DANOSI

Real-Time Pascal

Compiler

Reference Manual

Keywords:
RC5000, RC5010, RC5050, MegaSwitch, iAPXn86, RcPAX, Software.

Abstract:
This manual describes the usage of the MS-DOS implementation of the Real-Time Pascal Compiler for the RC5000 Network Switch, the RC5000 MegaSwitch, and the Intel iAPXn86 controllers.

Date:
September, 1993

Author:
Herluf Hansen and Henning Jakobsen

PN: 99001464

Table of Contents

1. Introduction

1.1 Purpose

The Real-Time Pascal compiler running on the DOS operating system is ment to be the primary tool for making software for three different target computers:

- rc3502 Main cpu of the RC5000 Network Switch.
- ta The RC5000 MegaSwitch controllers with transputer cpu and RTP runtime system (PI-5), i.e. RC5354 (MAC241) and RC5355 (MAC242).
- i86 The iAPXn86 based controllers (e.g. RC5320 (MFC202) or RC5371 (COM206)).

1.2 Prerequisites

The compiler requires:

- MS-DOS 3.3 or later

or

- Concurrent DOS vers. 6.6 or later

or

- UNIX with VP/ix rel. 1.2 or later.

1.3 Components

The compiler program: RTP.EXE and RTP.OVR.
Parameter table: RTPPAR.TAB
Error texts: RTPERR.TXT
Four files with standard environments for each target machine.
A file with compiled open routines for each target machine.

2. Invoking the Compiler

The RTP compiler is called according to the following syntax:

RTP option-list env-list source option-list

Call parameters:

- env-list

  Names of the environment files - the files may contain declarations of constants, types and external procedures and functions. The names must be given in the notation for MS- DOS files.

- source

  The name of the source file specified in the notation for MS-DOS files.

- option-list

  The options which can be be used in the call line are specified in section 3. Option parameters must be given enclosed in parantheses.

It is possible to have some or all the call parameters stored in a text file. If the call line contains a "-" (hyphen) followed by a file name, the contents of the file will substitute the file parameter in the command line.

Examples

RTP target(rc3502) codesize(9000) io.env sample.srp list

RTP -sample.irp

These two calls are equivalent if sample.irp contains:

```
target(rc3502)
codesize(9000)
io.env
sample.srp
list
```

RTP spacing(30) -sample.irp

3. Options to the Compiler

The options to the compiler can be grouped into the following three categories:

- Options used only in the call line when invoking the compiler.

- Options used both in the call line and within the sources compiled.

- Options used only within the sources.

Options in the source must be given with a $ sign in front of them, e.g. $CODE

3.1 Options in the Call Line

- TIME
  To output the duration times of the compiler passes.

- TARGET (<computer type>)
  The following computer types may be used:
  - ta (RC5000 MegaSwitch)
  - rc3502 (RC5000 Network Switch)
  - i86 (iAPXn86)

  (Default is i86)

- CREATESIZE (<number>)
  Determines the default stack size of incarnations of the following program(s) if the value of the parameter size in the function call create is 0, cf. Reference manual section 9.1. Number denotes the number of bytes in the stack.

- CODESIZE (<number>)
  Gives the size of the "program code pages" when generating code for the target computer. Codesize can be changed by the compiler if needed.
  (Default 1000)

- SAVE (<name>)
  Controls the compilation of environments to be saved under the name given, for later use by means of the LOAD option. The source file parameter in the call line must specify an empty program.

- LOAD (<name>)
  Includes environments saved with SAVE under the name given. Only one LOAD may be specified, but the call line may of course also contain further environments.

- START (PASS1)
  Only used when compiling standard environment.

- SPAGESIZE (<number>)
  Page size of the symbol table but rounded to the nearest power of 2 (Default 4096).

- NPAGESIZE (<number>)
  Page size of the name table but rounded to the nearest power of 2 (Default 1024).

- SPAGERES (<number>)
  Maximum number of resident symbol table pages.

- NPAGERES (<number>)
  Maximum number of resident name table pages.

- BREAK
  Inserts break points at the beginning of every program. Can only be used when target computer is i86.

- SPACING (<number>)
  number=0 : only a name table is generated;
  number<>0 : a line table is also generated, the number specifies the amount of bytes of generated code between the line numbers in the line table (Default 50).

- XREF
  If LIST is also given, a cross reference listing is generated (Default with LIST).

- NOXREF
  No cross reference listing is generated.

- LIST
  A program listing is generated.

- IFLIST
  A listing of the proper program is generated, i.e. no listing of code removed by conditional compilation ($IF).

- NOLIST
  No program listing (Default).

- IND
  Indention in the listing (Default).

- NOIND
  The original source indention is not modified in the program listing.

- MARK
  Marking of correlating BEGIN-END pairs in the program listing (Default).

- NOMARK
  No marking in the program listing.

- OWNKEY
  The printing of source keywords is not changed in the program listing.

- LCKEY
  Keywords are printed with small letters in the program listing.

- UCKEY
  Keywords are printed with capital letters in the program listing (Default).

- OWNID
  The printing of identifiers is not changed in the program listing.

- LCID
  Identifiers are printed with small letters in the program listing (Default).

- UCID
  Identifiers are printed with capital letters in the program listing.

3.2 Options in both the Call Line and within the Source Program

- CODE
  Causes code generated for the following lines of source text to be listed.

- NOCODE
  Suppresses listing of generated code (Default).

- INDEX
  Causes checking of subrange constraints before indexing to be included in code generated for the following lines of source text (Default).

- NOINDEX
  Switches index checking off. Has no effect when target computer is rc3502.

- RANGE
  Causes checking of subrange constraints before assignment to be included in code generated for the following lines of source text (Default).

- NORANGE
  Switches range checking off.

- ACCESS
  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 ?).

- NOACCESS
  Switches access checking off (Default).

- OVERFLOW
  Check arithmetic overflow (Default).

- NOOVERFLOW
  No check of arithmetic overflow.

- SET <switch_assignment_list>
  Setting compile-time variables.

For identification of the target computer type the following compiletime variables (switches) are predefined:

- i86
- ta
- rc3502
- target

where target takes the value of the target computer type.

For identification of the language dialects the following switches are predefined:

- rtp3502
- rtp2111
- rtp
- rtp86
- language

Language takes the value rtp.

- PAGELENGTH <number>
  Maximum number of lines per page in the program listing (Default 45).

- PAGEWIDTH <number>
  Maximum number of characters per line in the program listing (Default 115).

3.3 Options within the Source Program

- SEGMENT <name>
  Switches to a new code segment at first following place, where a segment switching can occur. Segmentation may only happen at the outermost level and only before routine declarations or before the statement part of the program or of a routine at the outermost external level. Before the first segmentation only external declarations are allowed. It is only allowed to have one compilation unit when segmentation is used. Has only effect if target is i86.

- EJECT
  Form feed in the program listing.

- TITLE"<character string>"
  The character string is placed in the title field of the header line of each page of the program listing.

- SUBTITLE"<character string>"
  The character string is placed in the subtitle line of each page of the program listing.

- LISTON
  The following lines of the source text will be listed if LIST was given in the call line.

- LISTOFF
  The following lines of the source text will not be listed.

- VERSION <version>.<issue>
  Sets the version attribute of the program.

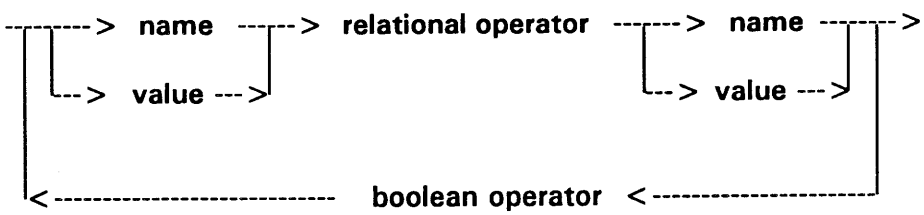
- IF <expression>
  See the Reference Manual (Ref. 2).

- ELSE
  See the Reference Manual (Ref. 2).

- ELSEIF <expression>
  See the Reference Manual (Ref. 2).

- ENDIF
  See the Reference Manual (Ref. 2).

Expressions with IF and ELSEIF are limited by:



boolean operator: AND, OR, XOR
relational operator: =, >=, <=, <>

4. Compiler Output

The compiler produces two output files: the listing file and the object file.

The listing file contains printed output from the compiler and the object file contains the code generated by the compiler.

The listing and the object file has the same name as the source file, except for the extension.

The listing has the extension .LST.

The extension of the object file depends on the target computer type.

Target computer	Extension of object file
i86	.POF
ta	.TAH
rc3502	.T22

5. Extensions to the RTP Language

The RTP compiler provides access to some types and routines which are not part of the RTP language. Their existence are known to the compiler via the standard environment and they do not have to be declared before used.

The following sections contain a listing of these types and routines. Further information about the routines can be found in the Reference Manual for the specific target computers. (Ref. 1 and 2).

Some of the routines are not available on all the target computers. If a routine is not available on all the target computers it is stated in the following listing. For further details see the actual set of routines in the latest released standard environment for the target in question. The IMC functions (Ref. 3, chapter 11) are not implemented on the ta computer.

5.1 General Types and Routines

The element type of a set type must not include negative values.

```
TYPE

    bit = 0 .. 1;

    alfa = string(12);

    coded_date = record
                 year_after_1900: byte;
                 month: byte;
                 day: byte;
                 end;

    coded_time = record
                 hour: byte;
                 minute: byte;
                 end;

    coded_secs = record
                 sec: byte;
                 msec: int16;
                 end;

    coded_inc = record
                days: 0..31;
                hours: 0..23;
                mins: 0..59;
                secs: 0..59;
                msecs: 0..1000;
                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;

tversion = record
            release, subrelease: byte;
            end;

tmoduleversion= record
            moduledate: coded_date;
            moduletime: coded_time;
            moduleversion: tversion;
            compilationdate: coded_date;
            compilationtime: coded_time;
            end;
```

- The mentioned records are packed on rc3502

```
function alias ( var p:port; inspect name: string ): boolean;
```
- Only available on rc3502

```
procedure break ( var proc: process; fault: integer );

procedure breakpoint;
```
- Not available on rc3502

```
function clock_difference ( var t1, t2: clocktype ): coded_inc;

function clock_increment ( var t: clocktype; var inc: coded_inc ):
clocktype;

function clock_less_than ( var t1, t2: clocktype ): boolean;

procedure crc16 ( value, quotient: integer );
```
- Only available on rc3502

```
procedure crc16buf ( var ref: reference;
                     from, tooff, quotient, startvalue: integer);
```
- Only available on rc3502

procedure definetimer (on: boolean);
- Not available on i86

function deletemailbox (inspect mbx_name: string): integer;

function equalhome (var a: reference; var b: reference): boolean;
function equalhome (var a: chain; var b: chain): boolean;
- Not available on rc3502

procedure exception (faultno: integer);

function getclock: clocktype;

procedure getservers (var ref: reference; mask: string);
- Only available on rc3502

function homeless (var ref: reference): boolean;
- Not available on rc3502

function homeless (var r: chain): boolean;
- Not available on rc3502

function hometest (var ref: reference; var p: pool): boolean;

function imcpresent: boolean;
- Only available on rc3502

function locktest (var ref: reference): boolean;
- Not available on i86

function mailbox_state (var m: mailbox): integer;
- Only available on i86.

function madd (a, b: integer): integer;
- Not available on i86

function mmul (a, b: integer): integer;
- Not available on i86

function msub (a, b: integer): integer;
- Not available on i86

function namemailbox (var m: mailbox; inspect mbx_name: string):
integer;

function own_stack_bound: integer;
- Only available on i86.

function own_stack_size: integer;
- Only available on i86.

function ownname (var name:string): byte;
- Result is length of name

function ownpriority: byte;
- Only available on i86.

function ownprogamname (var name:string): byte;

procedure ownversion (var myversion: tmoduleversion);
- Not available on rc3502

function pool_state (var p: pool): integer;
- Only available on i86.

function priority (var proc: process): byte;
- Only available on i86.

function ref (var m: mailbox): ^mailbox;
- Only available on i86.

function rest_of_delay: integer;
- Only available on i86.

function rotate (a, positions: integer): integer;
- Not available on i86

function searchmailbox (inspect mbx_name: string): ^mailbox;

procedure sendadam (var ref: reference);
- Only available on rc3502

procedure sendlinker (var ref: reference);
- Only available on rc3502

procedure sendtimer (var ref: reference);

procedure setownname (inspect name: string);

procedure setpriority (prio: integer);

function stack_bound (var proc: process): integer;
- Only available on i86.

function stack_size (var proc: process): integer;
- Only available on i86.

function swap (n: int16): integer;

procedure tofrom (var toref: reference; toindex: integer;
var fromref: reference; fromindex: integer;
bytes: integer);

function uadd (a, b: integer): integer;
- Not available on i86

```
function udiv ( a, b: integer ): integer;
        - Not available on i86

function ult ( a, b: integer ): boolean;
        - Not avaliable on i86

function umod ( a, b: integer ): integer;
        - Not available on i86

function umul ( a, b: integer ): integer;
        - Not available on i86

function usub ( a, b: integer ): integer;
        - Not available on i86
```

5.2 IO-Routines for i86

The following routines are know to the compiler when target computer is i86.

```
function data_segment ( var r: reference ): integer;
function data_segment ( var r: chain ): integer;

procedure disable;

procedure enable;

function input_byte ( ioport: integer ): byte;

function input_word ( ioport: integer ): integer;

procedure output_byte ( ioport: integer; data: byte );

procedure output_word ( ioport: integer; data: integer );
```

5.3 IO-Routines for rc3502

The following type and routines are know to the compiler when target computer is rc3502.

```
TYPE
        bufparamtype = array ( 1..8 ) of byte;


procedure clearinterrupt;

procedure control ( c: integer; var chmsg: reference );

procedure controlclr ( c: integer );

function ctrwaitid ( c: integer; ms: integer ): activation;
```

```
function ctrwaitim ( c: integer; var r: reference; var m: mailbox ):
activation;

function ctrwaitimd ( c:  integer; var r: reference; var m: mailbox;
                                                ms:    integer          ):
activation;

function eoi : boolean;

procedure getbufparam ( var bufparam : bufparamtype;
                                first, last: integer; var msg: reference );

procedure inbyteblock ( var next : integer;
        first, last: integer;
        var msg : reference );

procedure inword ( var word: integer; var chmsg: reference );

procedure inwordblock ( var next : integer;
        first, last: integer;
        var msg : reference );

procedure inwordclr ( var word: integer );

procedure iowbwc ( bufparam : bufparamtype );

function messagekind ( var r: reference ): integer;

procedure outbyteblock ( var next : integer;
        first, last: integer;
        var msg : reference );

procedure outword ( word : integer; var chmsg: reference );

procedure outwordblock ( var next : integer;
        first, last: integer;
        var msg : reference );

procedure outwordclr ( word : integer );

procedure sense (    var status: integer; status_out : integer;
                                        var chmsg: reference );

procedure senseclr ( var status: integer; status_out : integer;
                                                compare, mask: integer );

procedure setinterrupt ( var ch : reference );

function timedout: boolean;

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;
```

6. Memory Layout

In this chapter it is explained how values of various types are represented on the target computers.

6.1 Target ta

6.1.1 Not Structured Type

6.1.1.1 Ordinal Types

The representation of a value of an ordinal type is the two's complement representation of the value. If more than one byte is allocated to store a value the byte with the lower address contains the least significant bits.

6.1.1.2 Shielded and Pointer Types

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

6.1.1.3 Set Types

Values of set type are represented as a bit vector. If value "i" is a member of the set, the bit with index "i" in the bit vector is 1, otherwise it is 0. If more than one byte is allocated to store a bit vector, the byte with the lower address contains the bits with the lower indices. Within a byte the least significant bits contain the bit with the lower index.

6.1.1.4 Type Size

The following table gives the storage allocated at run-time for each type. For those types which are candidates for packing the minimum number of bits needed to store values of the corresponding data types are stated in the table. If the minimum number of bits are not given the types are not candidate for packing.

Type	Bytes	Bits
Boolean	1	1
Chain	20	-
Char	1	8
Integer	4	-
Int16	2	16
Int32	4	-
Mail Box	16	-
Reference	12	-
Pointer	4	-
Pool	24	-
Port	4	-
Process	8	-
Program	44	-
Scalar (n elements)		
n< =255	7	log (n+1)
n>255	4	log (n+1)
Set of n..m	(m+1)div8	-
Subrange n..m		
Static		
0< =n,m,< =255	1	log (m+1)
n<0,m<0	4	log (-n)+1
n<0,m> =0	4	log (max(-n,m))+1
else	4	log (m+1)
Dynamic	4	-

The number obtained by log must be rounded up to obtain an integral number.

6.1.2 Structured Type

All structures are byte-aligned and occupy an integral number of bytes. Values of record and array types are represented as a concatenation of the components. The first component occupies the lowest address.

6.1.2.1 Not packed

Each component is byte-aligned and occupies an integral number of bytes. The size of a record type is the sum of the size of the fields. The size of an array type is "number of elements" * "size of elements".

6.1.2.2 Packed

In the representation of a packed array and record consecutive components may be packed into one or more bytes if they are candidates for packing. Components not candidates for packing are byte-aligned and occupy an integral number of bytes.

Packing of components always starts from the least significant bit of a byte and each component is allocated as many bits as indicated by its size in the table in sec. 6.1.1.4. 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.

6.1.2.3 Packed3502

The representation of packed3502 array and record is the same as the representation of packed array and record when the target computer is rc3502 (described in ref. 2).

6.1.3 Layout of Variables

Memory for variables declared in a program or a routine block is allocated in the process stack to which the variables belong.

An integral number of words is allocated for each variable. The number of words is determined by the memory requirement of the type of the variables.

6.2 Target rc3502

For information about memory layout in the RC5000 Network Switch, see ref. 2.

6.3 Target i86

Follows the conventions for the Intel iAPXn86 processors.
An integer is 2 bytes.

7. Target Runtime-system Differences

The RTP compiler can generate code for different types of target computers. Minor difference in the runtime-system on these target computers does that the same RTP facilities acted:

- different, depending of the target computer,
- incorrectly according to the RTP language.

The programmer should be aware of the following points:

- Push/pop
  On the rc3502 target computer (RC5000 Network Switch) push and pop will not change the message attributes when an empty message is pushed/popped.

- Createsize
  On the i86 target computer (iAPXn86) the parameter 'size' in the routine create gives the size of stack in paragraphs (not in bytes).

- Searchmailbox
  On the i86 target computer (iAPXn86) searchmailbox will make a global search after the named mailbox.

- Process priority
  On the ta target computer (RC5000 MegaSwitch) the RTP processes will always run with the same priority regardless of how the priority of the process has been set.

A. References

1. The PI3 Machine, Reference Manual.
RCSL No.: 42 - i2307

2. Reference Manual for RC3502 Real-Time Pascal.
PN: 99001174

3. Reference Manual for the Programming Language
Real-Time Pascal.
PN: 99110141