCATOR. The 'home' mailbox in the messages is initialized to the anonymous pool mailbox. That means that the messages are returned to the pool variable by a 'release' call.

Example 4. ALLOCPOOL

```
PROGRAM example4;
CONST
  size =37;
VAR
  driver,
  m     : mailbox;
  r     : reference;
  p     : pool;

BEGIN
  IF allocpool(p, 1, size)=0 THEN
    (* no available memory now*)
  ELSE (* one message with bufsize=38 !! allocated *)
    BEGIN
      alloc(r, p, m);
       (* now the answer mailbox of r is m *)
      signal(r, driver);
      wait(r,m);
    END;
END; (* example4 *)   .
```

3.3.4 Bufcount

FUNCTION bufcount(VAR stack: reference): integer;

Predefined.

- The value returned by bufcount is the number of non-empty messages in the designated stack. The function returns the value 0 if the parameter has value NIL.

3.3.5 Bufsize

FUNCTION bufsize(VAR r: reference): integer;
FUNCTION bufsize(VAR c: chain): integer;

Predefined.

- If the 'bufsize' function is called with a parameter with value NIL, an exception occurs. Otherwise the buffer size in bytes of the designated message is returned as result.

3.3.6 Bytecount

FUNCTION bytecount(VAR r: reference): integer;
FUNCTION bytecount(VAR c: chain): integer;

Predefined.

- If the 'bytecount' function is called with a parameter with value NIL or the message is a header message only, an exception occurs. Otherwise the result is the value of the 'bytecount' attribute of the designated message.

NOTE: The 'bytecount' attribute is computed from the values of the 'first' and the 'next' attributes in the buffer. A message with bufsize greater than 0 always holds the 'bytecount' attribute. So the buffer is accessed without locking the reference variable.

3.3.7 Chaindequeue

PROCEDURE chaindequeue(VAR ref: reference; VAR ch: chain);

Predefined.

- The current element of the chain is removed, and ref designates the removed element. The successor of the removed element becomes the new current message. If the start point is removed from a list with more than one element the successor of the removed element becomes the new start point.

An exception occurs if:

- The value of 'ref' is not NIL before the call
- The length of the chain is 0 before the call
- 'ref' is locked before the call
- 'ch' is locked before the call

3.3.8 Chaindown

PROCEDURE chaindown(VAR ch: chain);

Predefined.

- The predecessor of the current element becomes the new current message of the list.

An exception occurs if:

- 'ch' is locked before the call

3.3.9 Chainenqueue

PROCEDURE chainenqueue(VAR ref: reference; VAR ch: chain);

Predefined.

- The stack designated by ref becomes the new predecessor of current message of the list, and the value of ref becomes NIL.

An exception occurs if:

- The value of ref is NIL before the call
- ref is locked before the call

3.3.10 Chainlength

FUNCTION chainlength(VAR ch: chain): integer;

Predefined.
- The length of the chain is returned as result

3.3.11 Chainreset

PROCEDURE chainreset(VAR ch: chain);

Predefined.

- The current message of the list is made the new start point of the list.

An exception occurs if:

- 'ch' is locked before the call

3.3.12 Chainstart

PROCEDURE chainstart(VAR ch: chain);

Predefined.

- The start point of the list becomes the new current message of the list.

An exception occurs if:

- 'ch' is locked before the call

3.3.13 Chainup

PROCEDURE chainup(VAR ch: chain);

Predefined.

- The successor of the current element becomes the new current message of the list.

An exception occurs if:

- 'ch' is locked before the call

3.3.14 Crc16buf

```
FUNCTION crc16buf(VAR r: reference; frombyte, tobyte: integer;
              quotient, startvalue: integer): integer;
```

Defined in rc3502env.

- Calculates crc16 checksum of the message referenced by r from the byte indexed by frombyte up to and including the byte referenced by tobyte. Quotient represents the quotient polynomium. Startvalue is used as the initial value of the calculated chekcsum.

An exception occurs if:

- the reference variable r is nil.
- the bufferattributeof r is empty.
- the bufsize of r is too small.
- ult (tobyte, frombyte).

3.3.15 Creditcount

FUNCTION creditcount(VAR r: reference): 0..maxint;

Defined in imc3502env.

- If the event kind of the designated message is credit, the result is the number of receive messages available at the remote end-point of the connection. If the event kind is reset_indication the result is the number of credits which have been taken back by a call of reset at the remote end-point.

3.3.16 Eventkind

FUNCTION eventkind(VAR r: reference): event_type;

Defined in imc3502env.

- An exception occurs if the reference is NIL. Otherwise the result is the value of the event kind attribute of the designated message. The result not_event indicates the message has been obtained from a pool or its event kind has been reset (cf. resetevent). The result message_event indicates the message has been signalled from a process. The result answer_event indicates the message has been returned by a process. The result process_removed indicates the message has been returned from a process which was removed. The remaining result values indicate IMC events.

3.3.17 Index

FUNCTION index(VAR r: reference): 0..maxint;

Defined in imc3502env.

- The result is the index in the relevant port of the connection end-point from which the designated message was returned as an event. It is defined for messages with event kind local_connect, remote_connect, disconnected, reset_completion, reset_indication, data_sent, data_arrived, data_overrun, credit, and for dummy events. If the event is dummy and there is no applicable index the result will be 0.

An exception occurs if the reference variable is nil.

3.3.18 Offset

FUNCTION offset(VAR r: reference): integer;
FUNCTION offset(VAR c: chain): integer;

Predefined.

- If the 'offset' function is called with a parameter with value NIL or the message is a header message only, an exception occurs. Otherwise the result is the value of the 'offset' attribute of the designated message.

NOTE: The 'offset' attribute is the first word in the buffer, according to the driver conventions /Driver Conv./. A message with bufsize greater than 0 always holds the 'offset' attribute. So the buffer is accessed without locking the reference variable.

3.3.19 Openpool

FUNCTION openpool(VAR p: pool): boolean;

Predefined.

- returns the value true if the pool contains a message, otherwise false.

3.3.20 Pop

PROCEDURE pop(VAR r1, r2: reference);

Predefined.

- The top message header from 'r2' is removed. If the new top message and the old top message refer to the same message buffer, only the message header is removed. If not, the top message buffer is removed also. 'r1' refers to the removed message.

An exceptions occurs if:

- r1 is not nil before call
- r2 is nil before call
- r2 is locked before call

If 'r2' denotes a stack with only one element before the call, the value of 'r2' becomes nil after the call.

3.3.21 Push

PROCEDURE push(VAR r1, r2: reference);

Predefined.

- The header designated by 'r1' becomes the new top header of the stack. After the call, 'r2' designates the new stack. After the call the value of 'r1' is nil.

If the new top message is a header message, the top message buffer of 'r2' remains the same.

The message accessible through 'r2' (possibly nil) is called the stack.

An exceptions occurs if:

- 'r1' is nil before call
- 'r1' is locked before call
- 'r2' is locked before call
- 'r1' designates a stack with more than one element
- 'r1'='r2' before call

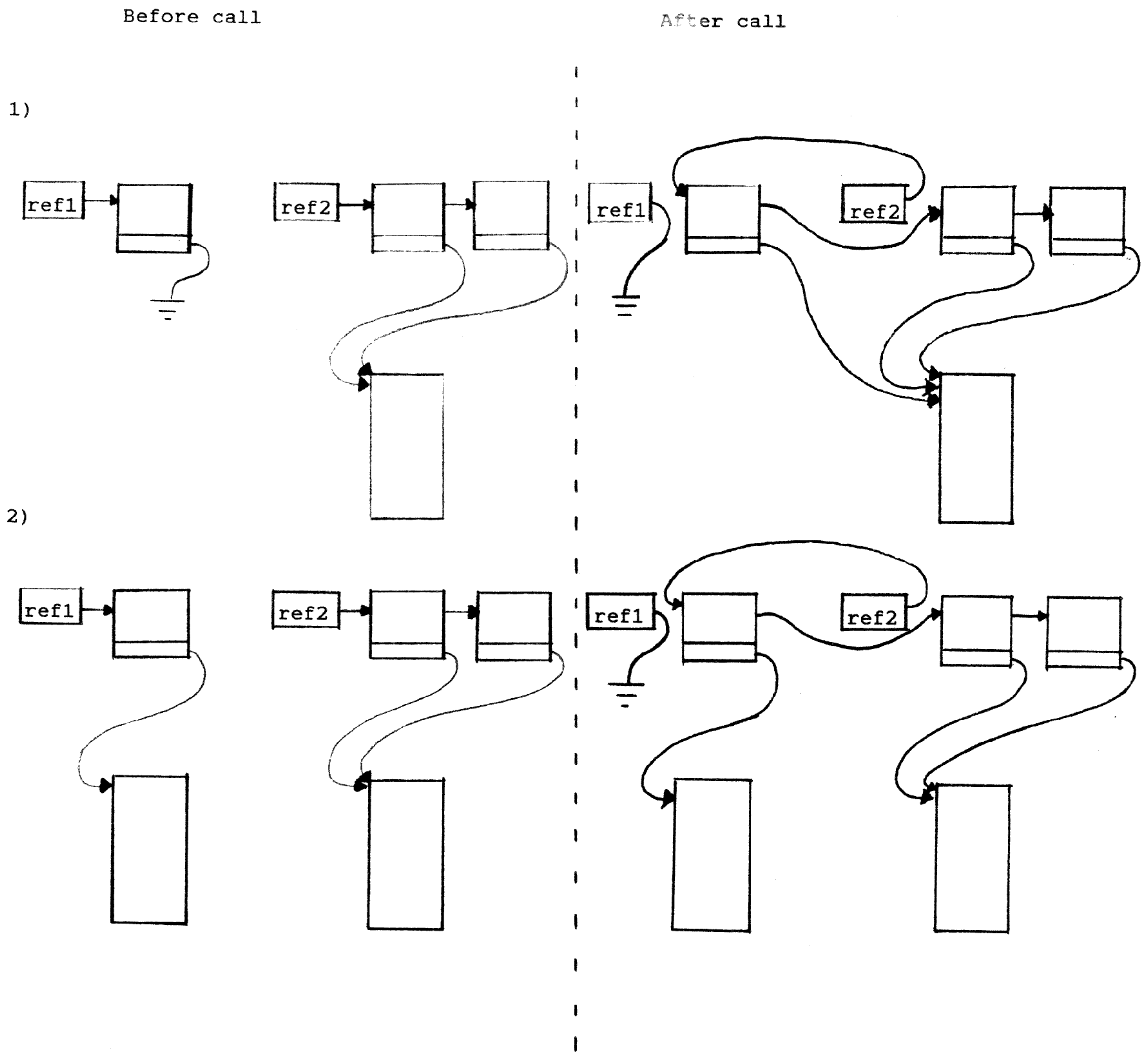


Fig. 3.1. Example on the Behaviour of Push

3.3.22 Reason

FUNCTION reason(VAR r: reference): reason_type;

Defined in imc3502env.

- An exception occurs if the value of the reference is NIL. Otherwise the result is the reason for an event. (Cf. RTP-11).

3.3.23 Release

PROCEDURE release(VAR r: reference);

Predefined.

- signals the message referenced by 'r' to the anonymous home mailbox.

An exception occurs if:
- the reference variable is nil.
- the reference variable is locked.

3.3.24 Releasepool

FUNCTION releasepool(VAR p: pool; number: integer): integer;

Predefined.

- returns a maximum of 'number' messages from the pool 'p' to the ALLOCATOR. The actual number of messages released is returned as the result.

3.3.25 Setbytecount

PROCEDURE setbytecount(VAR r: reference; val: integer);
PROCEDURE setbytecount(VAR c: chain; val: integer);

Predefined.

- If the 'setbytecount' procedure is called with a reference parameter with value NIL, or the message is a header message only, an exception occurs. Otherwise the value of the 'val' parameter is assigned to the 'bytecount' attribute of the designated message.

NOTE: The 'bytecount' attribute is computed from the value of 'next' and 'first', which are in the first part of the buffer, according to the driver conventions /Driver Conv./. A message with bufsize greater than 0 always holds the 'bytecount' attribute. So the buffer is updated without locking the reference variable.

3.3.26 Setoffset

PROCEDURE setoffset(VAR r: reference; val: integer);
PROCEDURE setoffset(VAR c: chain; val: integer);

Predefined.

- If the 'setoffset' procedure is called with a reference parameter with value NIL, or the message is a header message only, an exception occurs. Otherwise the value of the 'val' parameter is assigned to the 'offset' attribute of the designated message.

NOTE: The 'offset' attribute is the first word in the buffer, according to the driver conventions /Driver Conv./. A message with bufsize greater than 0 always holds the 'offset' attribute. So the buffer is updated without locking the reference variable.

3.3.27 Settop

PROCEDURE settop(VAR r: reference; val: integer);
PROCEDURE settop(VAR c: chain; val: integer);

Predefined.

- If the 'settop' procedure is called with a reference parameter with value NIL, or the message is a header message only, an exception occurs. Otherwise the value of the 'val' parameter is assigned to the 'first' attribute of the designated message.

NOTE: The 'top' attribute is computed from the value of the second word in the buffer, according to the driver conventions /Driver Conv./. A message with bufsize greater than 0 always holds the 'top' attribute. So the buffer is updated without locking the reference variable.'

3.3.28 Setu1

PROCEDURE setu1(VAR r: reference; val: byte);
PROCEDURE setu1(VAR c: chain; val: byte);

Predefined.

- If the 'setu1' procedure is called with a reference parameter with value NIL an exception occurs. Otherwise the value of the parameter 'val' is assigned to the u1-attribute of the designated message.

3.3.29 Setu2

PROCEDURE setu2(VAR r: reference; val: byte);
PROCEDURE setu2(VAR c: chain; val: byte);

Predefined.

- If the 'setu2' procedure is called with a reference parameter with value NIL an exception occurs. Otherwise the value of the parameter 'val' is assigned to the u2-attribute of the designated message.

3.3.30 Setu3

```
PROCEDURE setu3(VAR r: reference; val: byte);
PROCEDURE setu3(VAR c: chain; val: byte);
```

Predefined.

- If the 'setu3' procedure is called with a reference parameter with value NIL an exception occurs. Otherwise the value of the parameter 'val' is assigned to the u3-attribute of the designated message.

3.3.31 Setu4

```
PROCEDURE setu4(VAR r: reference; val: byte);
PROCEDURE setu4(VAR c: chain; val: byte);
```

Predefined.

- If the 'setu4' procedure is called with a reference parameter with value NIL an exception occurs. Otherwise the value of the parameter 'val' is assigned to the u4-attribute of the designated message.

3.3.32 Stackdepth

```
FUNCTION stackdepth(VAR stack: reference): integer;
```

Predefined.

The value returned by stackdepth is the number of messages in the designated stack, including empty ones. The function returns the value 0 if the parameter has value NIL.

3.3.33 Tofrom

```
PROCEDURE tofrom(VAR toref: reference; toindex: integer;
                 VAR fromref: reference; fromindex: integer;
                 bytes: integer);
```

Defined in rc3502env.

- Moves 'bytes' bytes from the message designated by 'fromref' starting at the location index 'fromindex' to the message designated by 'toref' starting at the location index 'toindex'. Indexing in the message buffers is according to the driver conventions /Driver Conv./. The 'bytes' and index parameters are considered integers in the range 0..65535. The move is performed as a forward move one byte a time without special action in case of overlapping source and destination areas.

3.3.34 Top

```
FUNCTION top(VAR r: reference): integer;
FUNCTION top(VAR c: chain): integer;
```

Predefined.

- If the 'top' function is called with a parameter with value NIL or the message is a header message only, an exception occurs. Otherwise the result is the value of the 'top' attribute of the designated message.

NOTE: The 'top' attribute is computed from the value of the second word in the buffer, according to the driver conventions /Driver Conv./. A message with bufsize greater than 0 always holds the 'top' attribute. So the buffer is accessed without locking the reference variable.

3.3.35 U1

```
FUNCTION u1(VAR r: reference): byte;
FUNCTION u1(VAR c: chain): byte;
```

Predefined.

- If the 'u1' function is called with a parameter with value NIL, an exception occurs. Otherwise the result is the u1-attribute of the designated message.

3.3.36 U2

```
FUNCTION u2(VAR r: reference): byte;
FUNCTION u2(VAR c: chain): byte;
```

Predefined.

- If the 'u2' function is called with a parameter with value NIL, an exception occurs. Otherwise the result is the u2-attribute of the designated message.

3.3.37 U3

```
FUNCTION u3(VAR r: reference): byte;
FUNCTION u3(VAR c: chain): byte;
```

Predefined.

- If the 'u3' function is called with a parameter with value NIL, an exception occurs. Otherwise the result is the u3-attribute of the designated message.

3.3.38 U4

```
FUNCTION u4(VAR r: reference): byte;
FUNCTION u4(VAR c: chain): byte;
```

Predefined.

- If the 'u4' function is called with a parameter with value NIL, an exception occurs. Otherwise the result is the u4-attribute of the designated message.

3.4 Conversion and Arithmetic Routines

3.4.1 Miscellaneous Routines

3.4.1.1 Chr

```
FUNCTION chr(int: 0..255): char;
```

Predefined.

- returns the character with the ordinal value 'int'.

3.4.1.2 Ord

```
FUNCTION ord(x: otype): integer;
```

Predefined.

- returns the ordinal value of the variable 'x' of type 'otype', where 'otype' is any ordinal type.

3.4.1.3 Pred

```
FUNCTION pred(x: otype): otype;
```

Predefined.

- gives the predecessor of the variable 'x' of type 'otype', where 'otype' is any ordinal type.

3.4.1.4 Succ

FUNCTION succ(x: otype): otype;

Predefined.

- gives the successor of the variable 'x' of type 'otype', where 'otype' is any ordinal type.

3.4.2 Double Arithmetics Routines

Doubleenv (cf. appendix K.1) contains five constant declarations, besides declarations of the following double arithmetics routines:

double_min	= double(:minint, 0:);
double_zero	= double(:0, 0:);
double_one	= double(:0, 1:);
double_two	= double(:0, 2:);
double_max	= double(:maxint, -1:);

3.4.2.1 Double add

FUNCTION double_add(d1, d2: double): double;

Defined in doubleenv.

- Addition in the range -2147483648..2147483647.

An exception occurs, if the result is outside this range.

3.4.2.2 Double dec

PROCEDURE double_dec(VAR d: double);

Defined in doubleenv.

- Decrements the parameter 'd' by one in the range -2147483648..2147483647, but with no overflow exception.

3.4.2.3 Double div

FUNCTION double_div(d1, d2: double): double;

Defined in doubleenv.

- Division in the range -2147483648..2147483647.

An exception occurs, if 'd2'='double_zero'.

3.4.2.4 Double inc

PROCEDURE double_inc(VAR d: double);

Defined in doubleenv.

- Increments the parameter 'd' by one in the range -2147483648..2147483647, but with no overflow exception.

3.4.2.5 Double int

FUNCTION double_int(i: integer): double;

Defined in doubleenv.

- Converts a variable of type integer to a variable of type 'double'.

3.4.2.6 Double lt

FUNCTION double_lt(d1, d2: double): boolean;

Defined in doubleenv.

- 'less than' ('d1'<'d2') in the range -2147483648..2147483647.

3.4.2.7 Double madd

FUNCTION double_madd(d1, d2: double): double;

Defined in doubleenv.

- Addition in the range -2147483648..2147483647, but with no overflow exception.

3.4.2.8 Double mod

FUNCTION double_mod(d1, d2: double): double;

Defined in doubleenv.

- Modulo operation in the range -2147483648..2147483647.

An exception occurs, if 'd2' = 'double_zero'.

3.4.2.9 Double msub

FUNCTION double_msub(d1, d2: double): double;

Defined in doubleenv.

- Subtraction in the range -2147483648..2147483647, but with no overflow exception.

3.4.2.10 Double mul

FUNCTION double_mul(d1, d2: double): double;

Defined in doubleenv.

- Multiplication in the range -2147483648..2147483647.

An exception occurs, if the result is outside this range.

3.4.2.11 Double sub

FUNCTION double_sub(d1, d2: double): double;

Defined in doubleenv.

- Subtraction in the range -2147483648..2147483647.

An exception occurs, if the result is outside this range.

3.4.2.12 Double uint

FUNCTION double_uint(i: integer): double;

Defined in doubleenv.

- Converts a variable of type integer, treated as an unsigned number, to a variable of type 'double'.

3.4.2.13 Int double

FUNCTION int_double(d: double): integer;

Defined in doubleenv.

- Converts a variable of type double to a variable of type integer.

An exception occurs, if the value is outside the range of the type integer.

3.4.2.14 Uint double

FUNCTION uint_double(d: double): integer;

Defined in doubleenv.

- Converts a variable of type double to a variable of type unsigned integer.

An exception occurs, if the value is outside the interval 0..65535.

3.4.3 Miscellaneous Arithmetic Routines

3.4.3.1 Abs

FUNCTION abs(x:integer): integer;

Predefined.

- The absolute value of the variable 'x' is delivered as result.

3.4.3.2 Crc16

FUNCTION crc16(op1, op2 : integer) : integer;

Defined in rc3502env.

- 'op1' represents the polynomium:

  $f(x)=a_{15}*x^{15}+a_{14}*x^{14}+...+a_1*x+a_0$

  where a_j=op1.bit$_j$.

Note: Bit 0 is the *most* significant bit.

- 'op2' represents the polynomium:

  $g(x)=x^{16}+b_{15}*x^{15}+b_{14}*x^{14}+...+b_1*x+b_0$

  where b_j=op2.bit$_j$.

  Note that x^{16} by convention is implicitly given.

  The routine delivers the remainder by the division:

  $f(x)*x^8/g(x)$

Example 5. CRC16

```
PROCEDURE example5;
   (* generate the crc16 remainder *)
   (* to be placed in the last*)
   (* positions of a buffer   *)

CONST
  quotient= -32768+8192+1;   (* x↑16 + x↑15 +x↑2 + 1 *)
  n= 768;

VAR
  remainder: integer;
  buffer: ARRAY (1..n+2) OF byte;

BEGIN
  remainder:= 0; (* initial value *)
  FOR step:= 1 TO n DO
    remainder:= crc16(remainder XOR buffer(step), quotient);
      (* now remainder contains the crc16 remainder *)
      (* the least significant byte of remainder    *)
      (* is supposed to be sent first               *)
  buffer(n+1):= remainder AND 255;
  buffer(n+2):= swap(remainder) AND 255;
END; (* example5 *)
```

3.4.3.3 Dec

PROCEDURE dec(VAR i: integer);
PROCEDURE dec(VAR b: byte);

Predefined.

- decrements the parameter by one with no underflow exception. The procedure is intended for statistical purposes, like i:= i-1, where the access path to the parameter 'i' is hard. The address of the parameter is only calculated once.

3.4.3.4 Inc

PROCEDURE inc(VAR i: integer);
PROCEDURE inc(VAR b: byte);

Predefined.

- increments the parameter by one with no overflow exception. The procedure is intended for statistical purposes, like i:= i+1, where the access path to the parameter 'i' is hard. The address of the parameter is only calculated once.

3.4.3.5 Intel

FUNCTION intel(i: integer): intel_integer;

Not predefined.

- converts an RC3502 integer to an INTEL integer by exchanging the least and most significant byte.

An 'intel_integer' is defined as:

```
intel_integer = RECORD
                   low: byte; (* least significant *)
                   high: byte; (* most significant *)
                END;
```

Example: See example under LAMBDA.

3.4.3.6 Lambda

FUNCTION lambda(ii: intel_integer): integer;

Not predefined.

- converts an INTEL integer to an RC3502 integer by exchanging the least and most significant byte.

An INTEL integer is defined as:

```
      intel_integer = RECORD
                         low: byte; (* least significant *)
                         high: byte; (* most significant *)
                      END;
```

Example 6. INTEL and LAMBDA

```
PROCEDURE example6;

TYPE
  intel_integer = RECORD
                     low, high: byte;
                  END;
FUNCTION intel(i: integer): intel_integer; EXTERNAL;
FUNCTION lambda(ii: intel_integer): integer; EXTERNAL;
VAR
  ii: intel_integer;
  i: integer;

BEGIN
  ii.low:= 15;
  ii.high:= 2;
  i:= lambda(ii); (* i:= 2*256+15 *)
  i:= 259;
  ii:= intel(i); (* ii.low:= 3, ii.high:= 1 *)
END; (* example6 *)
```

3.4.3.7 Madd

FUNCTION madd(a, b: integer): integer;

Not predefined.

- addition, but with no overflow exception.

3.4.3.8 Mmul

FUNCTION mmul(a,b: integer): integer;

Not predefined.

- multiplication, but with no overflow exception.

3.4.3.9 Msub

FUNCTION msub(a,b: integer): integer;

Not predefined.

- Subtraction, but with no overflow exception.

3.4.3.10 Rotate

FUNCTION rotate(a, shifts: integer): integer;

Not predefined.

- makes a cyclic shift of the parameter 'a' 'shifts' times. The result of the rotation is returned by the function. If 'shifts' is positive, the shift is to the left; otherwise the shift is to the right.

3.4.3.11 Swap

FUNCTION swap(i: integer): integer;

Predefined.

- The conversion takes place by swapping the two bytes of the integer i. This effective way of swapping may be used for conversion between integers and the u-fields in message headers.

Example 7. SWAP

```
PROGRAM example7;

VAR
  i, j: integer;
  r: reference;
```

```
    m: mailbox;

  BEGIN
    wait(r, m);
    i:= swap(u2(r)) OR u3(r);   -- i:= u2*256+u3
    setu3(r, swap(j) AND 255);  -- u3:= j DIV 256
    setu4(r, j AND 255);        -- u4:= j MOD 256
  END. (* example7 *)
```

3.4.3.12 Uadd

FUNCTION uadd(a, b: integer): integer;

Not predefined.

- Addition in the range 0..65535.

An exception occurs, if the result is outside the range 0..65535.

3.4.3.13 Udiv

FUNCTION udiv(a, b: integer): integer;

Not predefined.

- Division in the range 0..65535.

An exception occurs, if 'b'=0.

3.4.3.14 Ult

FUNCTION ult(a, b: integer): boolean;

Not predefined.

- 'less than' ('a'<'b') in the range 0..65535.

3.4.3.15 Umod

FUNCTION umod(a, b: integer): integer;

Not predefined.

- Modulo operation in the range 0..65535.

An exception occurs, if 'b'=0.

3.4.3.16 Umul

FUNCTION umul(a, b: integer): integer;

Not predefined.

- Multiplication in the range 0..65535.

An exception occurs, if the result is outside the range 0..65535.

3.4.3.17 Usub

FUNCTION usub(a, b: integer): integer;

Not predefined.

- Subtraction in the range 0..65535.

An exception occurs, if the result is outside the range 0..65535.

3.5 Clock Routines

This section describes a number of routines declared in rc3502env for operations on the types 'clocktype' and 'coded_inc'. These types are also defined in rc3502env.

3.5.1 Clock difference

FUNCTION clock_difference(t1, t2: clocktype): coded_inc;

Defined in rc3502env.

- Returns the difference between two clock values. If the difference exceeds 32 days, the value 31 days, 23 hours, 59 minutes, 59 seconds, and 999 milliseconds is returned.

3.5.2 Clock increment

FUNCTION clock_increment(t: clocktype; inc: coded_inc): clocktype;

Defined in rc3502env.

- Returns the clock value 't+inc'.

3.5.3 Clock less than

FUNCTION clock_less_than(t1, t2: clocktype): boolean;

Defined in rc3502env.

- Returns the value 'true' if 't1' precedes 't2', otherwise 'false'.

3.5.4 Getclock

FUNCTION getclock: clocktype;

Defined in rc3502env.

- Returns the current value of the system clock. The type 'clocktype' is defined in rc3502env (cf. I.1). As opposed to the get clock request, when using the sendtimer procedure, the 'getclock' function works without message communications with the TIMER.

3.6 Miscellaneous Routines

3.6.1 Getid

FUNCTION getid: integer;

Not predefined.

- returns the value of the 16 bit ID register, IDR201. The register is normally installed on RC3502 when used as a node in a communications network.

3.6.2 Getlfgf

PROCEDURE getlfgf(VAR lf, gf: 32bittype);

Not predefined.

- delivers the current values of the stack pointers 'local frame' and 'global frame'. Together with the procedure 'print' it is possible to print specific areas of a stack.

Example 8. GETLFGF

```
PROGRAM example8;
  TYPE
    addr= RECORD
            base, disp: integer;
          END;

  PROCEDURE getlfgf(VAR lf, gf: addr); EXTERNAL;

  PROCEDURE print(VAR z: zone; base: byte; first_disp, last_disp,
                  words_per_line: integer); EXTERNAL;
  VAR
    lf, gf: addr;
    z: zone;

  PROCEDURE exception(code: integer);
  BEGIN
    (* the zone must be initialized for output *)
    getlfgf(lf, gf);
    print(z, gf.base, gf.disp, lf.disp, 1);
  END;

BEGIN
  (* program body *)
END. (* example 8 *)
```

3.6.3 Getswitches

PROCEDURE getswitches(VAR b0, b1 : byte);

Not predefined.

- Returns the current values of the Cswitches controlling the autoload procedure of RC3502.If the first byte of Cswitches equals zero, the contents of the Switches are transferred to Cswitches and returned to the caller. For further information, consult /Operating Guide/.

3.6.4 Maxconnections

FUNCTION maxconnections: 0..maxint;

Defined in imc3502env.

- Returns the maximum number of connections which the IMC network allows to any one port.

3.6.5 Memerrorlog

FUNCTION memerrorlog(baseword: integer): integer;

Not predefined.

- returns an errorlog word from the memory module specified by the parameter 'baseword', which is the base of the first 64K byte RAM module on MEM205. For further details consult "MEM205 General Information" /MEM205/.

3.6.6 New

PROCEDURE new(ptr: ptrtype);

Predefined.

- Memory for a variable of the base type of the type of the parameter 'ptr' is allocated on the heap of the calling process. If an initial value is defined for the variable or any components of it (pointers and shielded), the initialization takes place immediately after allocation. The value of the parameter becomes a pointer to the allocated variable. If the claimed amount of memory is not available, the pointer becomes NIL after the call of new. Memory for the heap is allocated from the machine's pool of free memory, i.e. it is not part of the memory, which was reserved when the process was created. The heap is for dynamic allocation of objects, not necessarily dynamic sized objects. But the amount of heap storage has nothing to do with the reservation of stack for static and dynamic sized objects and routine calls. The heap of a process is returned to the machine's pool of free memory when the process is removed.

3.6.7 Readram

PROCEDURE readram(VAR result: byte; index: integer);

Not predefined.

- returns the value of the byte with index 'index' in the control microprocessor ram.

Relevant indices are:

9:	version
10:	switches 0-7
11:	switches 8-15
42:	COM204 mask
43:	COM204 result

(For further information, refer to /Operating Guide/, appendix K).

3.6.8 Restart

PROCEDURE restart;

Not predefined.

- Performs a remove, create, and start of ADAM. All messages at OPERATOR and TIMER are deallocated. The programs in the LINKER catalogue are not unloaded.

3.6.9 Setswitches

PROCEDURE setswitches(b0, b1: byte);

Not predefined.

- Updates the current values of the Cswitches, controlling the autoload procedure of RC3502. For further information, consult /Operating Guide/.

3.6.10 Setwatchdog

PROCEDURE setwatchdog(i: integer);

Not predefined.

- the "watchdog high" byte in the control microprocessor is initialized to 'i'.

```
'i'= 0 disables the watchdog function.
'i'=   1  equals      640 msec.
'i'= 255  equals 163,200 msec.
```

3.6.11 Wild compare

FUNCTION wild_compare(name, key: alfa): boolean;

Not predefined.

- The value 'key' may contain wild characters, "*", where one wishes to specify zero or more occurrences of "I don't care" characters. E.g.: 'a*b' matches 'ab', 'aab', 'abb', 'acdegfb', and in fact any name that begins with an "a" and ends with a "b".

The value 'true' is returned if this comparison is successful, otherwise 'false'.

3.7 Routines for Operator Communication

This section describes a set of routines which may be used for input/output to the OPERATOR process or another character oriented input/output driver.

Communication takes place via variables of type zone.

The type zone is a simple implementation of the zone concept known from ALGOL8 /ALGOL/ and MUSIL /MUSIL/.

Section 3.7.1 describes the initialization of variables of type zone.

Section 3.7.2 describes the output procedure outchar and a set of output procedures, which use the procedure outchar.

Section 3.7.3 describes the input procedure inchar and a set of input procedures, which use the procedure inchar.

Ioenv contains declarations of the types and routines and is listed in appendix K.3.

Appendix L contains a more comprehensive example of OPERATOR communication.

The routines conform to the driver conventions /Driver Conv./ regarding the use of the user u-fields and the buffer attributes.

3.7.1 Initialization

The type zone is defined in the environment ioenv as:

```
zone= RECORD
        driver,
        answer: ↑ mailbox;
        dataready,
        free: mailbox;
        cur: reference;
        u2val,
        state: byte
        readstate,
        nextp,
        lastpos: integer
      END;
```

'driver' - points to the driver mailbox (e.g. the OPERATOR mailbox).

'answer' - point to the mailbox, where answers arrive from the driver

'dataready' - holds the answers from the driver, if this mailbox is specified as 'answer' in the call of openzone or openopzone. 'dataready' is normally specified, when the zone is used for input from OPERATOR

'free' - holds the empty messages.

'cur' - refers the message which is currently in use for reading input or writing output.

'u2val' - whenever a message is signalled to the driver mailbox, the u2 attribute (result attribute) of the message is initialized to u2val. According to the driver conventions /Driver Conv./, some drivers utilizes the specific convention, that 'u2' never takes the value 7 in answers from a driver, thereby distinguishing messages from application processes and answers from e.g. internal driver interruption processes.

'state' - holds the result attribute (u2) from the message referred by 'cur'. State is updated when a message is taken from 'free' as a new current message for output in the procedure outchar, or when a message is taken from 'dataready' in the procedure opwait as a new current message holding input data.

'readstate' - specifies the state of the zone when used for input after each call of one of the input routines.

Readstate is initialized to -1 after call of 'openzone'. or 'openopzone'.

The following interpretation of readstate holds:

'Readstate'=-1
No input was ready ('cur'='nil') when an input procedure was called or the current input message became empty during call.

'readstate'=0
The call of an input procedure succeeded with no syntax errors.

'readstate'>0
This value range is intended for indication of a syntax error detected during the call of the input procedure. The routines in this manual do not deliver positive values.

'nextp - is the index of the next position in current message for reading or writing.

'lastpos' - is the index of the last position in current message for reading or writing.

3.7.1.1 Openopzone

```
PROCEDURE openopzone(VAR z: zone; driver, answer: ↑ mailbox;
                     bufs: integer; VAR home: pool; v1, v2, v3, v4: byte);
```

Defined in ioenv.

- Messages from the pool 'home' should be able to hold variables of type:

```
      RECORD
         first, last, next: integer;
```

```
          name: alfa;
          chars: ARRAY (firstindex..lastindex) OF char;
        END;
```

'first', 'last', and 'next' are the buffer attributes, according to the driver conventions /Driver Conv./ of the earlier Real-Time Pascal implementations. The values of the attributes are used for computation of the buffer attribute values 'offset', 'top', and 'bytecount'.

A type 'opbuffer' is defined in ioenv with
            lastindex-firstindex+1=80.

The procedure assigns to the field 'name' the value of the process name field of the process descriptor (cf. ownname) in all messages, whereupon it performs as the procedure openzone.

The OPERATOR process identifies the input and output messages by means of the parameter 'name'.

If the actual driver mailbox pointer is nil (an undefined mailbox pointer), the procedure will automatically open the zone with the operator mailbox as driver mailbox.

3.7.1.2 Openzone

```
PROCEDURE openzone(VAR z: zone; driver, answer: ↑ mailbox;
              bufs: integer; VAR home: pool; v1, v2, v3, v4: byte);
```

Defined in ioenv.

Openzone prepares the zone 'z' with messages for input/output.

Note: If OPERATOR communication is wanted, use openopzone.

'z' - the zone which is initialized.

'driver' - a pointer to the driver mailbox, which is assigned to 'z.driver'.

'answer' - a pointer to a mailbox where answers arrive from the driver. 'answer' is assigned to 'z.answer'.

'bufs' - specifies the number of messages which the procedure will allocate the zone z. The messages are placed in 'z.free'.

'home' - the messages are allocated from the pool 'home'. The messages conform to the driver conventions /Driver Conv./ and should be able to hold a variable of type:

```
RECORD
  first, last, next: integer;
  chars: ARRAY (6..max) OF char;
```

```
            END;
```

'v1', 'v2',
'v3', 'v4' - the user attributes of the messages are initialized to 'v1', 'v2', 'v3', and 'v4'. 'v2' is used to reset the result attribute (u2), whenever a message is signalled to the driver.

3.7.2 Output

If the process does not want to be activated, when an output message returns from the driver, it uses a mailbox pointer with value nil as the actual parameter 'answer' in the call of openzone or openopzone. In this way empty output messages return directly to the mailbox 'z.free' as available messages for continued output.

If the program wants to supervise the answers, it uses a mailbox pointer which is not nil in the call of openzone or openopzone. The messages must be signalled to 'z.free' afterwards.

3.7.2.1 Outaddr

PROCEDURE outaddr(VAR z: zone; a: 32bittype);

Not predefined.

- writes the three least significant bytes of the variable a in hexadecimal form with the layout:

  BB.DDDD

The routine is intended for output of RC3502 addresses.

3.7.2.2 Outalfa

PROCEDURE outalfa(VAR z: zone; VAR text: !alfa);

Defined in ioenv.

- writes the variable 'text' by calling outchar. The character '#' acts as a stop character.

3.7.2.3 Outchar

PROCEDURE outchar(VAR z: zone; ch: char);

Defined in ioenv.

- places the character 'ch' in the current message 'z.cur'.

If no current message is available, a wait is performed on the mailbox 'z.free'.

If the message becomes full, the procedure outend is called.

3.7.2.4 Outdate

PROCEDURE outdate(VAR z: zone; date: coded_date);

Defined in ioenv.

- writes the parameter date with the layout:

  YYYY.MM.DD

The type 'coded_date' is defined in rc3502env and is used throughout the run time system, especially the timer system (see description of the procedure sendtimer).

3.7.2.5 Outdouble

PROCEDURE outdouble(VAR z: zone; d: double; pos: integer);

Defined in ioenv.

- writes the double 'd' on decimal form. If the double occupies less than 'pos' positions (including the '-' character, if negative), the double is prefixed a number of spaces to fill the 'pos' positions. If the number occupies more than 'pos' positions, all significant digits are written (inclusive a possible minus sign).

3.7.2.6 Outend

PROCEDURE outend(VAR z: zone);

Defined in ioenv.

- signals he current message 'z.cur' to the driver mailbox 'z.driver↑'.

3.7.2.7 Outfill

PROCEDURE outfill(VAR z: zone; filler: char; rep: integer);

Defined in ioenv.

- repeats the character 'filler', 'rep' times.

If 'rep' is negative, no fill character is written.

3.7.2.8 Outhex

PROCEDURE outhex(VAR z: zone; i, pos: integer);

Defined in ioenv.

- writes the number 'i' in hexadecimal form. If pos is greater than four, pos -4 space characters are prefixed the number. If 'pos' is less than four, the following is valid. If the number occupies more than 'pos' positions, then number is printed as four hex characters. If the number occupies less than 'pos' positions, the number is prefixed a number of zeroes to fill the 'pos' positions.

3.7.2.9 Outinteger

PROCEDURE outinteger(VAR z: zone; i, pos: integer);

Defined in ioenv.

- writes the number 'i' on decimal form. If the number occupies less than 'pos' positions (including the '-' character, if negative), the number is prefixed a number of spaces to fill the 'pos' positions. If the number occupies more than 'pos' positions, all significant digits are written (inclusive a possible minus sign).

3.7.2.10 Outnl

PROCEDURE outnl(VAR z: zone);

Defined in ioenv.

- writes the character nl and signals the message to the driver by calling outend.

3.7.2.11 Outtext

PROCEDURE outtext(VAR z: zone; text: alfa);

Not predefined.

- works as outalfa, which should be used instead of outtext.

3.7.2.12 Outtime

PROCEDURE outtime(VAR z: zone; time: coded_time);

Defined in ioenv.

- writes the parameter time with the layout:

    HH.MM

The type coded_time is defined in rc3502env.

3.7.2.13 Print

```
PROCEDURE print(VAR z: zone; base: byte; first_disp, last_disp,
                words_per_line: integer);
```

Not predefined.

- prints the memory words:

    'base.first_disp' through 'base.last_disp'

with the layout:

```
<address>  {<word hex> <word decimal> <MSB decimal>

                                                          words_per_line
             <LSB decimal> <MSB char> <LSB char>}                    <nl>
                                                          1
```

The procedure terminates if the 'state' field of the zone becomes not OK (<>0).

3.7.2.14 Print descriptor

```
PROCEDURE print_descriptor(VAR z: zone;
                 VAR d: lookup_descriptor_segment);
```

Not predefined.

- prints selected fields from the type lookup_descriptor_segment, which is defined in rc3502env with the layout:

```
<kind> <name> <source date> <object date> <version>
<program size> <no_of_pages> <pagesize> <last_page_length>
<default appetite> <no_of_params> <nl>

 <kind>::= PROGRAM | PROCEDURE | FUNCTION | DATA
```

A record of type lookup_descriptor_segment may be obtained by means of the 'sendlinker(lookupname)' routine. See 3.2.25.

3.7.2.15 Printmessage

```
PROCEDURE printmessage(VAR z: zone; VAR r: reference; firstindex,
               lastindex, words_per_line: integer);
```

Defined in ioenv.

- prints the attributes of the message and the specified data area. The message buffer is supposed to be of type:

```
buffer= ARRAY (0..max) OF byte
```

The printed area is from 'buffer(firstindex)' to 'buffer(lastindex)'.

3.7.3 Input

The input procedures consist of routines for communication with the driver (OPERATOR) process, i.e. opin and opwait, besides a set of routines for converting the contents of an input message to variables of type 'char', 'integer', 'double', or 'alfa'.

If the process wants to be activated only when an input message returns from the driver, it uses a mailbox pointer with value nil as the actual parameter 'answer' in the call of openzone or openopzone.

More commonly, the process is also activated by other events. In that situation, it specifies a mailbox pointer which is not nil as the actual parameter 'answer' in the call of openzone or openopzone and uses opanswer to place the message in the input zone for further reading.

3.7.3.1 Inchar

```
PROCEDURE inchar(VAR z: zone; VAR ch: char);
```

Defined in ioenv.

- the next character from the current message 'z.cur' is read.

If the message becomes empty, it is signalled to 'z.free'.

After the call the variable 'z.readstate' indicates the state of the zone with the interpretation:

z.readstate	
0	Successful reading.
- 1	This message was empty before call. The character nl is returned.

3.7.3.2 Indouble

PROCEDURE indouble(VAR z: zone; VAR d: double);

Defined in ioenv.

- works as ininteger except that reading is in the range -2147483648..2147483647.

3.7.3.3 Inhex

PROCEDURE inhex(VAR z: zone; VAR i: integer);

Defined in ioenv.

- reads a hexadecimal number according to the pseudo syntax:

{<non hex digit>}* {<hex digit>}$^+$

z.readstate	
0	At least one hex digit is read.
- 1	No hex digit was met in the buffer. The value 0 is returned and the buffer is empty

Hex digits are read as long as the number is in the range #h0000..#hFFFF.

3.7.3.4 Ininteger

PROCEDURE ininteger(VAR z: zone; VAR i: integer);

Defined in ioenv.

- reads a decimal number according to the pseudo syntax:

{<nondigit>}* <sign>{<digit>}$^+$

<sign>::= + | - | <empty>

Digits are read as long as the number is in the range -32768..32767.

z.readstate	
0	At least one digit is read.
- 1	No digit was met in the message. The message is empty and the value 0 is returned.

Examples:

```
        Input                     Result(i):
↓         ↓
a b c 1 2 * a b c d e                  12
↓                   ↓
a b c + - - + 1 7 9 a b               179
↓                       ↓
a b c + - 0 0 1 2 3 4 5 6 7        -12345
↓             ↓
a b c 3 2 7 6 8                      3276
```

(↓ indicates the value of nextp before and after the call).

3.7.3.5 Inname

PROCEDURE inname(VAR z: zone; VAR name: alfa);

Defined in ioenv.

- reads a name of maximum 12 characters after the syntax:

{<not letter or _>}* <letter or _> {<letter, digit or _>}

Examples:

```
Input:                   Result:

↓↓
a bc                     a
↓             ↓
     _abc89_6c;          _abc89_6c
↓     ↓
  12ab                   ab
```

The characters are delivered in the parameter 'name' from left to right. 'name' is not initialized by 'inname', so 'name' must be initialized before the call. The variable 'z.nextp' may be used to calculate the number of characters read (inclusive leading blanks).

z.readstate

0	At least one <letter or _> is transferred to 'name'.
- 1	No <letter or _> was met in the buffer. The buffer is empty and possibly a number of spaces was skipped.

3.7.3.6 Inwildname

PROCEDURE inwildname(VAR z: zone; VAR name: alfa);

Not predefined.

- Works as the routine INNAME except that the legal characters are extended with the wild character "*".

3.7.3.7 Opanswer

PROCEDURE opanswer(VAR r: reference; VAR z: zone);

Defined in ioenv.

- signals the message referenced by 'r' to the mailbox 'z.dataready'.

The routine is intended for use, when the process waits at a main mailbox and sorts out all arriving input messages.

3.7.3.8 Opin

PROCEDURE opin(VAR z: zone);

Defined in ioenv.

- signals a message from 'z.free', if any, to the driver mailbox 'z.driver↑'.

3.7.3.9 Optest

FUNCTION optest(VAR z: zone): boolean;

Defined in ioenv.

- is true if a message is queued at 'z.dataready', otherwise false. This may be used to avoid a wait in the procedure opwait.

Example:

```
IF optest(z) THEN
  BEGIN
    opwait(z, inputpool);
      (* process input data from zone z *)
  END
  ELSE
     (* do something else *)
```

3.7.3.10 Opwait

PROCEDURE opwait(VAR z: zone; VAR inputpool: pool);

Defined in ioenv.

- is used to wait for specific input to the zone, which returns directly to 'z.dataready', or to a mainmailbox together with other messages. If a message is queued at 'z.dataready', this message is taken. Otherwise, a wait is performed on 'z.answer '. 'opwait' checks, that an arriving message originates from the 'inputpool' when the zone was opened, and that (u1 MOD 8)=1 (read). Other arriving messages are queued temporarily in the zone until a zone message returns. The queued messages are put back in the mainmailbox and the zone message prepared for later calls of inchar.

3.7.4 Advanced Use

Besides the ordinary use of the OPERATOR to send output and receive input from the console, additional facilities are offered. These services are requested by sending a message to the OPERATOR asking for the wanted facility.

3.7.4.1 Input - Output Mode

Normally OPERATOR sends output to the console and receives input from the console.

On request OPERATOR will send output to
- the console
- a process requesting the output

On request OPERATOR will send input message to (receive input from)
- the console
- a process requesting the input message (and generating input)

There are four i/o modes:

loc_i_glob_o (0):	input from console output to request process
loc_i_loc_o (1):	input from console output to console
glob_i_loc_o (2):	input from request process output to console
glob_i_glob_o (3):	input from request process output to request process

Default mode is 'loc_i_loc_o'.

The mode 'loc_i_loc_o' is kept unchanged until a change mode request is received.

The modes are reset to 'loc_i_loc_o' after 30 seconds, if no process requests the messages. Otherwise, the mode is kept for a new period of 30 seconds.

Section 3.7.4.2 on events describes how a process can use the event mechanism to manipulate the input - output streams.

The change of i/o mode is requested by sending a change message to the OPERATOR mailbox.

	to OPERATOR	from OPERATOR
u1	0	unchanged
u2	-	ok
u3	wanted mode	unchanged
u4	-	unchanged

A process requests the messages in accordance with the actual i/o mode by sending a request message to the OPERATOR

	to OPERATOR	from OPERATOR
u1	12	unchanged
u2	-	result
u3	-	unchanged
u4	-	unchanged

The request message is placed in a queue of waiting request messages.

The waiting input messages and/or output messages in accordance with i/o mode are stacked and the first request message is pushed upon the stack. Finally the stack is returned to the answer mailbox of the request message.

Results	Meaning
0	ok, some messages are returned
1	not_processed, i/o mode was loc_i_loc_o

3.7.4.2 Events

Processes may send some event messages to the OPERATOR.

Event messages are queued until
- they are regretted (see section 3.7.4.3)
- the operator presses 'ESC' and types the 'name' of the event message.

Event messages can be used by processes to control the input-output streams as described in the following example:
A process normally sends output to and receives input from request processes, but sometimes the operator wants to send a command from the console. In this case, the process:

- sends an event message to OPERATOR with the name set to 'command'
- requests 'glob_i_glob_o' mode continuosly.

When the operator presses 'ESC' and types 'command', the event message is returned. Now the process can change i/o mode according to the need, and later continue with the 'glob_i_glob_o' mode.

	to OPERATOR	from OPERATOR
u1	4	unchanged
u2	-	result
u3	-	unchanged
u4	-	unchanged

Message buffer:
The buffer is supposed to hold a record of type 'opbuffer' (appendix K.3). The field 'name' is used to identify the message.

Results	Meaning
0	ok, no input message available when needed.
1	not_processed, the event message is regretted.

3.7.4.3 Regret

A process can regret messages previously sent to OPERATOR by sending a regret message.

	to OPERATOR	from OPERATOR
u1	8	unchanged
u2	-	0 (ok)
u3	-	unchanged
u4	-	unchanged

The queues holding event messages, input messages and output messages are searched. Messages originating from the same pool as the regret message are returned with result 'not_processed' and finally the regret message is returned.

3.7.4.4 Match

A process may simulate input from the console by sending a match message to OPERATOR.

	to OPERATOR	from OPERATOR
u1	6	unchanged
u2	-	result
u3	-	unchanged
u4	-	unchanged

The buffer is supposed to hold a record of type 'opbuffer' (appendix K.3). The field 'name' is used as the key in searching the queue holding input messages. If an input message is found with matching 'name' field, the data from the 'match' message is copied to the input message, and both messages are returned.

Results	Meaning
0	ok, data copied
1	Notprocessed, no matching input message.

3.8 Driver Input/Output Routines

In the description of the input/output routines a device is considered as containing a number of registers:

- control
- statusin
- statusout
- datain
- dataout

where information is transferred to/from by means of commands issued by the RC3502 machine.

The procedures inbyteblock, inwordblock, iowbwc, outbyteblock, and outwordblock interprete the actual message buffers as being of the pseudo type:

```
buffertype= ARRAY (0..65535) OF byte
```

The message buffer attributes, according to the driver conventions /Driver Conv./, are therefore treated as unsigned attributes in the range 0..65535.

3.8.1 Clearinterrupt

PROCEDURE clearinterrupt;

Not predefined.

- clears the current interruption level and waits for an interrupt. If the process has status 'timedout', the call has no effect.

3.8.2 Control

PROCEDURE control(control_word: 16bittype;VAR chmsg: reference);

Not predefined.

- The contents of the parameter 'control_word' are transferred to the CONTROL register in the device selected by the channel message chmsg. The current interruption level is not cleared so the next statement is executed without waiting for interrupt from the device. The type of 'control_word' may be any type of size 16 bits.

An exception occurs if:

- the reference variable 'chmsg' is nil.
- the reference variable 'chmsg' is not a channel message.

3.8.3 Controlclr

PROCEDURE controlclr(control_word: 16bittype);

Not predefined.

- The contents of the parameter 'control_word' are transferred to the CONTROL register in the device selected by the current interrupt level. If the process does not have status 'timedout', the current interruption level is cleared, and the next statement is executed when an interrupt arrives from the device. If the process has status 'timedout', execution continues immediately.

The type of control_word may be any type of size 16 bits.

An exception occurs if:

- the reference variable chmsg is nil
- chmsg is not a channel message

3.8.4 Ctrwaitid

FUNCTION ctrwaitid(c: 16bittype; msecs: integer): activation;

Not predefined.

- waits for two kinds of event:

  1. An interrupt (a_interrupt)
  2. Expiration of a delay period (a_delay)

The timer field of the process descriptor is initialized to the value of 'msecs' DIV 1000 and the control word 'c' is sent to the external device connected to the current interruption level, before the wait is performed.

Note that delay activation only takes place, if the routine 'definetimer' has been called.

3.8.5 Ctrwaitim

```
FUNCTION ctrwaitim(c: 16bittype; VAR r: reference;
                   VAR m: mailbox): activation;
```

Not predefined.

- waits for two kinds of event:

  1. An interrupt (a_interrupt)
  2. The arrival of a message to the mailbox m (a_mailbox)

The control word 'c' is sent to the external device connected to the current interruption level, before the wait is performed.

An exception occours if:

- the reference variable 'r' is not nil.

3.8.6 Ctrwaitimd

```
FUNCTION ctrwaitimd(c: 16bittype; VAR r: reference;
               VAR m: mailbox; msecs: integer): activation;
```

Not predefined.

- waits for three kinds of event:

  1. An interrupt (a_interrupt)
  2. The arrival of a message to the mailbox m (a_mailbox)
  3. Expiration of a delay period (a_delay)

The timer field of the process descriptor is initialized to the value of 'msecs' DIV 1000, the control word 'c' is sent to the external device connected to the current interruption level, and the wait is performed.

Delay activation only takes place, if a call of 'definetimer' has been issued. An exception occurs if:

- the reference variable is not nil.

3.8.7 Eoi

```
FUNCTION eoi: boolean;
```

Predefined.

- true if the EOI (End Of Information) status bit is 1 in the process status word in the process descriptor. The EOI status bit is updated whenever a read or write data command is issued by the process.

A call of eoi returning the value true after a read command indicates that the device has responded with no data. A call of eoi returning the value true after a write command indicates that the device has accepted the data and wants no more data.

3.8.8 Getbufparam

```
PROCEDURE getbufparam(VAR bufparam: 64bittype; first, last: integer;
              VAR msg: reference);
```

Not predefined.

- The parameter 'bufparam' is supposed to be of type

```
     bufparamtype= RECORD
                     top, count: integer;
                     datastart: 32bittype;
                   END;
```

- returns the start address of the byte with index first in the message buffer referenced by msg. As a side effect count and top are updated as:

```
     count:= madd(usub(last, first), 1)
     top:= madd(last, 1)
```

The procedure is intended for initializing a DMA controller with the start address and count for an input/output operation.

The following exceptions may occur

- the value of the reference is NIL
- no message buffer
- size too small
- ult(last, first)

3.8.9 Inbyteblock

```
PROCEDURE inbyteblock(VAR next: integer; first, last: integer;
              VAR msg: reference);
```

Defined in rc3502env.

- inputs a block of bytes to the message buffer specified by 'msg', 'first' and 'last' from the device specified by the current interruption level. When the procedure terminates, 'next' will be the index of the byte following the last byte input.

The procedure will terminate in two situations:

- when next=madd(last, 1);
- when eoi=true

If nothing is input 'next'='first'.

The following exceptions may occur:

- the reference variable 'msg' is nil
- the message 'msg' has no message buffer

- size of 'msg' is too small
- ult(last, first)

3.8.10 Inword

PROCEDURE inword(VAR word: 16bittype; VAR chmsg: reference);

Not predefined.

- the contents of the DATAIN register in the device selected by the channel message 'chmsg' is transferred to the parameter 'word'. If eoi=true after the call, the contents of 'word' are undefined.

The type of 'word' may be any type of size 16 bits.

An exception occurs in the following situations:

- the reference variable 'chmsg' is nil
- 'chmsg' is not a channel message

3.8.11 Inwordblock

```
PROCEDURE inwordblock(VAR next: integer; first, last: integer;
                VAR msg: reference);
```

Defined in rc3502env.

- inputs a block of words to the message buffer specified by 'msg', 'first', and 'last' from the device specified by the current interruption level.

The procedure terminates in two situations:

- when next=madd(last, 1)
- when eoi=true

If nothing is input 'next'='first'.
First and last must specify an even number of bytes (usub(last, first) must be odd).

The following exceptions may occur:

- the reference variable 'msg' is nil
- the message 'msg' has no message buffer
- 'madd(usub(last, first), 1)' is odd
- size of 'msg' is too small
- ult(last, first)

3.8.12 Inwordclr

```
PROCEDURE inwordclr(VAR word : 16bittype);
```

Not predefined.

- The contents of the DATAIN register in the device, selected by the current interruption level, is transferred to the parameter 'word'. If the process has status 'timedout', execution continues immediately. If the process does not have status 'timedout', the current interruption level is cleared, so the next statement is executed when an interrupt arrives from the device.

The type of 'word' may be any type of size 16 bits.

3.8.13 Iowbwc

```
PROCEDURE iowbwc(bufparam: 64bittype);
```

Not predefined.

- the parameter 'bufparam' is supposed to be declared as

```
   bufparamtype= RECORD
                       top, count: integer;
                       datastart: 32bittype;
                    END;
```

The procedure outputs a block of words from a buffer specified by 'datastart'. The number of words will be 'count' DIV 2 (count=0 is interpreted as 32 K words).

The procedure is intended for simultaneous output of data residing in one message. The actual parameter to the procedure may be obtained by means of the procedure 'getbufparam'.

The procedure can be called after declarations as shown below

```
TYPE
  bufparamtype = RECORD
                    top, count: integer;
                    datastart: 32bittype;
                 END;

PROCEDURE iowbwc(bufparam: bufparamtype); EXTERNAL;
```

3.8.14 Messagekind

FUNCTION messagekind(VAR msg: reference): integer;

Defined in rc3502env.

- returns a negative number if the designated message is a channel message, otherwise the result is greater than or equal to 0.

An exception occurs if:

- the reference variable 'msg' is nil

3.8.15 Outbyteblock

```
PROCEDURE outbyteblock(VAR next: integer; first, last: integer;
               VAR msg: reference);
```

Defined in rc3502env.

- outputs a block of bytes from the message buffer specified by 'msg', 'first', and 'last' to the device specified by the current interruption level. When the procedure terminates, 'next' will be the index of the byte following the last byte output. The procedure will terminate when 'next=madd(last, 1)'.

If nothing is output next=first.

The following exceptions may occur:

- the reference variable 'msg' is nil
- the message 'msg' has no message buffer
- size of 'msg' is too small
- ult(last, first)

3.8.16 Outword

PROCEDURE outword(word: 16bittype; VAR chmsg: reference);

Not predefined.

- The contents of the parameter 'word' are transferred to the DATAOUT register in the device selected by the channel message 'chmsg'. The current interruption level is not cleared so the next statement is executed without waiting for interrupt from the device.

The type of 'word' may be any type of size 16 bits.

An exception occurs in the following situations:

- the reference variable 'chmsg' is nil

- 'chmsg' is not a channel message

3.8.17 Outwordblock

```
PROCEDURE outwordblock(VAR next: integer; first, last: integer;
                VAR msg: reference);
```

Defined in rc3502env.

- outputs a block of words from the message buffer specified by 'msg', 'first', and 'last' to the device specified by the current interruption level.

The procedure terminates when next=madd(last, 1).

If nothing is output the value of next becomes the value of first.

First and last must specify an even number of bytes (usub(last, first) must be odd).

The following exceptions may occur:

- the reference variable is nil
- the message 'msg' has no message buffer
- the value of madd(usub(last, first), 1) is odd
- size of 'msg' is too small
- ult(last, first)

3.8.18 Outwordclr

```
PROCEDURE outwordclr(word: 16bittype);
```

Not predefined.

- the contents of the parameter 'word' are transferred to the DATAOUT register in the device selected by the current interruption level. If the process does not have status 'timedout', the current interruption level is cleared, so the next statement is executed, when an interrupt arrives from the device. If the process has status 'timedout', execution continues immediately.

The type of 'word' may be any type of size 16 bits.

3.8.19 Reservech

```
FUNCTION reservech(VAR chmsg: reference; level, mask: integer):
               integer;
```

Defined in rc3502env.

- allocates the channel message to the interruption level specified by the parameter 'level'. The parameter 'mask' is intended for specification of the actions the user wants to perform on the interruption level. The parameter is not used in this revision.

Results	Meaning
0	Reservation OK. 'chmsg' refers to the allocated channel message.
1	The interruption level is already reserved or is not installed.
2	The reference variable 'chmsg' is not nil before call.

3.8.20 Reserveextmem

```
FUNCTION reserveextmem(VAR r: reference; class, size, memno,
               disp: integer): integer;
```

Not predefined.

- allocates a message buffer in the area 80.0000 to 9e.ffff, called 'external RAM'. The start address is specified by 'memno' and 'disp'. The buffer size is specified by 'size' in *words*. 'Class' specifies the type of check, the routine performs during reservation.

The memory has the format (by convention):

```
mem= RECORD
        intr,
        reset,
        class,
        sizelsb,
        sizemsb,
        version,
        intrno: byte;
        spare: ARRAY (7..31) OF byte;
    END
```

If 'size'= 0 , the size is taken from the memory locations 'sizelsb', and 'sizemsb', which is the size in bytes on Intel integer form.

The following checks are performed:

class=1	No check.
class=2	mem.reset=6.
otherwise	mem.class=class.

Result Meaning

0 Reservation ok. 'r' refers a message, where the buffer is the specified memory. The value of 'bufsize' is the actual size in bytes.

1 Already reserved. The memory is occupied, or the requested memory overlaps an area, which is already reserved.

2 The reference variable 'r' is not nil before call.

3 Illegal 'memno'. Only modules in the range 0..15 are legal.

4 No memory. No physical memory is installed in the requested area.

5 Illegal class. The requested class does not match the location 'mem.class'. Note class1 and class2.

6 No resources. Message headers could not be allocated to describe the requested memory area.

7 Illegal size. 'size=0' is returned from the memory locations 'sizelsb', and 'sizemsb'.

3.8.21 Sense

```
PROCEDURE sense(VAR status_in: 16bittype; status_out: 16bittype;
                VAR chmsg: reference);
```

Not predefined.

- The contents of the parameter 'status_out' are transferred to the STATUSOUT register in the device selected by the channel message 'chmsg'. As a response from the device the contents of the STATUSIN register are transferred to the parameter 'status_in'. The current interruption level is not cleared so the next statement is executed without waiting for interrupt from the device.

The type of 'status_in' and 'status_out' may be any type of size 16 bits.

An exception occurs in the following situations:

- the reference variable 'chmsg' is nil
- chmsg is not a channel message

3.8.22 Senseclr

```
PROCEDURE senseclr(VAR status_in: 16bittype; status_out: 16bittype;
                   compare, mask: integer);
```

Not predefined.

- The contents of the parameter 'status_out' are transferred to the statusout register in the device selected by the current interruption level. The contents of the STATUSIN register are transferred to RC3502. If the value of statusin is true and 'mask' has value 'compare', the original contents of the statusin register are transferred to the parameter 'status_in', and the next statement is executed without waiting for interrupt from the device. Otherwise the current interruption level is cleared and the procedure is repeated, when the next interrupt arrives from the device, unless the process changes status to 'timedout'.

The type of 'status_in' and 'status_out' may be any type of size 16 bits.

3.8.23 Setinterrupt

PROCEDURE setinterrupt(VAR ch: reference);

Not predefined.

- activates the interruption level controlled by the channel message 'ch'.

The following exceptions may occur:

- the reference variable chmsg is nil
- chmsg is not a channel message

3.8.24 Timedout

FUNCTION timedout: boolean;

Not predefined.

The results are:
- true: The process has status 'timedout'. The status is cleared and the timer field of the process descriptor is set to zero.
- false: The process is not 'timed out'.

3.8.25 Waiti

PROCEDURE waiti;

Defined in rc3502env.

- waits for an interrupt from an external device. If executed on interruption level 0 (class II or III), the caller will be permanently descheduled (i.e. suicide).

3.8.26 Waitid

FUNCTION waitid(msecs: integer): activation;

Defined in rc3502env.

- waits for two kinds of event:
    1. An interrupt (a_interrupt)
    2. Expiration of a delay period (a_delay)

The timer field of the process descriptor is initialized to the value of 'msecs' DIV 1000 before the wait is performed.

Note that delay activation only takes place if the routine 'definetimer' has been called.

3.8.27 Waitim

FUNCTION waitim(VAR r: reference; VAR m: mailbox): activation;

Defined in rc3502env.

- waits for two kinds of event:
    1. An interrupt (a_interrupt)
    2. The arrival of a message to the mailbox m (a_mailbox)

An exception occurs if the reference 'r' is not nil.

3.8.28 Waitimd

```
FUNCTION waitimd(VAR r: reference; VAR m: mailbox;
                 delay: integer): activation;
```

Defined in rc3502env.

- waits for three kinds of events:

    1. An interrupt (a_interrupt).
    2. The arrival of a message to the mailbox m (a_mailbox).
    3. Expiration of a delay period (a_delay).

An exception occurs if the reference 'r' is not nil.

3.9 Routines for Backwards Compatibility

The following routines, defined in rc3502env, are included for backwards compatibility, only. They should not be used, except in very special cases, where the use of the equivalent RTP constructs is impossible.

3.9.1 First

FUNCTION first=offset(VAR r: reference): integer;
FUNCTION first=offset(VAR c: chain): integer;

Defined in rc3502env.

- Cf. offset.
- The attribute name 'first' is the old RTP name for the RTP defined attribute 'offset'
- If the 'first' function is called with a parameter with value NIL or the message is a header message only, an exception occurs. Otherwise the result is the value of the 'first' attribute of the designated message.

NOTE: The 'first' attribute is the first word in the buffer, according to the driver conventions /Driver Conv./. A message with bufsize greater than 0 always holds the 'first' attribute. So the buffer is accessed without locking the reference variable.

3.9.2 Last

FUNCTION last(VAR r: reference): integer;
FUNCTION last(VAR c: chain): integer;

Defined in rc3502env.

- Cf. top.
- The attribute name 'last' is the old RTP name for the RTP defined attribute 'top', the relation is: top=last+1.
- If the 'last' function is called with a parameter with value NIL or the message is a header message only, an exception occurs. Otherwise the result is the value of the 'last' attribute of the designated message.

NOTE: The 'last' attribute is the second word in the buffer, according to the driver conventions /Driver Conv./. A message with bufsize greater than 0 always holds the 'last' attribute. So the buffer is accessed without locking the reference variable.

3.9.3 Next

FUNCTION next(VAR r: reference): integer;
FUNCTION next(VAR c: chain): integer;

Defined in rc3502env.

- Cf. bytecount.
- The attribute name 'next' is the old RTP name for the RTP defined attribute 'bytecount', the relation is:
  bytecount=next-first.

- If the 'next' function is called with a parameter with value NIL or the message is a header message only, an exception occurs. Otherwise the result is the value of the 'next' attribute of the designated message.

NOTE: The 'next' attribute is the third word in the buffer, according to the driver conventions /Driver Conv./. A message with bufsize greater than 0 always holds the 'next' attribute. So the buffer is accessed without locking the reference variable.

3.9.4 Ref

FUNCTION ref(VAR mbx: mailbox): ↑ mailbox;

Defined in rc3502env.

- returns a pointer value to the pointed object 'mbx'. 'Ref' can only be applied to variables of type mailbox.

Ref is the old RTP means for getting mailbox pointers. As an alternative, a pointer to mailbox may be achieved by allocating the mailbox on the heap.

3.9.5 Setfirst

PROCEDURE setfirst(VAR r: reference; val: integer);
PROCEDURE setfirst(VAR c: chain; val: integer);

Defined in rc3502env.

- Cf. setoffset.
- If the 'setfirst' procedure is called with a reference parameter with value NIL, or the message is a header message only, an exception occurs. Otherwise the value of the 'val' parameter is assigned to the 'first' attribute of the designated message.

NOTE: The 'first' attribute is the first word in the buffer, according to the driver conventions /Driver Conv./. A message with bufsize greater than 0 always holds the 'first' attribute. So the buffer is updated without locking the reference variable.

3.9.6 Setlast

PROCEDURE setlast(VAR r: reference; val: integer);
PROCEDURE setlast(VAR c: chain; val: integer);

Defined in rc3502env.

- Cf. settop.
- If the 'setlast' procedure is called with a reference parameter with value NIL, or the message is a header message only, an

exception occurs. Otherwise the value of the 'val' parameter is assigned to the 'first' attribute of the designated message.

NOTE: The 'last' attribute is the second word in the buffer, according to the driver conventions /Driver Conv./. A message with bufsize greater than 0 always holds the 'last' attribute. So the buffer is updated without locking the reference variable.'

3.9.7 Setnext

```
PROCEDURE setnext(VAR r: reference; val: integer);
PROCEDURE setnext(VAR c: chain; val: integer);
```

Defined in rc3502env.

- Cf. setbytecount
- If the 'setnext' procedure is called with a reference parameter with value NIL, or the message is a header message only, an exception occurs. Otherwise the value of the 'val' parameter is assigned to the 'next' attribute of the designated message.

NOTE: The 'next' attribute is the third word in the buffer, according to the driver conventions /Driver Conv./. A message with bufsize greater than 0 always holds the 'next' attribute. So the buffer is updated without locking the reference variable.

3.10 RTP Language Supporting Routines

The following list of routine names denote the routines which the RTP system uses for support of the more complicated language features. These routines should not be explicitly called by the user, if done, an error message may be given.

```
checkstack,
delete_semaphore,
empty,
getmembyte,
initpool,
initsem,
_bufsize__rc,
_dec_byte_rc,
_exchange_rc,
_exit_____rc,
_first____rc,
_heapproc_rc,
_heapref__rc,
_heapsh___rc,
_inc_byte_rc,
_initchainrc,
_initmbx__rc,
_initpool_rc,
_initproc_rc,
_initprog_rc,
_initref__rc,
_initsem__rc,
_initsh___rc,
_last_____rc,
_new______rc,
_nilref___rc,
_region___rc,
_reg_exit_rc,
_sendallocrc,
_shift____rc
```

4. THE RC3502 MACHINE

4.1 Run Time Environment

After autoload and system initialization, the process structure is:

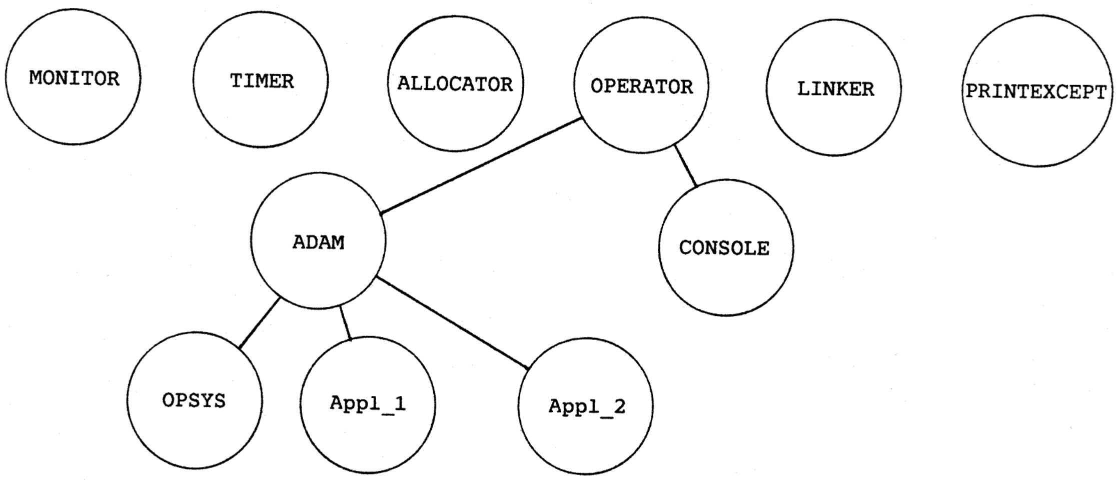


Fig. 4.1. Process structure after initialization

MONITOR performs medium term scheduling (start, stop).

TIMER performs delay timing, time out of processes, and controls a system clock for time stamping.

ALLOCATOR administers allocation and deallocation of RAM memory and interruption levels.

ADAM is the root of the dynamic tree of processes. ADAM automatically creates and starts a number of incarnations:
- FS
- IMCSTART
- OPSYS
- S

LINKER administers a catalogue of programs and routines, the LINKER catalogue.

OPERATOR is the interface between a human operator and the running processes. OPERATOR performs input/output to the debug console.

OPERATOR handles messages signalled to the operator mailbox.

OPSYS is a command interpreter, functioning as an interface between a human operator and the Run Time processes.

EXCEPTION prints a list of the dynamic chain of routine calls, when the procedure trace is called or the systems exception procedure is called as a result of run time error in a process.

S If a program S exists in the LINKER catalogue, an incarnation of S will be created and started. This will be the case when a program S is blasted in PROM or autoloaded.
S may replace OPSYS by its own NEW_OPSYS.

4.2 Monitor Process

The main purpose of the monitor is to control the set of active processes.

The active processes are divided into three priority classes:

Class I:	High priority
Class II:	Medium priority
Class III:	Low priority

Scheduling of class I processes is managed by the hardware interrupt priority mechanism. Processes in class I are running on an interruption level greater than zero.

Class II and III processes are running on interruption level 0.

Scheduling of class II and III processes is also performed by the hardware.

The class II processes (coroutines) are scheduled according to internal priority in the class and round robin for a given priority.

The class III processes are scheduled after a time sliced round robin algorithm with built-in priority.

The monitor is activated, when processes call the routines break, remove, start, resume or stop.

4.3 Driver Processes

By definition, a driver process is a program which contains the

CHANNEL <reference variable> DO <statements>;

construction or the standard input/output routines from section 3.

Access to most input/output routines and the CHANNEL statement is a reference variable which refers to a message of kind 'channel message' (cf. FUNCTION messagekind).

A channel message is obtained by calling the routine reservech specifying the device or interruption level, the process wants to control.

The system guarantees that at most one channel message is allocated per device or interruption level.

When a process executes a CHANNEL statement or calls an input/output routine, which demands a channel message, it is checked that the reference variable is not nil and that the reference variable refers to a message of kind 'channel message'.

In the input/output routines, which do not wait for an interrupt from the device, this is also checked.

Before execution of the first statement in the CHANNEL statement, the incarnation has entered the class I priority class. A possible 'timed out' status is cleared.

The statements in the CHANNEL statement are executed on the hardware interruption level specified by the channel message.

After execution of the last statement in the CHANNEL statement, the process reenters the priority class (II or III) which the process left, when entering the CHANNEL statement. An eventually 'timed out' status is cleared.

The user should be very much aware of the fact that the execution of processes in the priority classes II and III, besides all processes running at a hardware interruption level less than the interruption level of the user's processes, is disabled while executing statements in a CHANNEL statement.

This is true for all statements except those input/output routines, which wait for an interrupt, and thereby allow processes on lower interruption levels to execute instructions.

Therefore, the following recommendations should be followed:

- minimize the number of statements which are executed in a CHANNEL statement.
- the statements in a CHANNEL statement should mainly be input/output routine calls.

The system allows a process to possess several channel messages, but it is emphasized that it is the channel message used in the CHANNEL statement that defines the interruption level, where the incarnation is sensitive for interrupts.

Therefore it is normally the same channel message which is used both in the CHANNEL statement and in the input/output routines.

The channel messages in a CHANNEL statement and an input/output routine may differ. This may be used to sense a device with another device address than the interruption level defined by the CHANNEL statement, or even outside a CHANNEL statement.

4.3.1 Time Out

Time out is the activation of a process with status 'timed out'.

Time out of processes is performed by the TIMER process which decrements once per second the timer field of the process descriptor in all incarnations, which have called the routine definetimer.

Time out takes place, when the variable is decremented from 1 to 0.

The 'timed out' status is cleared, by calling the routine timedout or by exit of a CHANNEL statement.

4.4 Timer Process

The Timer Process:

- returns messages after a specified interval (Delay Timing),
- maintains a system clock,
- controls time out of processes.

Delay timing is requested by calling the procedure sendtimer (see section 3.2.27).

Time out is requested by assigning the time out period in seconds to the timer field of the process descriptor (assignment to own.timer), or by using one of the combined wait functions specifying a delay. Count down of the timer field will only happen after a call of the routine definetimer.

4.5 Allocator Process

After startup of the system, ALLOCATOR controls the available memory.

Memory is allocated to contain the process stack, when a process is created.

Variables of type pool may be allocated memory, when a newborn process is started, or when the routine allocpool is called.

The memory possessed by a process is deallocated when the controlling father process or an ancestor calls the remove procedure.

Memory is allocated in the following order: c0, c2, ..., fe, a0, a2, ..., be. This is done for hardware test purposes.

ALLOCATOR also controls access to all interruption levels. This is done by messages of kind channelmessage.

An interruption level is reserved by calling the routine reservech specifying:

```
     level - interruption level,
     mask  - facility mask.
```

A channel message is released by the statement:

```
      release(channel_message);
```

4.6 Linker Process

LINKER administrates the LINKER catalogue describing all programs in the system. The programs are linked to physical memory (statically relocated) and all calls of external routines are resolved.

The LINKER processes link/unlink requests from running processes (see the routines link and unlink).

A lookup function and a crc16 check function may also be requested (see the sendlinker routine).

4.7 Adam Process

Immediately after restart of the system the process tree may look like the figure below. (The figure is not complete).

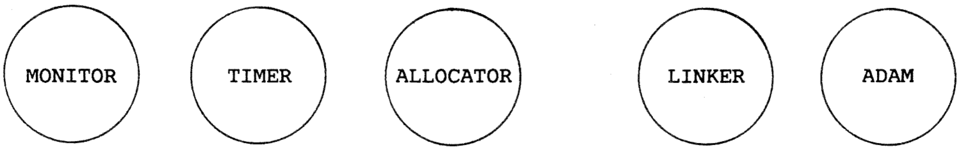


Fig. 4.2. Snapshot of part of the process structure

ADAM is the root of the dynamic tree of processes.
ADAM creates and starts a number of processes during initialization:

1. OPSYS, which interprets commands from the console and converts the commands to ADAM, LINKER, or TIMER messages.

2. S, which is the root of an application.

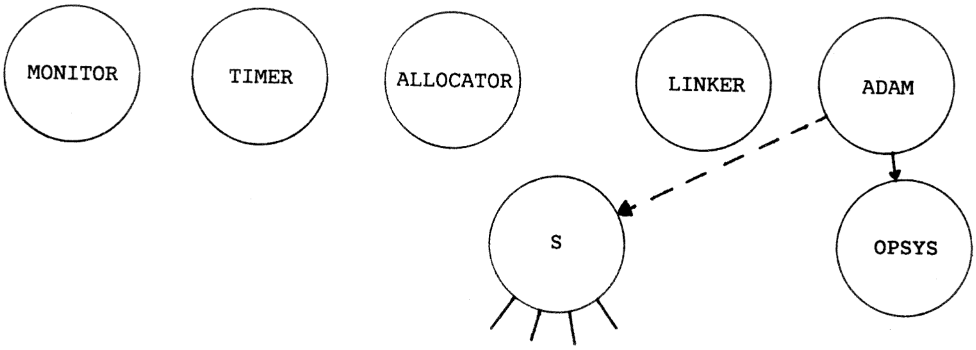


Fig. 4.3. Snapshot of process structure if S is included.

ADAM contains declarations of two classes of programs. The first class has no parameters. The second has one parameter, the 'system-vector':

```
s_type0= EXTERNAL PROGRAM;

s_type1= EXTERNAL PROGRAM(VAR sv: system_vector);
```

S is application dependent, and the formal parameters of the actual S must conform to one of these declarations.

ADAM controls a pool of program declarations, which conform to these declarations. These declarations may be used to request creation of new application sub-trees with ADAM as the controlling father (see later).

The parameter 'sv' may be used to pass references to system mailboxes like:

adam mailbox	sv(adammbx)
operator mailbox	sv(operatormbx)

These mailboxes might as well be obtained by the routine search_mailbox, because ADAM names the same mailboxes during initialization:

systemvector	searchmailbox
adammbx	'adammbx'
operatormbx	'operatormbx'
loadermbx	'loadermbx'
globalmbx	'globalmbx'
loaddrivermbx	'loaddrivermbx'
paxmbx	'paxmbx'
restartmbx	'restartmbx'

Furthermore these mailboxes and new mailboxes may be manipulated by a number of requests to ADAM.

ADAM may be requested to stop and remove any of the children and unlink the program by sending a message to the ADAM mailbox specifying the function to perform.

Example:

1. A human operator may stop, remove, and unlink the whole application tree (S) and start up a completely different application tree.
2. The application tree may stop, remove, and unlink OPSYS. A NEW_OPSYS may be created and started as a child of S.

Messages signalled to the ADAM mailbox are interpreted as having a message buffer of type:

```
adamtype= RECORD
            name1: alfa;
            name2: alfa;
            aux: integer;
          END;
```

Be aware of the difference between sending a link, create, start, stop, break, remove, or unlink message to ADAM and the call of the predefined routines link, create, start, stop, break, remove, unlink. The former operates on declarations in ADAM, the latter on declarations in the calling process. Messages to/from ADAM have the following user attribute interpretation:

	ADAM message	Answer
u1	function	unchanged
u2	not used	result
u3	not used	unchanged
u4	not used	unchanged

All messages which cannot hold a variable of type adamtype are returned with result=1. An unknown function is returned with result=15.

Function=1 (link)

If ADAM has a free program declaration of either s_type0 or s_type1 the program 'name1' is linked to a free program declaration.

Results	Meaning
0	ok
2	A program is already linked to a program declaration in ADAM with the external name 'name1'.
3	No free program declarations.
4	Program with name 'name1' does not exist in the LINKER catalogue.
5	Program with name 'name1' exists in the LINKER catalogue, but the number of parameters or the type of parameters do not match.

Function=2 (create)

An incarnation with name 'name2' of the program 'name1' is created with size 'aux'. More than one incarnation per program can be created.

Results	Meaning
0	ok
6	An incarnation of program 'name1' with name 'name2' is already created.
7	No program with external name 'name1' is linked to a program declaration in ADAM.
8	No storage or demanded size (aux) is too small.

Function=3 (start)

The process with name 'name2' is started with priority 'aux'.

Results	Meaning
0	ok
9	Unknown process.

Function=4 (stop)

The process with name 'name2' is stopped.

Results	Meaning
0	ok

10	Unknown process.

Function=5 (remove)

The process with name 'name2' is removed.

Results	Meaning
0	ok
11	Unknown process name.

Function=6 (unlink)

The link to the program with external name 'name1' is deleted.

Results	Meaning
0	ok
12	No program with external name 'name1' is linked to a program declaration in ADAM.
13	ADAM still controls an incarnation of the program 'name1'.

Function=7 (break)

A break operation is performed upon the process with name 'name2' with exception code 'aux'.

Results	Meaning
0	ok
14	Unknown process.

Function=8 (name mailbox)

A mailbox is created and catalogued under the name 'name1'.

Results	Meaning
0	ok
16	Overlap.
17	No resources.'

Function=9 (delete mailbox)

The mailbox catalogued under the name 'name1' is made invisible, i.e. further calls of search mailbox will fail.

Results	Meaning
0	OK
18	Unknown mailbox

Function=10 (rename mailbox)

The mailbox catalogued under the name 'name1' is catalogued under the name 'name2'. The name 'name1' is deleted from the mailbox catalogue of ADAM.

Results	Meaning
0	OK
19	Overlap, a mailbox is already catalogued under the name 'name2'.
20	No catalogued mailbox with the name 'name1' is found.

4.8 Operator Process

Read and write messages signalled to the OPERATOR mailbox sv(operatormbx) should at least contain a message buffer of type:

```
buffertype= RECORD
                first: integer;
                last: integer;
                next: integer;
                name: alfa; (* 12 chars *)
                databuf: ARRAY (18..max) of char
            END;
```

The message buffer follows the driver conventions /Driver Conv./. It is checked that the following assertions hold:

1. 18<=first
2. first-1<=last
3. databuf(last) is accessible in the buffer

OPERATOR identifies the input and output messages by the parameter 'name'.

Section 3.7 describes a set of routines for OPERATOR communication.

Messages to OPERATOR

- control: u1=0
- read: u1=1

The message is queued until it is "activated" by the human operator.

- write: u1=2

The message is printed as soon as possible.

Answers from OPERATOR

All messages are returned, when the appropriate action has been performed.

u1, u3, u4 are unchanged
u2 = result: 0=ok
1=not processed
4=unintelligible

"next" is undefined, unless result=ok.

A. REFERENCES

/ALGOL/ P. Naur (Ed.)
Revised Report on the Algorithmic Language ALGOL 60
Regnecentralen
Copenhagen, 1962

/Driver Conv./ RCSL No. 31-D652:
PASCAL80, Driver Conventions

/DSA-IMC/ RCSL No. 42-I1983
DSA Inter Module Communication
Functional Description

/MEM205/ RCSL No. 52-AA1182:
MEM205, General Information

/MUSIL/ RCSL No. 44-RT740
MUSIL

/Operating Guide/ PN: 99000771
RC3502/2, Operating Guide

/RC3502 Loader/ RCSL No. 52-AA1137:
RC3502 LOADER, Reference Manual

/RC3502 Ref. Man/ RCSL No. 52-AA1192:
RC3502/2, Reference Manual

/RTP/ PN: 99110141:
Reference Manual for the Programming Language Real-Time Pascal

B. USE OF THE REAL-TIME PASCAL COMPILER

B.1 Call of the Compiler

The RC8000 calling syntax is:

```
{<bin file>=}  RTP3502 {{<s><option>} {<s><context>}}*
                         {<s><option>}*  {<s><source>}
```

- <source> is a text file defining a program, a routine or a library of routines/programs (cf. Appendix B.2). If no source is specified, the compiler reads the source from current input.

- <context> is a text file containing declarations of types, constants, and external routines. Contexts can be used for definition of libraries. The syntax is described in appendix B.2.

- <bin file> is a file descriptor describing the backing storage area where the object code is stored.

If "<bin file>=" is omitted, "pass6code=" is assumed.

```
- <option>::=      codelist.<yes or no>
                 | codesize.<size>
                 | exception.<appetite>
                 | list.<yes or no or name>
                 | measure.<yes or no>
                 | pagelength.<length>
                 | pagewidth.<width>
                 | preserve.<yes or no>
                 | set.switchname.switchvalue
                 | short.<yes or no>
                 | spacing.<interval>
                 | stack.<appetite>
                 | stop.<pass nr>
                 | survey.<yes or no or pass>
                 | title.<title_name>
                 | version.<version>
```

- <size> is an unsigned integer in the range 0-65535. The size denotes the maximum number of bytes to be generated on one "program code page".

- <yes or no>::= yes | no

- <yes or no or name>::= yes | no | <name>

- list.yes means turn on listing of input on current output.
- list.no means turn off listing of input (default)
- list.<name> means turn on listing of input on file with name <name>

  Listing of the standard environments is always suppressed, even if list.yes is specified.

- <title_name> will appear in the page headings, if listing is turned on. Cf. the $TITLE directive.

- <length> and <width> indicate the length, in lines, and the width, in characters, of the pages of the list file, if listing is turned on. Cf. the $PAGELENGTH and the $PAGEWIDTH directives.

- measure.<yes or no>: consult appendix C: Performance Measurement.

- short.yes controls the compiler to use short encoded instructions in the generation of code whenever possible.

- stack.<appetite> (in words)
  This option overrides the default stack size, which the compiler estimates and outputs in the generated code (also known as default create size). The default stack size is used when the create routine is called with size=0.

  The "size" of a program incarnation may be calculated in the following way:

  ```
  size:= dynamic appetite
         + exception code appetite
         + 10 (for dump of the register stack)
  ```

  The dynamic appetite is the static appetite as mentioned for the program in the compilation survey plus the dynamic appetite for the most hungry of the called routines. The dynamic appetite of a routine is the static appetite plus the dynamic appetite for the most hungry of the called routines etc. In this connection please note the standard routines, the appetite of which is specified in the package description.

  For recursive procedures the appetite has to be multiplied by the maximum depth of the recursion.

The exception appetite amounts to a default value, which is specified in the package description. The appetite may be overruled by the 'exception' option in the call of the compiler.

NOTES:
If declaration of shielded objects occur, some standard initialization procedures are called.

Memory for the heap is *not* part of the stack space.

The stack space of the children is *not* taken from the stack of the parent process, but from the machine's pool of free memory. Create checks that there is memory for the process descriptor, the actual program parameters, and for dump of the register stack, before the child is created.

- exception.<appetite> (in words)
  This option is used for specifying the amount of stack space a process will reserve for exception handling. The default value is the room necessary for a call of the standard exception procedure. For a user defined exception procedure the option value may be the appetite seen in the information list from the compilation of the routine,(see further under the 'stack' option).

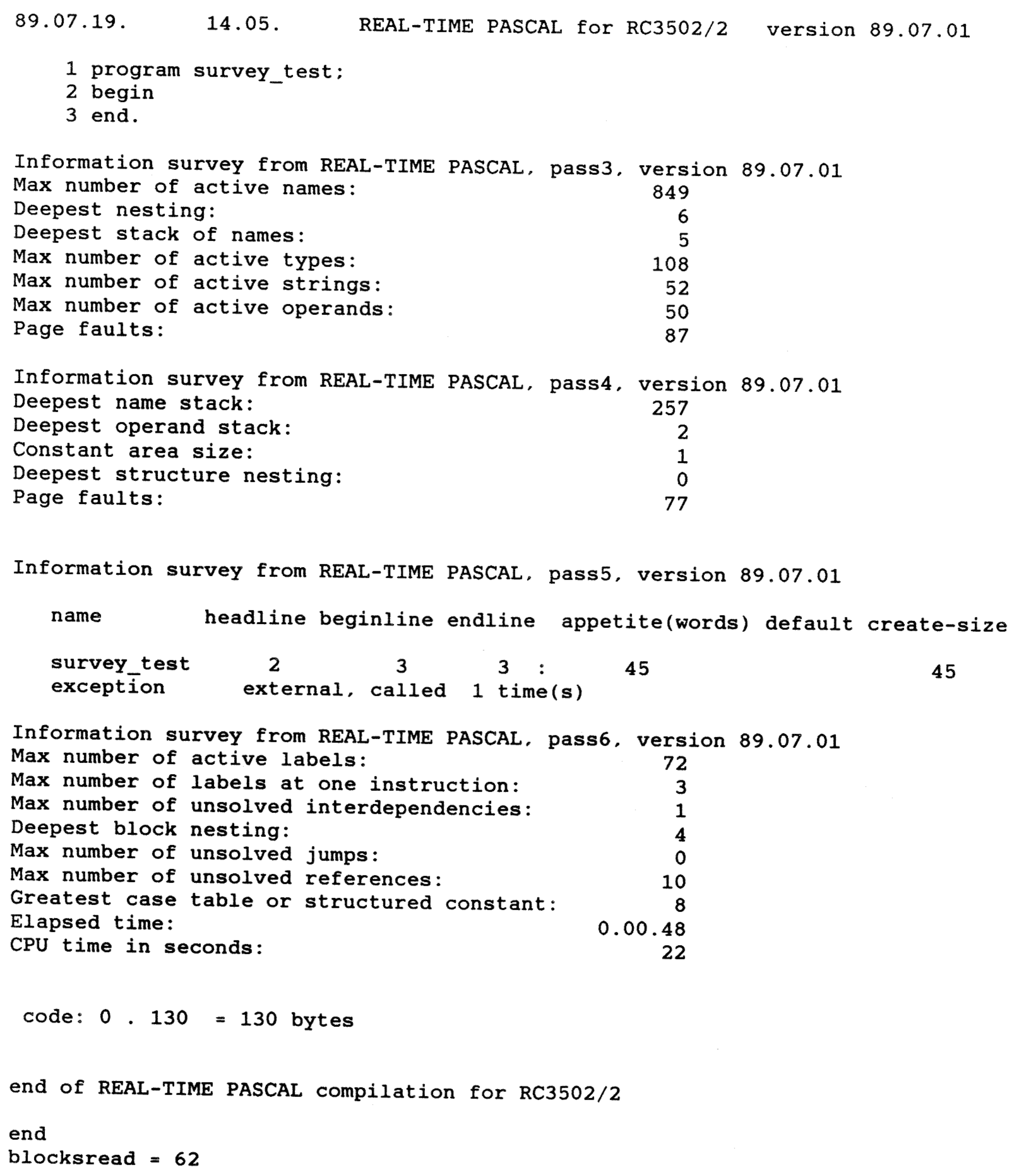
- codelist.yes means, list the generated code (on current output) in symbolic form with information about instruction execution times, logical addresses, and some comments.

- preserve.yes means, do not remove intermediate work files after use

- <yes or no or pass>::= yes | no | pass number

- survey.yes implies listing of some statistical information. cf. the following figure.

- survey.pass_number implies listing of some statistical information from the indicated pass of the compiler.



```
89.07.19.       14.05.      REAL-TIME PASCAL for RC3502/2   version 89.07.01

    1 program survey_test;
    2 begin
    3 end.

Information survey from REAL-TIME PASCAL, pass3, version 89.07.01
Max number of active names:                          849
Deepest nesting:                                       6
Deepest stack of names:                                5
Max number of active types:                          108
Max number of active strings:                         52
Max number of active operands:                        50
Page faults:                                          87

Information survey from REAL-TIME PASCAL, pass4, version 89.07.01
Deepest name stack:                                  257
Deepest operand stack:                                 2
Constant area size:                                    1
Deepest structure nesting:                             0
Page faults:                                          77


Information survey from REAL-TIME PASCAL, pass5, version 89.07.01

   name        headline beginline endline  appetite(words) default create-size

   survey_test       2         3       3  :      45                        45
   exception      external, called  1 time(s)

Information survey from REAL-TIME PASCAL, pass6, version 89.07.01
Max number of active labels:                          72
Max number of labels at one instruction:               3
Max number of unsolved interdependencies:              1
Deepest block nesting:                                 4
Max number of unsolved jumps:                          0
Max number of unsolved references:                    10
Greatest case table or structured constant:            8
Elapsed time:                                    0.00.48
CPU time in seconds:                                  22


 code: 0 . 130  = 130 bytes


end of REAL-TIME PASCAL compilation for RC3502/2

end
blocksread = 62
```

Fig. B.1. Survey Information from the Compiler

- set.switchname.switchvalue implies setting/defining a switch. The so defined switches may be used in the expressions of IF and ELSEIF directives (cf. 2.18 Compiler Directives).

- version.<version> implies setting of the version directive (cf. 2.18 Compiler Directives).

```
<version>::=    <integer>
              | <byte>.<byte>
              | <release>.<edition>.<variant>
```

- <interval> is the distance between two source line number records. The line number records are used by the standard exception procedure to relate the address of a run time error to a line interval of the source program.

  <pass nr>::= 1 | 3 | 4 | 5 | 6

- stop.<pass nr> terminates the translation after the pass specified.

  A short description of the passes:

  pass 1 performs syntax analysis
  pass 3 performs machine independent semantic analysis
  pass 4 performs storage allocation and machine dependent semantic analysis
  pass 5 performs symbolic code generation
  pass 6 performs transformation of symbolic code to binary format

- default values:

  the call:

  ```
  RTP3502 inputfile
  ```

  is equivalent to:

  ```
  pass6code=RTP3502 list.no codesize.512 spacing.52 stop.6 ,
    measure.no exception.11 short.yes survey.no codelist.no,
    stack.*** preserve.no inputfile
  ```

  ***: the default stack appetite is set according to a compiler generated value, computed as the sum of static appetites of the routines of and main body of the program.

Resource requirements

At least 45,000 hW memory (size 45000) area 8 temp disc at least 6 slices and 6 entries.

C. PERFORMANCE MEASUREMENT

It is possible to make some performance measurements on running Real-Time Pascal programs. The result of measuring includes the number of routine activations and the amount of time spent in the routines. The time indications are real-time, i.e. system overhead, time slicing and time for wait are included.

Measurement is initiated by a call of the procedure 'start_measure' with a mailbox as parameter. All succeeding routine calls and returns trigger update of a table. The table is allocated and initialized by 'start_measure'.

The measurement is terminated by a call of 'stop_measure' with the same mailbox parameter as 'start_measure'. 'Stop_measure' signals the table to a system mailbox and a new call of 'start_measure' is possible. Measurements may be started and stopped as long as it is possible to allocate an area of 2574 bytes to be used as measurement area.

The measurement results may be printed on the console by means of the program 'print_statis(tical_information)'. The order of results is FIFO. The output program deallocates the measurement areas after printing of the results.

There is an upper limit of 255 different routines per program, i.e. per program including internal sub-programs. If more than 255 different internal and external routines are called the (statically) last routines will use one common table entry, and the output program will put '*************' instead of name.

The output format may be seen in the following example. The number of calls is modulo 64K and the time amounts are given as:

hours.minutes.seconds.hundredths of a second

Since the names are found in the code of the routines, the output program must be run before the routines and programs are unloaded.

Use of the measurement tools:

The programs to be measured must be compiled together with 'measureenv' and with 'measure.yes' in the call of the compiler. 'Measureenv' includes external declarations of 'start_measure' and 'stop_measure', cf. K.4.

The necessary routines for measuring are supplied in 'measureplib', which is required in order to load/link processes compiled with measure.yes.

The output program is supplied as 'bprintstat'.

Example 9. Traptest

```
PROGRAM traptest;

VAR
  measure_mbx: mailbox;

  PROCEDURE empty_1;
  BEGIN END;

  PROCEDURE empty_2;
  BEGIN END;

  BEGIN
    start_measure(measure_mbx);
    FOR step1:= 1 TO 32 DO
      FOR step2:= 1 TO 1000 DO
        BEGIN
          empty_1;
          empty_2;
        END;
    stop_measure(measure_mbx);
    start_measure(measure_mbx);
    FOR step1:= 1 TO 33 DO
      FOR step2:= 1 TO 1000 DO
        BEGIN
          empty_2;
          empty_1;
        END;
    stop_measure(measure_mbx);
  END.
```

```
run traptest
run print_statis

>print_statis
Performance measurement for traptest

 name       called       time
------------------------------

traptest        01     0.00.34.60
start_measur    01     0.00.00.00
empty_1      32000     0.00.15.55
empty_2      32000     0.00.15.30
end
Performance measurement for traptest

 name       called       time
------------------------------

traptest        01     0.00.34.75
start_measur    01     0.00.00.05
empty_1      33000     0.00.16.75
empty_2      33000     0.00.16.20
end
```

Fig. C.1. Performance Measuring.

D. REAL-TIME PASCAL MESSAGES

If errors are detected during compilation, the compiler initializes the ok-bit to ok.no.

When the contexts and the source is spread over more files, the line number is set to 0, when a new file from the call line is to be compiled, files included as the result of an INCLUDE directive lead not to resetting the line number.

D.1 Messages from Pass 1

Messages from pass 1 are warnings, errors and fatal errors. All three kinds are marked with an arrow below the text line where the error was discovered. Warnings are given when the compiler is able to repair the problem, and compilation goes on. Errors are worse and the compilation stops after pass 1. The fatal errors are worst and compilation stops immediately.

The messages are:
0 Illegal character or fatal error
1 Identifier expected
2 '.' expected (end of process, prefix or context)
3 ';' expected
4 Identifier expected
5 '=' expected
6 ',' or ':' expected
7 Error in declaration
8 Set element or '.)' expected
9 (structured) constant or ':)' expected
10 '(' expected
11 ')' expected
12 Error after '&', string or character constant name expected
13 Constant, variable or '(<expression>)' expected
14 Expression expected
15 Expression expected
16 ')' or ',' expected
17 Actual parameter expected
18 ':)' or ',' expected
19 '.)' or ',' expected
20 Identifier or '?' expected
21 'OF' expected
22 'PROGRAM' expected
23 Expression expected
24 ',' expected
25 'ARRAY' or 'RECORD' expected after 'PACKED'
26 '..' expected
27 'END' or ';' expected
28 ';' or ')' expected
29 ':' expected
30 '.<field name>' or ';' expected
31 Unsigned integer expected
32 'BEGIN' expected

33 Error in FOR-variable specification
34 'TO' or 'DOWNTO' expected
35 Error in CHANNEL- or REGION-variable specification
36 variable denotation or 'TO' expected
37 'DO' expected
38 'DO' or ',' expected
39 'then' expected
40 Label expected (name or integer)
41 'else' expected
42 ';', 'end', or 'otherwise' expected
43 'end' or ';' expected
44 'until' expected
45 'endloop' or ';' expected
46 Error in label declaration list (',' or ';' expected)
47 '=' expected
48 '=' expected
49 Only procedure or function declaration allowed in a prefix
50 End of program expected
100 Error in real constant: digit expected
101 String did not terminate within line
102 Line too long, more than 150 characters.
103 Comment not terminated
104 export kind expected (value, size, address, disp, or offset)
105 lock type expected
106 ':' met, changed to '='
107 String too long, remaining part of string skipped
108 Name after concatenation symbol ('&') is not name of char constant
109 Exitloop or continueloop statement not inside repetitive statement
110 Routines in environment must be EXTERNAL declarations
111 VAR-section not allowed in contexts

Warnings in conjunction with compiler directives:
*** expression expected
*** (X)OR expected
*** AND expected
*** ")" expected
*** "=" expected
*** switch not defined, value 0 assumed
*** unknown directive
*** value outside range
*** ENDIF without matching IF
*** ELSEIF without matching IF
*** ELSE without matching IF or ELSEIF

Except for the 'not defined' message all the messages are followed by the line:
*** directive line skipped

The following messages indicating fatal errors may appear from pass1 of the Real-Time Pascal compiler. The message will be preceded by the line just being parsed with an indication of error number 0 discovered.

E.g.:

```
917  if prod = 1305    then
                     ↑ 0
***  const 'chbufmax' too small.
```

In case no other errors are discovered a fatal error may indicate that one or more of the compiler tables are insufficient in size. But most often this kind of fatal errors appear in consequence of syntactical errors, and after correction of the marked errors the fatal error may disappear.

The messages are:

*** parse stack overflow. const 'stackmax' too small
Parsing of a too complicated syntactical construction.

*** end of file encountered
Input exhausted before the parsing of the program/prefix has been successful.

*** recovery abandoned
The error recovery was unsuccessful.

*** reduction buffer overflow. const 'redumax' too small
Parsing of a too complicated syntactical construction.

*** const 'stringmax' too small
Literal text string too long.

*** const 'chbufmax' too small
Parsing of a too complicated syntactical construction.

*** const 'typebuffersize' too small
Too complicated type definition.

D.2 Messages from Pass 3

Error messages from pass 3 have the format:

*** pass 3 line <lineno>.<operand no> <kind> <error no> (, in environment '<env. name>') (: '<name>')

where:

<lineno> is the line number where the error is detected,

<operand no> gives a hint of where in the line the error was.

Operands are: identifiers and numbers. (Note: The empty set does not count).

First operand in a line has number 1. (Note: If <operand no> is 0, the error occurred before the first operand in the line, maybe even in the last part of the previous line).

<kind> indicates whether it is a warning or an error.

<error no> indicates which error was discovered. A list of error descriptions follows the error indications.

<env. name> is, if present, the name of the environment where the error is found.

<name> is, if present, the name (or the type name) of the identifier which caused the error. For example the name of an undeclared identifier.

The error descriptions are:

1 = identifier not declared
2 = identifier used before declaration
3 = identifier already declared at this level
4 = label-identifier not declared at all
5 = other identifier used as label-name
6 = label defined several times at this level
7 = label-identifier declared at surrounding level
8 = use of a multiple defined label
9 = a label-ident has been used in inner routine
10 = goto leading into control-structure
11 = goto out of lock- or channel statements
12 = label is not defined
13 = identifier is not a type-identifier
14 = object is not of parameterized type
15 = not parameter name of objects type
16 = unbound parameterized type not implemented
17 = unbound parameterized type only allowed as formal parameter type
18 = := assumed between type and (:
19 = type of structured constant not specified
20 = (: expected after type name
21 = error in record or array etc.
22 = constant is used in its own definition-expression
23 = pool ... of <illegal type>, shielded components not allowed
24 = illegal size'ing of pool type
25 = illegal limit-types in subrange def
26 = redefinition of object
27 = mailbox, chain, external program and pool may only be declared at program level
28 = programs inside functions/procedures forbidden
29 = formal type may not be used in this context
30 = functiontype may not contain shielded components
31 = only variables of surrounding scope allowed
32 = call of function not allowed in environment
33 = 'new' paramlist may be empty or exact the same
34 = forward-declared type/routine not followed by the real body

35 = function-value has not been defined at all

36 = locktype contains pointer-types

37 = lock-/with-type contains shielded component(s)

38 = operands not of same typename

39 = for-variable/startvalue/endvalue not of compatible types

40 = case-expression/caselabels not of compatible types

41 = if-expression must be boolean type

42 = until-expression must be boolean type

43 = while-expression must be boolean type

44 = short form of multiple locks not allowed

45 = buffer name missing in lock statement

46 = buffer type specification is missing

47 = lock-variable must be reference type

48 = channel-variable must be reference type

49 = type of operand must be enumeration-type

50 = varsizecall: identifier is not name of a variable

51 = operand cannot be used as variable

52 = <variable> in front of <.> is not a record

53 = <name> after <.> is not a fieldname of <variable>

54 = operand is not a SHARED object name

55 = <variable> in front of <uparrow> is not a pointer

56 = elements in set-value may not be of mixed types

57 = illegal mixture of types in relation

58 = illegal mixture of types in term or factor

59 = illegal type for monadic operator

60 = real occurring in expression

61 = real-division of integers not impl

62 = illegal operand kind in expression

63 = too few actual parameters to routinecall (or strucrecord

64 = error in routinecall

65 = error in structured-record constant

66 = typename in front of structured constant must be record or array

67 = the '***' operator must only occur in structured-array constant

68 = name in front of arglist is not of array-type

69 = index-expression doesn't match array-declaration

70 = incompatible types in assignment

71 = incompatible types in exchange

72 = the statement is not a procedure-call

73 = for-variable, with-variable or actual var-param is packed

74 = operand may not be assigned to, type is shielded or frozen

75 = operand may not be exchanged, type is mailbox, external program, pool or frozen

76 = type must be: reference or process

77 = formal and actual type must match exactly

78 = actual and formal types are not compatible

79 = formal is not frozen, therefore actual may not be frozen

80 = the '?' may only occur in structured constants

81 = incomp. types in structured array-constant

82 = incomp. types in structured record-constant

83 = incomp. types in var-initialization

84 = repetition must be integer

85 = value-export demands constant

86 = offset-export demands variable
87 = size-,disp-,addr-export demands constant or variable
88 = disp-export demands 'fielding'
89 = xor, integer-and, and integer-or not implemented on z80
90 = a program may not be FORWARD declared
91 = FORWARD specified type must be pointer
92 = process, chain, external program, and reference not allowed in SHARED
93 = actual parameter type must be reference or chain

Fatal Errors from Pass 3

The following message indicates a more complicated source program structure than the dimensioning of pass 3 is prepared for.
The message will be preceded by:

*** error found in pass3, at sourceline: <line no>

The message is:
max_nested_calls exceeded <limit>

More than <limit> nested routine calls and/or index specifications.

Compiler Errors from Pass 3.

The following messages indicate errors in the compiler, and should be reported to the maintenance staff. The messages are preceded by:

*** program inconsistency in pass3, at sourceline <line no>
cause is:

The messages are:

unknown-namekind
unknown-argkind
unknown-opcode
unknown-routinekind
unknown-paramkind
unknown-spix
unknown-stdtype-name
unknown-typekind
unknown-rectype
wrong pass1/pass3 combination
unstack-nontype
inconsistent allocation of type node

the message is followed by a trace of active routines of pass3 and an RC8000 fp break 0.

D.3 Messages from Pass 4

All error messages from pass 4 have the format:

*** pass 4 line <no>, <text>

where <no> is the line number where the error is detected.

<text> is one among the following:

subrange def.
Error in the definition of subrange type - lower bound greater than upper bound.

set def.
Error in the definition of set type - lower bound of the basis subrange type is negative.

pool def.
Error in the definition of pool type - number of elements is negative.

record size
Size of record type greater than 65536 bytes.

array size
Size of array type greater than 65536 bytes.

no init. or program in environment
Initialization of variables and program declarations in environment not allowed.

constant value
Value of constant outside interval bounds.

case label range
Value of first label greater than value of last label in a case label interval.

constant
Syntax error in number.

set constant
Error in set constant, - negative constant, or value of constant element, or interval with element(s) with value greater than 1023.

times
Wrong number of values in constant of array type.

not constant
Variable, or SET constant used in expression, outside the statement part of a procedure or program.

stack
Variable(s) in a block occupies more than 65536 bytes.

overflow
Value of constant or constant expression outside the interval -32768..32767.

dynamic variable not allowed
Illegal use of dynamic string variable.

initialize dynamic
type not allowed
Variables of dynamic types cannot be initialized in the declaration part.

dynamic type
not allowed
Dynamic types are not allowed for: set, pool, program parameter, VALUE routine parameter, function result, and structured constant.

access to zero sized object

WITH-AS: size of new type exceeds that of old

compiler error
<text>
Error in pass 4. Should be reported to the maintenance staff. <text> may be one of the following:
addressing
bit count
for error
scalar error
wrong input: <no>
illegal type
error code = <no>

Compilation terminated by <error> where <error> is:

stack error
internal compiler inconsistency; should be reported.

name table overflow
inconsistency between pass3 and pass4

wrong pass 4
wrong version of pass 4.

Pass 4 may give a warning:
WARNING, trouble with code file: <file name>
After this warning the next pass of the compiler may not be able to read the output from pass4. Possibly because of disc troubles.

D.4 Messages from Pass 5

Errors detected by pass 5 are indicated by the text:

*** pass 5 line <line no>, <text>

where <text> is one of the following:

+decl caselabel
Case labels are not distinct.

too many formal parameters
    Too many formal parameters of program or routine.

RTP3502lib does not exist
    The library of standard routines is not found, this error may also be indicated by the Pascal system message: "file does not exist: RTP3502lib", followed by a trace. This error should be reported to the operator of the RC8000.

user is not allowed to call system-routines
    Some system-routines are protected against user call.

Fatal errors are indicated by:

*** pass 5 line <line no>, compilation terminated by <text>

where <text> is one of the following:

external routine table overflow
    Use of too many external declared routines.

too many formal parameters
    Too many formal parameters of program or routine.

operand stack overflow

pass 4 code error
    Internal inconsistency, should be reported.

inconsistency in pass 5
    Should be reported.

wrong pass 4/pass 5 combination

hardware stack overflow, should be reported.

The fatal error messages are followed by some internal pass 5 status information:

```
    token        = <current code>
    no of items= <no of read items>
    stack        = stack size
    hw stack size=<size of work area>
```

This status information is followed by the pascal error message: Break, followed by a trace of the active pass5 routines.

Internal errors in pass5 during code generation (should be reported):

error in hw stack evaluation.

followed by a line with one of the messages.

- occurred in generating code for simple open function.
- occurred when generating code for array with system components
- occurred when generating code for ref removal
- occurred when generating code for process removal

followed by the line:
hw_stack_size is: <size>

D.5 Messages from Pass 6

The messages are:
constant <name> too small. Source line number = <line no>.

> If <name> is "buffersize" the program contains a structure occupying more than 64k bytes.
>
> If <name> is "descr1_size" there are too many formal parameters in the program or routine. The indicated source line is the last line of the program.

Compiler error detected. Please inform the compiler group.

> Meaning: internal compiler inconsistency.

Error in compilation.
Use option codesize, with at least <number> as page size.

> Meaning: the program contains a structure which needs <number> consecutive bytes on the same code page.

E. REAL-TIME PASCAL SOURCE PROGRAM UTILITIES

E.1 Indent, Text Formatting Utility

The program performs indention of source programs depending on the options specified in the call and on the keywords(reserved words) of Pascal/Real-Time Pascal.

call :
 {output_file =} indent input_file {option}*

Available options are:

noind	no indention, the text is left justified
myind	indention is not changed
lines	line numbers are generated
lc	lower case identifiers, upper case key words
uc	the text is converted to upper case
mark(.count)	mark begin/case/record - end matching. If count specified then mark with digits, else with !
blockcount	show current block number at the left of the text
list	equivalent to: lines mark
pascal	RC8000 pascal indention mode
rtp	RTP3502 indention mode
RTP35022	RTP35022 indention mode
include	expand $INCLUDE directives
set.switch.val	assign the value of val to switch
help	produce this list of help information

Storage requirements:
The core store required for indent is 16000 hW (size 16000).

Error messages:

If errors are detected, the ok-bit is initialized to ok.no.

The messages are:
??? illegal input/filename
Input file must be specified.

** warning, end(s) missing
An error in the begin-end structure has been detected.

** premature end of file.
Comment or string not terminated.

The default language, i.e. set of key words, are detected from the catalogue entry tail of indent. The word following the date and time is coded as follows:

1	for Pascal
2	for RTP
22	for RTP35022

E.2 Cross Reference Program

Produces a cross reference listing of the identifiers and numbers and a use count of the PASCAL/Real-Time Pascal keywords used in the input text.

The cross reference list is made with no regard to the block structure of the program. The list is sorted according to the ISO-alphabet, i.e. numbers before letters, but with no discrimination between upper and lower case letters.

The occurrence list for an identifier consists of a sequence of Pascal/Real-Time Pascal line numbers. The occurrence kind is specified by means of the character following the line number:

*	meaning the identifier or number is found in a declaration part.
=	meaning the identifier is assigned to in the line specified.
:	meaning the identifier or number occurred as a label.

blank all other uses.

<<<<<<<<<<<<<< in the list is a warning denoting that the name consists of more than 12 characters, which is the number of significant characters for Pascal-identifiers. Only produced if the language is Pascal.

Syntax of call:
 output_file = cross input_file {option}*

Available options:

bossline.<yes or no>	default is no, i.e. do not add boss line numbers
survey.<yes or no>	default is no, i.e. no print of survey information
pascallines.<yes or no>	yes is default for Pascal programs, no is default for RTP programs.
title.<name>	default name is input_file name the name will appear in the page headings
keywords.<yes or no>	default is yes, i.e. the keywords are listed separate with occurrence counts
lpp.<number>	number of program/cross ref Lines Per Page
cpp.<number>	number of cross reference Columns Per Page
language.<language>	for selection of language specific keywords

<language>= pascal | rtp (for RTP3502) | RTP35022

The default language and page format of CROSS is obtained from the catalogue entry tail. The word following the date and time holds the information. By means of the utility crossinst is it possible to check/change the default settings, see below.

Storage requirements:

The core store required for cross is at least 40000 hW (size 40000), but the requirements depend on the size of the input text.

If errors are detected, the ok-bit is initialized to ok.no.

The messages are:

```
???   illegal output-filename
      Left hand side of the call must be a name.

???   output file must be specified

???   illegal input-filename
      Input file must be specified.

???   yes or no expected

???   error in bracket structure, detected at line: xx
      Missing ")"('s)

???   error in blockstructure, detected at line: xx
      Unmatched end.

***** warning: element queue ran full. Some of the names of a list
      are not marked with the correct attribute ( :, =, * ).
```

E.2.1 Crossinst

The default settings of language and page format of the cross reference program may be checked/changed by means of the program crossinst. Below is an example of a run of crossinst followed by the result of a lookup of the catalogue entry tail of cross. The word after the date and time is coded as follows:
Left half word (in front of the period) contain language information:

1	for Pascal
2	for RTP
22	for RTP35022

Right half word (following the period) contain page format information, the value is coded as:

Lines Per Page*16+Columns Per Page.

Example on use of crossinst:

```
*crossinst
Current settings are shown in brackets


Language: 1: pascal / 2: RTP3502 / 3: RTP35022  (1)
1


Page format:
Program/cross reference lines per page, 5-255, (46)
46


Number of cross reference columns in the table, 1-15, (15)
15

end
blocksread = 7
*lookup cross
cross       =set 41 discl d.890411.1301 1.751 0 2.0 68  ; project
            ;  193 76 3 -8388607 8388605
```

E.3 RTP-Compress

Some compilation time may be saved if common contexts are compressed. Typically, contexts are compiled over and over, but only changed now and then. Hence, compilation time is spent on comments and superfluos blanks from contexts which are of minimal interest for the main program. Supplied with the compiler is a utility program called rtpcompress. This program may compact the text by removing comments and blanks. The syntax of call of the program is:

{outfile=} rtpcompress inputfile

If <outfile> is omitted current output is chosen.

F. REAL-TIME PASCAL OBJECT PROGRAM UTILITIES

F.1 PLIBINSERT

Program or routine libraries for RC3502 may be constructed by means of the program plibinsert.

Plibinsert may collect a group of object programs into one object file. The cross linker may then include all the programs if specified as <obj file> in the call, or just include the programs referenced from an included object program if specified as <lib file>.

Plibinsert may also be used for updating an existing library. If the program to insert already is in the library the old one will be replaced by the new one.

The call is:

```
 {<new lib>=} plibinsert {<new object>}* {lib.<old lib>}
```

<new lib> is the updated library; default name is "lib".

<old lib> is the old library, possibly empty; default name is "oldlib".

<new object> is the name of a file containing object programs to be inserted into <old lib>, possibly replacing existing entries; default name is "pass6code".

F.2 PLIBLOOKUP

Pliblookup produces a catalogue listing of the programs included in a library. Programs of imagefiles may also be listed. The call is:

```
 pliblookup  {<library>}
```

<library> is the library to be scanned; default name is "lib".

F.3 PLIBALL

Pliball produces a catalogue listing of the programs in a library. A list of all the external routines called by the programs is produced for each program in the library. The call is identical to the call of PLIBLOOKUP.

F.4 PLIBDELETE

Plibdelete may be used to remove programs from an existing library.

The call is:

```
{<new lib>=} plibdelete {<entry>}* {lib.<old lib>}
```

<new lib>   is the updated library; default name is "lib".

<old lib>   is the old library; default name is "oldlib".

<entry>::= entry.<number>
    <program name>

Program names consisting of more than 11 characters and names including underscore may only be specified as "entry.<number>"; the entry number may be found by means of a call of pliblookup.

F.5 PLIBEXTRACT

Plibextract may be used to extract programs from an existing library.

The call is:

```
{<new lib>=} plibextract {<entry>}* {lib.<old lib>}
```

<new lib>   is the result library containing the extracted entries; default name is "lib".

<old lib>   is the old library; default name is "oldlib".

<entry>::= entry.<number>
    <program name>

Program names consisting of more than 11 characters and names including underscore may only be specified as "entry.<number>"; the entry number may be found by means of a call of pliblookup.

F.6 PLIBCONVERT

Plibconvert produces an RTP3502 library object file from a list of input files of arbitrary contents. The individual programs in the result file have the new kind DATA and the same name as the corresponding input file. The result file may be inspected by PLIBLOOKUP.

<outfile> = plibconvert {descriptor.<yes or no>} <infile>

<outfile>	is the resulting library object file.
<infile>	is the name of a file of arbitrary contents.
descriptor	defines whether the descriptor segments are generated for the input files during converting. 'descriptor.yes' is mandatory, when the final receiver of the result file is the RC3502 LOADER. 'descriptor.no' may be specified, if you want a transparent file transfer to the receiver (NOT the RC3502 LOADER).
Default:	descriptor.yes.

G. LOAD OR AUTOLOAD FILE GENERATION ON RC8000

The Real-Time Pascal compiler generates binary relocatable object programs. The object programs may be loaded by the RC3502 LOADER (cf. /RC3502 Loader/ and /Operating Guide/) via an FPA, or from an application program, simulating an external device. Subsection G.1 describes this way of program load. The RC3502 LOADER performs the necessary linkage editing and allocation of memory for the programs.

Another way to load programs in the RC3502 is autoloading by means of the RC3502 BOOT program. In that case the necessary linkage editing of the object programs must be performed by the CROSSLINK program on RC8000, before an autoload file is produced. This is described in subsection G.2.

G.1 How to Generate a Load File

G.1.1 Generating an FTS Load File

If 'xxxplib is a binary object program, produced by the Real-Time Pascal compiler, a load file for later load via RcLAN from an FTS server is generated on RC8000 by the call:

```
     <out file>=convertplib xxxplib
```

G.1.2 Generating an FPA Load File

If 'xxxplib' is a binary object program, produced by the Real-Time Pascal compiler, a loadfile for later load from FPA is generated on RC8000 by the call:

```
     <out file>=crc16 xxxplib
```

If the name of the outfile is the name of an RC8000 main process, the CRC16 program outputs directly to the device denoted by the main process description according to the autoload protocol.

G.2 How to Generate an Autoload File

G.2.1 CROSSLINK

The CROSSLINK program will generate a file containing all the specified object programs and the necessary library routines. The file will be a core image, i.e. references between the programs are solved.

Call:

```
<outfile>=crosslink {<lib file> | <obj file> | <params>}
```

<outfile>   contains the generated coreimage.

<obj file>::= <file name>

<file name>
contains one or more object programs which (all) must be included in the core image.

<lib file>::= lib.<file name>

<file name>
contains one or more object programs (routines) which may be included by CROSSLINK, if they are referenced from an included object program.

<obj file> and <lib file> can be output file(s) from the Real-Time Pascal compiler or may be generated by the utility program PLIBINSERT (see appendix F).

```
<params>::=    start.<base>.<displacement>
              | descr.<yes or no>
              | map.<yes or no>
              | print.<yes or no>.<words per line>
              | bind.<yes or no>
```

start

<base> and <displacement> specify where the first word of the coreimage is supposed to be autoloaded. The start-param may only occur before the first filename.

```
<base>::= <integer> | h<hexadecimal digits>
<displ>::= <integer> | h<hexadecimal digits>
```

Default is: hc0.h0100.

NOTE: If the coreimage is to be autoloaded by the BOOT program, start must be: start.hc0.h0100.

descr

defines whether the descriptor segments of the following programs are included in the coreimage or not.

NOTE: This option should *not* be used. Descr.no will generate a fatal error from CROSSLINK.

Default: descr.yes.

map

controls listing of the included programs. Each included program is listed with:

<programkind> <programname> <date time> <start of descriptor segment> <start of code segment> <length>

Default: map.yes.

<date time> is date and time of compilation of the program.

print

controls printing of the core image. Each line consists of:

<A>.<B> <C>.<D> <hexadecimal contents of core image words>

<A> and <B> is the absolute address(i.e. base and displacement) of the first word in the line.

<C> and <D> is the corresponding relative address(within the program).

bind

bind.yes secures that a new memory module always starts with a descriptor segment.

Default: bind.yes.

Bind.no will generate an error message from CROSSLINK if the programs are not for RC3502/2.

Example:

```
coreimage=   crosslink start.hc0.h0100,
             blinker,    *
             bmonitor,    *
             ballocator,  *
             bexception, *  (or bminiexcept),
             btimer,     *
             boperator, *
             bopsys,       *   may be substituted by your own
             ,                 operating system
             bloader,     *
             userprocess 1,
              ...,
             userprocess n,
             lib.stdplib
```

will generate a core image in a file with the name coreimage containing (all) the object program(s) in the files explicitly mentioned, extended by the necessary object programs from the library file stdplib, which contains all standard runtime routines (link, create, ...).

The core image will start in memory module c0, displacement 256, as demanded by the BOOT program.

The modules marked with an * constitutes the library 'system3502', this name might have been specified instead of all the module names.

A supplementary loader map is printed (see the example below).

```
*coreimage=crosslink system3502 lib.stdplib

   cross linker, version 88.10.29
    linking solved in: 4 scans

  kind       name            date      time     descr     code   length(hex)
----------------------------------------------------------------------------
PROGRAM    linker         1989.06.19 12.19    c0.0100  c0.012c  0fb4
PROGRAM    monitor        1989.06.19 12.19    c0.10b4  c0.10e0  1218
PROGRAM    timer          1989.06.19 12.19    c0.22cc  c0.22f8  0cb0
PROGRAM    allocator      1989.06.19 12.19    c0.2f7c  c0.2fa8  113e
PROGRAM    printexcept    1989.06.19 12.19    c0.40ba  c0.40e6  14c4
PROGRAM    adam           1989.06.19 12.19    c0.557e  c0.55aa  2010
PROGRAM    opsys          1989.06.19 12.19    c0.758e  c0.75be  22ca
PROGRAM    loader         1989.06.19 12.19    c0.9858  c0.9888  2180
PROGRAM    loaddriver     1989.06.19 12.19    c0.b9d8  c0.ba10  021c
PROCEDURE break          1989.05.30 14.06    c0.bbf4  c0.bc28  00f6
FUNCTION   create         1989.05.30 14.06    c0.bcea  c0.bd2a  033e
PROCEDURE definetimer    1989.05.30 14.06    c0.c028  c0.c058  00ac
FUNCTION   deletemailbo   1989.05.30 14.06    c0.c0d4  c0.c108  00dc
FUNCTION   delete_semap   1989.05.30 14.06    c0.c1b0  c0.c1e4  0076
FUNCTION   empty          1989.05.30 14.06    c0.c226  c0.c25a  007c
PROCEDURE exception      1989.05.30 14.06    c0.c2a2  c0.c2d2  00e0
PROCEDURE getswitches    1989.05.30 14.06    c0.c382  c0.c3b6  0086
FUNCTION   hometest       1989.05.30 14.06    c0.c408  c0.c440  0076
FUNCTION   initpool       1989.05.30 14.06    c0.c47e  c0.c4ba  0152
FUNCTION   initsem        1989.05.30 14.06    c0.c5d0  c0.c60c  013e
FUNCTION   link           1989.05.30 14.06    c0.c70e  c0.c746  00e2
FUNCTION   link           1989.05.30 14.06    c0.c7f0  c0.c828  0078
FUNCTION   memerrorlog    1989.05.30 14.06    c0.c868  c0.c89c  0092
FUNCTION   namemailbox    1989.05.30 14.06    c0.c8fa  c0.c932  0116
FUNCTION   name_semapho   1989.05.30 14.06    c0.ca10  c0.ca48  007c
PROCEDURE remove         1989.05.30 14.06    c0.ca8c  c0.cabc  0470
FUNCTION   reservech      1989.05.30 14.06    c0.cefc  c0.cf38  00e4
PROCEDURE restart        1989.05.30 14.06    c0.cfe0  c0.d00c  0078
FUNCTION   searchmailbo   1989.05.30 14.06    c0.d058  c0.d08c  00d8
FUNCTION   search_semap   1989.05.30 14.06    c0.d130  c0.d164  0078
PROCEDURE _sendallocrc   1989.05.30 14.06    c0.d1a8  c0.d1d8  006e
PROCEDURE sendlinker     1989.05.30 14.06    c0.d216  c0.d246  006e
PROCEDURE sendtimer      1989.05.30 14.06    c0.d284  c0.d2b4  006e
PROCEDURE setfirst       1989.05.30 14.06    c0.d2f2  c0.d326  0070
PROCEDURE setlast        1989.05.30 14.06    c0.d362  c0.d396  0070
PROCEDURE setswitches    1989.05.30 14.06    c0.d3d2  c0.d406  006e
PROCEDURE setu1          1989.05.30 14.06    c0.d440  c0.d474  006c
PROCEDURE setu2          1989.05.30 14.06    c0.d4ac  c0.d4e0  006c
PROCEDURE setwatchdog    1989.05.30 14.06    c0.d518  c0.d548  00a8
PROCEDURE start          1989.05.30 14.06    c0.d5c0  c0.d5f4  00f0
PROCEDURE stop           1989.05.30 14.06    c0.d6b0  c0.d6e0  00b8
PROCEDURE tofrom         1989.05.30 14.06    c0.d768  c0.d7a8  00a6
FUNCTION   unlink         1989.05.30 14.06    c0.d80e  c0.d842  00fe
FUNCTION   wild_compare   1989.05.30 14.06    c0.d90c  c0.d944  016e
PROCEDURE checkstack     1989.05.30 14.06    c0.da7a  c0.daaa  006e
PROCEDURE openopzone     1989.05.30 14.06    c0.dae8  c0.db38  0178
PROCEDURE opin           1989.05.30 14.06    c0.dc60  c0.dc90  00a8
PROCEDURE opwait         1989.05.30 14.06    c0.dd08  c0.dd3c  013c
PROCEDURE alloc          1989.05.30 14.06    c0.de44  c0.de7c  00aa
PROCEDURE print_descri   1989.05.30 14.06    c0.deee  c0.df22  01d2
  .
  .
  .
FUNCTION   double_mul     1989.05.30 14.16    c2.0330  c2.0368  0222
FUNCTION   double_divmo   1989.05.30 14.16    c2.0552  c2.058e  032e
FUNCTION   double_div     1989.05.30 14.16    c2.0880  c2.08b8  0082
FUNCTION   double_mod     1989.05.30 14.16    c2.0902  c2.093a  0082
   tree height : 8
   externals   : 91
   end, cross linker.

end
blocksread = 92
*o c
```

Fig. G.1. Example on Loader Map from CROSSLINK

G.2.1.1 Error messages from CROSSLINK

The possible error/warning messages are:

No outfile specified
Error in crosslink
Error in call of crosslink; parameter number NN
Illegal hexadecimal number in call; parameter number NN
Bad version, recompile NNNNN
yes or no expected after "descr."
Illegal contents of object or library file
Too many modules, (more than 200)
Illegal value of "start"-parameter; parameter number NN
Too many formal parameters, module: NNNNN
yes, no or yes.<number> expected after "print."
Unknown option; parameter number NN
Name expected after "lib."; parameter number NN
Inconsistency in loader map
yes or no expected after "map."
yes or no expected after "bind."
"bind.yes" required by RC3502/2

G.2.2 Generating an FTS Autoload File

Assume coreimage has been generated by CROSSLINK.
The call:

```
    <out file>=convertplib coreimage
```

will generate a file with the correct format for autoload via RcLAN from an FTS server.

G.2.3 Generating an FPA Autoload File

Assume coreimage has been generated by CROSSLINK.

The call:

```
    <out file>=crc16 coreimage
```

will generate a file with the correct format for the RC8000 AUTOLOAD-program.

If the name of the outfile is the name of an RC8000 main process, the CRC16 program outputs directly to the device denoted by the main process description according to the autoload protocol.

H. COMPLETE LIST OF LANGUAGE SYMBOLS

AND
ARRAY
AS
BEGIN
BEGINBODY *)
CASE
CHANNEL
CONST
CONTINUELOOP
DIV
DO
DOWNTO
ELSE
END
ENDLOOP
EXIT
EXITLOOP
EXPORT **)
EXTERNAL
FOR
FORWARD
FUNCTION
getswitch
GOTO
IF
IN
INSPECT
LABEL
LOCKBUF
LOCKDATA
LOOP
MOD
NOT
OF
OR
OTHERWISE
PACKED
POOL
PREFIX
PROCEDURE
PROGRAM
RECORD
REGION
REPEAT
SET
SHARED
SHIFT
THEN
TO
TYPE
typesize
UNTIL
VAR
varsize
WHILE
WITH
XOR

*) reserved for internal use.
**) reserved for Z80 version.

```
+      -      *      "      '      <      >
<>     <=     >=     (      )      (.     .)     (:
:)     =      :=     :=:    .      ,      :      ;
***    (*     *)     <*     *>     --     !      ?
..     $      &      #      ↑
```

I. PREDEFINED CONSTANTS, AND TYPES

The following constants and types are predefined:

```
CONST
  alfalength= 12;
  maxint= 32767;
  minint= -32768;
  maxpriority= 0;
  minpriority= -2;
  stdpriority= minpriority;

TYPE
  priotype= minpriority..maxpriority;

CONST
  create_ok= 0;
  create_process_not_nil= 1;
  create_program_not_linked= 2;
  create_no_memory= 3;

TYPE
  create_result= create_ok..create_no_memory;

CONST
  link_ok= 0;
  link_not_found= 1;
  link_no_parameter_match= 3;
  link_already_linked= 6;

TYPE
  link_result= link_ok..link_already_linked;

CONST
  unlink_ok= 0;
  unlink_no_program_linked= 1;
  unlink_program_busy= 2;

TYPE
  unlink_result= unlink_ok..unlink_program_busy;

  bit= 0..1;
  byte= 0..255;

  base_type= PACKED
  RECORD
    onebit: bit;
    mem_no: 0..63;
    nilbit: bit
  END;

  adr=
  RECORD
    base: base_type;
    disp: integer
  END;

  addr=
  RECORD
    nulbyte: byte;
    base: base_type;
    disp: integer;
  END;
  activation= (a_interrupt, a_mailbox, a_delay);
  boolean= (false, true);
  char=
   (
(*   0 *)     nul, soh, stx, etx, eot, enq, ack, bel, bs, ht,
(*  10 *)     nl, vt, ff, cr, so, si, dle, dc1, dc2, dc3,
(*  20 *)     dc4, nak, syn, etb, can, em, sub, esc, fs, gs,
(*  30 *)     rs, us, sp, ?, ?, ?, ?, ?, ?, ?,
(*  40 *)     ?, ?, ?, ?, ?, ?, ?, ?, ?, ?,
(*  50 *)     ?, ?, ?, ?, ?, ?, ?, ?, ?, ?,
(*  60 *)     ?, ?, ?, ?, ?, ?, ?, ?, ?, ?,
(*  70 *)     ?, ?, ?, ?, ?, ?, ?, ?, ?, ?,
(*  80 *)     ?, ?, ?, ?, ?, ?, ?, ?, ?, ?,
(*  90 *)     ?, ?, ?, ?, ?, ?, ?, ?, ?, ?,
(* 100 *)     ?, ?, ?, ?, ?, ?, ?, ?, ?, ?,
(* 110 *)     ?, ?, ?, ?, ?, ?, ?, ?, ?, ?,
(* 120 *)     ?, ?, ?, ?, ?, ?, ?, del, ?, ?,
(* 130 *)     ?, ?, ?, ?, ?, ?, ?, ?, ?, ?,
(* 140 *)     ?, ?, ?, ?, ?, ?, ?, ?, ?, ?,
(* 150 *)     ?, ?, ?, ?, ?, ?, ?, ?, ?, ?,
(* 160 *)     ?, ?, ?, ?, ?, ?, ?, ?, ?, ?,
(* 170 *)     ?, ?, ?, ?, ?, ?, ?, ?, ?, ?,
(* 180 *)     ?, ?, ?, ?, ?, ?, ?, ?, ?, ?,
(* 190 *)     ?, ?, ?, ?, ?, ?, ?, ?, ?, ?,
(* 200 *)     ?, ?, ?, ?, ?, ?, ?, ?, ?, ?,
(* 210 *)     ?, ?, ?, ?, ?, ?, ?, ?, ?, ?,
(* 220 *)     ?, ?, ?, ?, ?, ?, ?, ?, ?, ?,
(* 230 *)     ?, ?, ?, ?, ?, ?, ?, ?, ?, ?,
(* 240 *)     ?, ?, ?, ?, ?, ?, ?, ?, ?, ?,
(* 250 *)     ?, ?, ?, ?, ?, ?);

  alfa= ARRAY (1..alfalength) OF char;

  mailbox=
  RECORD
    ?: addr;
    ?: ↑mailbox;
  END;

  message_header=
  RECORD
    ?: byte;
    ?: ↑message_header;
    ?, ?, ?, ?: byte;
    ?: integer;
    ?: integer;
    ?: addr;
    ?: ↑mailbox;
    ?: ↑mailbox;
    ?: ↑message_header;
    ?: ↑message_header;
    ?: byte;
    ?: byte;
    ?: byte;
    ?: byte;
  END;

  process=
  RECORD
    ?: ↑niltype;
    ?: ↑process;
  END;

  reference= ↑message_header;

  chain_element=
  RECORD
    ?: reference;
    ?: ↑chain_element;
    ?: ↑chain_element;
  END;
  chain=
  RECORD
    ?: ↑chain_element;
    ?: ↑chain_element;
    ?: 0..maxint;
  END;

  shared_descr=
  RECORD
    ?: reference;
    ?: ↑mailbox;
  END;

  program_descriptor=
  RECORD
    ?: integer;
    ?: ↑niltype;
    ?: integer;
    ?: ↑niltype;
    ?: alfa;
    ?: ↑program_descriptor;
  END;

  programrec=
  RECORD
    programref: ↑program_descriptor;
    ?: ↑niltype;
    ?: integer;
  END;

  adammbxtype= (adammbx, operatormbx, loadermbx, globalmbx,
                loaddrivermbx, paxmbx, restartmbx, ?, ?, ?, ?, ?, ?, ?,
                ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?);

  adamvector= ARRAY (adammbxtype) OF ↑mailbox;

  system_vector= !adamvector;

  process_descriptor=
  RECORD
    timer: integer;
    ?: byte;
    ?: ↑process_descriptor;
    ?: integer;
    ?: addr;
    ?: addr;
    ?: integer;
    ?: integer;
    ?: integer;
    ?: integer;
    ?: ↑program_descriptor;
    ?: ↑mailbox;
    ?: ↑reference;
    ?: ↑process;
    ?: ↑message_header;
    ?: ↑program_descriptor;
    ?: mailbox;
    ?: ↑niltype;
    ?: reference;
    ?: ↑niltype;
    ?: alfa; (* procname *)
    ?: ↑process_descriptor;
    ?: ↑process;
  END;

VAR
  own: process_descriptor;
```

The procname field is initialized by the create routine, and may be changed by the setownname routine.

The values of the predefined type integer constitutes the subrange minint..maxint.

I.1 Implementation Dependent Definitions

The following constants and types are defined in rc3502env:

```
CONST
  break_by_father= 47;

TYPE
  double=
  RECORD
    msp, lsp: integer;
  END;

  coded_date= PACKED
  RECORD
    year_after_1900: 0..127;
    month: 0..12;
    day: 0..31;
  END;

  coded_time= PACKED
  RECORD
    ?: 0..31;
    hour: 0..23;
    minute: 0..59;
  END;

  coded_secs= PACKED
  RECORD
    sec: 0..59;
    msec: 0..999;
  END;

  coded_inc= PACKED
  RECORD
    days: 0..31;
    hours: 0..23;
    mins: 0..59;
    secs: 0..59;
    msecs: 0..999;
  END;

  delaytype=
  RECORD
    prev_date: coded_date;
    prev_time: coded_time;
    prev_secs: coded_secs;
    inc: coded_inc;
  END;

  clocktype=
  RECORD
    date: coded_date;
    time: coded_time;
    secs: coded_secs;
  END;
```

```
lookup_descriptor_segment=
RECORD
  name: alfa;
  descriptor_length,
  no_of_pages,
  pagesize,
  last_page_length,
  kind: integer;
  default_appetite,
  last_param_offset,
  no_of_params: integer;
  date: coded_date;
  time: coded_time;
  source_date: coded_date;
  source_time: coded_time;
  version: integer;
END;
```

J. PREDEFINED ROUTINES

Routine names starting with one or more _ characters are routines used by the system, and cannot be called as user routines. The formal type reforchain indicates that the routine in question may be called with actual parameters of type reference or chain. The type niltype indicates that the type checking is special and known by the compiler.

J.1 Language Intrinsic Routines

```
FUNCTION  abs(n: niltype): niltype;

PROCEDURE alloc(VAR r: reference; VAR p: pool;
          VAR m: mailbox);

FUNCTION  allocdelay(VAR r: reference; VAR p: pool;
          VAR m: mailbox; msecs: integer): activation;

FUNCTION  allocpool(VAR p: pool;
          number, bytes: integer): integer;

FUNCTION  bufcount(VAR r: reference): integer;

FUNCTION  bufsize(VAR r: reforchain): integer;

FUNCTION  bytecount(VAR r: reforchain): integer;

PROCEDURE chaindequeue(VAR r: reference; VAR ch: chain);

PROCEDURE chaindown(VAR ch: chain);

PROCEDURE chainenqueue(VAR r: reference; VAR ch: chain);

FUNCTION  chainlength(VAR ch: chain): integer;

PROCEDURE chainreset(VAR ch: chain);

PROCEDURE chainstart(VAR ch: chain);

PROCEDURE chainup(VAR ch: chain);

FUNCTION  chr(int: 0..255): char;

FUNCTION  create(INSPECT processname: alfa; prog: programrec;
          VAR proc: process; bytes: integer;
          priority: priotype): create_result;

PROCEDURE dec(VAR i: integer);

PROCEDURE _dec_byte_rc(VAR b: byte);

PROCEDURE delay(msecs: integer);

PROCEDURE exception(excode: integer);

FUNCTION  hometest(VAR r: reference; VAR p: pool): boolean;

PROCEDURE inc(VAR i: integer);

PROCEDURE _inc_byte_rc(VAR b: byte);

FUNCTION  link(INSPECT external_name: alfa;
          VAR prog: program_descriptor): link_result;

FUNCTION  locked(VAR m: mailbox): boolean;
```

```
PROCEDURE new(VAR p: ↑niltype);
FUNCTION  nil(VAR r: ↑niltype): boolean;
FUNCTION  _nilref___rc(VAR r: reference): boolean;
FUNCTION  offset(VAR r: reforchain): integer;
FUNCTION  open(VAR m: mailbox): boolean;
FUNCTION  openpool(VAR p: pool): boolean;
FUNCTION  ord(x: niltype): integer;
FUNCTION  passive(VAR m: mailbox): boolean;
PROCEDURE pop(VAR r1, r2: reference);
FUNCTION  pred(x: niltype): niltype;
PROCEDURE push(VAR r1, r2: reference);
PROCEDURE release(VAR r: reference);
FUNCTION  releasepool(VAR p: pool; no: integer): integer;
PROCEDURE remove(VAR proc: process);
PROCEDURE resume(VAR proc: process);
PROCEDURE return(VAR r: reference);
PROCEDURE setbytecount(VAR r: reforchain; val: integer);
PROCEDURE setoffset(VAR r: reforchain; val: integer);
PROCEDURE settop(VAR r: reforchain; val: integer);
PROCEDURE setu1(VAR r: reforchain; val: byte);
PROCEDURE setu2(VAR r: reforchain; val: byte);
PROCEDURE setu3(VAR r: reforchain; val: byte);
PROCEDURE setu4(VAR r: reforchain; val: byte);
PROCEDURE signal(VAR r: reference; VAR m: mailbox);
FUNCTION  stackdepth(VAR r: reference): integer;
PROCEDURE start(VAR proc: process; priority: integer);
PROCEDURE stop(VAR proc: process);
FUNCTION  succ(x: niltype): niltype;
FUNCTION  swap(i: integer): integer;
FUNCTION  top(VAR r: reforchain): integer;
PROCEDURE trace=exception(code: integer);
FUNCTION  unlink(VAR prog: program_descriptor): integer;
FUNCTION  u1(VAR r: reforchain): byte;
FUNCTION  u2(VAR r: reforchain): byte;
FUNCTION  u3(VAR r: reforchain): byte;
FUNCTION  u4(VAR r: reforchain): byte;
```

```
PROCEDURE wait(VAR r: reference; VAR m: mailbox);

FUNCTION  waitdelay(VAR r: reference; VAR m: mailbox;
          msecs: integer): activation;

FUNCTION  _bufsize__rc(VAR r: reference): integer;

PROCEDURE _exchange_rc(VAR proc1, proc2: process);

PROCEDURE _exit_____rc;

FUNCTION  _first____rc(VAR r: reference): integer;

PROCEDURE _heapproc_rc(p: ↑niltype);

PROCEDURE _heapref__rc(p: ↑niltype);

PROCEDURE _initchainrc(p: ↑niltype);

PROCEDURE _initmbx__rc(p: ↑niltype);

PROCEDURE _initpool_rc(VAR m: mailbox;
          number, size: integer);

PROCEDURE _initproc_rc(p: ↑niltype);

PROCEDURE _initprog_rc(p: ↑niltype; q: ↑niltype);

PROCEDURE _initref__rc(p: ↑niltype);

FUNCTION  _last_____rc(VAR r: reference): integer;

FUNCTION  _new______rc(bytes: integer; wordalignment: boolean): ↑niltype;

PROCEDURE _region___rc(VAR sh: shared_descr);

PROCEDURE _reg_exit_rc(VAR sh: shared_descr);

PROCEDURE _shift____rc(a, b: integer);
```

J.2 Implementation Dependent Standard Routines

The following routines are defined in rc3502env:

```
PROCEDURE break(VAR proc: process; excode: integer);

PROCEDURE callremote(VAR r: reference; INSPECT m_name: alfa): integer;

FUNCTION  clock_difference(t1, t2: clocktype): coded_inc;

FUNCTION  clock_increment(t: clocktype; inc: coded_inc): clocktype;

FUNCTION  clock_less_than(t1, t2: clocktype): boolean;

FUNCTION  crc16(op1, op2: integer): integer;

FUNCTION  crc16buf(VAR r: reference; frombyte, tobyte: integer;
          quotient, startvalue: integer): integer;

PROCEDURE definetimer(onoff: boolean);

FUNCTION  deletemailbox(INSPECT mbx_name: alfa): integer;

FUNCTION  eoi: boolean;

FUNCTION  first=offset(VAR r: reforchain): integer;

FUNCTION  getclock: clocktype;
```

```
PROCEDURE inbyteblock(VAR next: integer; first, last: integer;
          VAR msg: reference);

PROCEDURE inwordblock(VAR next: integer; first, last: integer;
          VAR msg: reference);

FUNCTION  last(VAR r: reforchain): integer;

FUNCTION  messagekind(VAR msg: reference): integer;

FUNCTION  namemailbox(VAR m: mailbox; INSPECT mbx_name: alfa):
          integer;

FUNCTION  next(VAR r: reforchain): integer;

PROCEDURE outbyteblock(VAR next: integer; first, last: integer;
          VAR msg: reference);

PROCEDURE outwordblock(VAR next: integer; first, last: integer;
          VAR msg: reference);

FUNCTION  ownname(VAR name: alfa): byte;

FUNCTION  ownprogramname(VAR name: alfa): byte;

FUNCTION  ref(VAR m: mailbox): ↑mailbox;

FUNCTION  reservech(VAR ch_msg: reference;
          level, mask: integer): integer;

FUNCTION  searchmailbox(INSPECT mbx_name: alfa): ↑mailbox;

PROCEDURE sendlinker(VAR r: reference);

PROCEDURE sendtimer(VAR r: reference);

PROCEDURE setfirst(VAR r: reforchain; val: integer);

PROCEDURE setlast(VAR r: reforchain; val: integer);

PROCEDURE setnext(VAR r: reforchain; val: integer);

PROCEDURE setownname(INSPECT name: alfa);

PROCEDURE setpriority(priority: integer);

PROCEDURE tofrom(VAR toref: reference; toindex: integer;
          VAR fromref: reference; fromindex: integer; bytes: integer);

PROCEDURE waiti;

FUNCTION  waitid(msecs: integer): activation;

FUNCTION  waitim(VAR r: reference; VAR m: mailbox): activation;

FUNCTION  waitimd(VAR r: reference; VAR m: mailbox;
          msecs: integer): activation;
```

K. RC3502 SYSTEM SUPPORTED ENTITIES

The following subsections contain the definitions of constants, types, and routines which are supported by the RTP3502 system for the RC3502 machine.

K.1 Doubleenv

```
CONST
  double_min= double(:minint, 0:);
  double_zero= double(:0, 0:);
  double_one= double(:0, 1:);
  double_two= double(:0, 2:);
  double_max= double(:maxint, -1:);

FUNCTION double_add(d1, d2: double): double;

PROCEDURE double_dec(VAR d: double);

FUNCTION double_div(d1, d2: double): double;

PROCEDURE double_inc(VAR d: double);

FUNCTION double_int(i: integer): double;

FUNCTION double_lt(d1, d2: double): boolean;

FUNCTION double_madd(d1, d2: double): double;

FUNCTION double_mod(d1, d2: double): double;

FUNCTION double_msub(d1, d2: double): double;

FUNCTION double_mul(d1, d2: double): double;

FUNCTION double_sub(d1, d2: double): double;

FUNCTION double_uint(i: integer): double;

FUNCTION int_double(i: double): integer;

FUNCTION uint_double(i: double): integer;
```

K.2 Imc3502env

Entities for inter module communication (IMC)

```
CONST
  max_imc_id= 10;

TYPE
  scope_type= (anonymous, local, regional, global, ?);

  portmessage=
  RECORD
    ?: 1..maxint;
    name: alfa;
    scope: scope_type;
    no_of_conns: 0..maxint;
    rcv_all: boolean;
    ?: 1..maxint;
  END;
  port=                          <->
  RECORD
    ?: ↑ mailbox;
    ?: pool 1 OF portmessage;
    ?: mailbox;
    ?: reference;
    ?: reference;
    port_params: !portmessage;
  END;

  event_type=
   (not_event, message_event, answer_event, process_removed, port_closed,
   disconnected, ?, ?, local_connect, remote_connect,
   reset_indication, reset_completion, credit,
   data_sent, data_arrived, data_overrun, ?,
   dummy_lcnct, dummy_rcnct, dummy_rindic,                          ✓
   dummy_rcmpl, dummy_credit, dummy_sent, dummy_arrived, ?, ?);

  reason_type=
   (reason_ok, reason_name, reason_resource,
   reason_closed, reason_network);                  ✓

  conn_service=
   (cs_normal, cs_high); ✓

  alfa3= ARRAY(1..3)OF char; ✓

  alfa9= ARRAY(1..9)OF char; ✓

  masknames_t=
  RECORD
    linkno: integer;
    length: byte;
    name: alfa9;   ✓
  END;

PROCEDURE closeport(VAR p: port); ✓

PROCEDURE connect(VAR p: port; index: 1..maxint;
          VAR compl, disc: reference; remote_name: alfa; ✓
          service: conn_service);

FUNCTION  creditcount(VAR r: reference): 0..maxint; ✓

PROCEDURE disconnect(VAR p: port; index: 1..maxint);✓

FUNCTION  eventkind(VAR r: reference): event_type; ✓

PROCEDURE getconnection(VAR p: port; index: 1..maxint; ✓
          VAR compl, disc: reference);

PROCEDURE getcredit(VAR p: port; index: 1..maxint;  ✓
          VAR credbuf: reference);

PROCEDURE getreset(VAR p: port; index: 1..maxint;  ✓
          VAR indic: reference);

FUNCTION  imcexists(imc_id: 0..maxint): boolean;  %

FUNCTION  index(VAR r: reference): 0..maxint;  ✓

PROCEDURE initport(VAR p: port; imc_id: 0..maxint);  %

PROCEDURE masknames(VAR r: reference; VAR mask: !alfa3;   -> set servers % imc-id
          imc_id: 0..maxint);

FUNCTION  maxconnections: 0..maxint;              ✓

PROCEDURE openport(VAR p: port; VAR closebuf: reference;
          name: alfa; scope: scope_type;
          no_of_conns: 0..maxint; rcv_all: boolean);
                                   <-> control type
```

```
FUNCTION  reason(VAR r: reference): reason_type;

PROCEDURE receive(VAR p: port; index: 1..maxint;
          VAR databuf: reference);

PROCEDURE receiveall(VAR p: port; VAR databuf: reference);

PROCEDURE reset(VAR p: port; index: 1..maxint;
          VAR compl: reference);

PROCEDURE resetevent(VAR r: reference);

PROCEDURE send(VAR p: port; index: 1..maxint;
          VAR databuf: reference);
```

K.3 Ioenv

Entities for text input/output by means of so-called zones.

```
 CONST
   linelength= 80;
   firstindex= 6+alfalength;
   lastindex= firstindex+linelength-1;
 TYPE
   opbuffer=
   RECORD
     first,
     last,
     next: integer;
     name: alfa;
     chars: ARRAY (firstindex..lastindex) OF char
   END;

   zone=
   RECORD
     driver: ↑mailbox; (* operator program *)
     answer: ↑mailbox; (* answers returns here *)
     dataready: mailbox; (* buffers with data *)
     free: mailbox; (* free buffers *)
     cur: reference; (* current buffer *)
     u2val: byte; (* u2 to driver *)
     state: byte; (* resultcode from answer *)
     readstate: integer; (* 0: ok, >0: error, -1: cur=nil *)
     nextp: integer; (* pointer into databuf  *)
     lastpos: integer; (* last position in buffer*)
   END;

PROCEDURE inchar(VAR z: zone; VAR t: char);

PROCEDURE indouble(VAR z: zone; VAR d: double);

PROCEDURE inhex(VAR z: zone; VAR num: integer);

PROCEDURE ininteger(VAR z: zone; VAR num: integer);

PROCEDURE inname(VAR z: zone; VAR name: alfa);

PROCEDURE opanswer(VAR msg: reference; VAR z: zone);

PROCEDURE openzone(VAR z: zone; driv, answ: ↑mailbox;
          bufs: integer; VAR home: pool 1; v1, v2, v3, v4: byte);

PROCEDURE openopzone(VAR z: zone; driv, answ: ↑mailbox;
          bufs: integer; VAR home: pool 1; v1, v2, v3, v4: byte);

PROCEDURE opin(VAR z: zone);

FUNCTION  optest(VAR z: zone): boolean;
```

```
PROCEDURE opwait(VAR z: zone; VAR inputpool: pool);

PROCEDURE outaddr(VAR z: zone; a: addr);

PROCEDURE outalfa(VAR z: zone; VAR text: !alfa);

PROCEDURE outchar(VAR z: zone; t: char);

PROCEDURE outdate(VAR z: zone; date: coded_date);

PROCEDURE outdouble(VAR z: zone; d: double; pos: integer);

PROCEDURE outend(VAR z: zone);

PROCEDURE outfill(VAR z: zone; filler: char; rep: integer);

PROCEDURE outhex(VAR z: zone; num, pos: integer);

PROCEDURE outinteger(VAR z: zone; num, pos: integer);

PROCEDURE outnl(VAR z: zone);

PROCEDURE outtime(VAR z: zone; time: coded_time);

PROCEDURE printmessage(VAR z: zone; VAR r: reference;
          firstindex, lastindex, words_per_line: integer);
```

K.4 Measureenv

Entities for performance measurement of a program.

```
TYPE
  time_stamp=
  RECORD
    hour: coded_time;
    msec: integer;
  END;

  measure_rec=
  RECORD
    access_count: integer;
    name_ptr: addr;
    hours, minutes: byte;
    h_secs: integer
  END;

  measurement_area=
  RECORD
    current, former: time_stamp;
    dif_msec, dif_minute, dif_hour: integer;
    table: ARRAY (0..255) OF measure_rec;
  END;

PROCEDURE trap_code_rc;

PROCEDURE start_measure(VAR mbx: mailbox);

PROCEDURE stop_measure(VAR mbx: mailbox);
```

L. OPERATOR INPUT/OUTPUT EXAMPLE

```
PROGRAM testio1;
  (* Communication with operator     *)
  (* All input is terminated by <NL> *)
  (* Input is echoed on the console  *)

CONST
  readcode= 1;
  writecode= 2;
  message= 7;
  no_of_opbuffers= 2;
  no_of_inbuffers= 1;
  no_of_outbuffers= no_of_opbuffers-no_of_inbuffers;

VAR
  keyboard, display: zone;
  nilmbx: ^mailbox;     -- pointer to mailbox
  t: char;
  homepool: pool 2 OF opbuffer;

BEGIN
  (* Open zone display for output *)
  openopzone(display,
  nilmbx,                -- nilmbx is by default the operator
  nilmbx,                -- no action on return of messages
  no_of_outbuffers,
  homepool,
  writecode,
  message,
  0, 0);
  outfill(display, '.', 10);
  outalfa(display, 'Welcome ');
  outalfa(display, 'to ');
  outalfa(display, own.procname);   -- Incarnation name
  outfill(display, '.', 10);
  outnl(display);                  -- New Line

  (* Open zone keyboard for input *)
  openopzone(keyboard,
  nilmbx,                -- nilmbx is by default the operator
  nilmbx,                -- activated on return of messages
  no_of_inbuffers,
  homepool,
  readcode,
  message,
  0, 0);

  LOOP
    outalfa(display, 'Input <NL>: ');
    outend(display);
    opin(keyboard);
    opwait(keyboard, homepool);
    REPEAT
      inchar(keyboard, t);
      outchar(display, t);
      IF t= NL THEN
        BEGIN
          outchar(display, BEL);
          outend(display);
        END
    UNTIL keyboard.readstate<0;
  ENDLOOP
END.
```

M. EXCEPTION CODES

Code (Hex)	Error Text
01	- parity error
02	- registerstack error
03	- undefined opcode
04	- odd number of bytes
05	- stack overflow
06	- pointer = nil
07	- signal: reference = nil
07	- push: first param = nil
07	- pop: second param = nil
07	- lock: reference = nil
07	- reference = nil
08	- wait: reference <> nil
08	- pop: first param <> nil
09	- push: param locked
09	- pop: second param locked
09	- signal: reference locked
09	- reference locked
0A	- lock overflow
0B	- arithmetic overflow : "lop"-"rop"
0B	- arithmetic overflow : "lop"+"rop"
0B	- arithmetic overflow : "lop"*"rop"
0B	- arithmetic overflow : "lop" div "rop"
0B	- arithmetic overflow : "lop" mod "rop"
0B	- arithmetic overflow : -"op"
0B	- arithmetic overflow : abs "op"
0C	- index out of bounds: "index value"
0C	- subrange out of bounds: "operand value"
0D	- illegal textstate
0E	- field overflow : "operand value"
0F	- move wraparound :
10	- push: identical arguments
11	- push: first param not empty
12	- lock: size error : "buffer size" "new size"
12	- size too small
13	- top <= offset
14	- lock: not data message
14	- not data message
15	- not channel message
16	- word block i/o : odd number of bytes
17	- block i/o at level 0
18	- setcr: first limit negative
19	- setad: truncation error
1A	- no resources
1B	- file does not exist : "file name"
1C	- position outside file
1D	- wrong answer
1E	- setpriority: illegal priority
1F	- pool : no core
20	- process = nil
21	- arithmetic overflow : "ldop" * "rdop"

22	- system error
23	- system error
24	- illegal switch in case construction
25	- upper limit in call of succ
26	- lower limit in call of pred
27	- with: retype overflow : "size of old" < "size of new"
28	- lockdata: top < computed
29	- local reference variable not nil at routine exit
2A	- local process variable not nil at routine exit
2B	- system error
2C	- system error
2D	- system error
2E	- system error
2F	- break by father

N. INDICES

N.1 Survey of Figures

N.2 Catchword Index