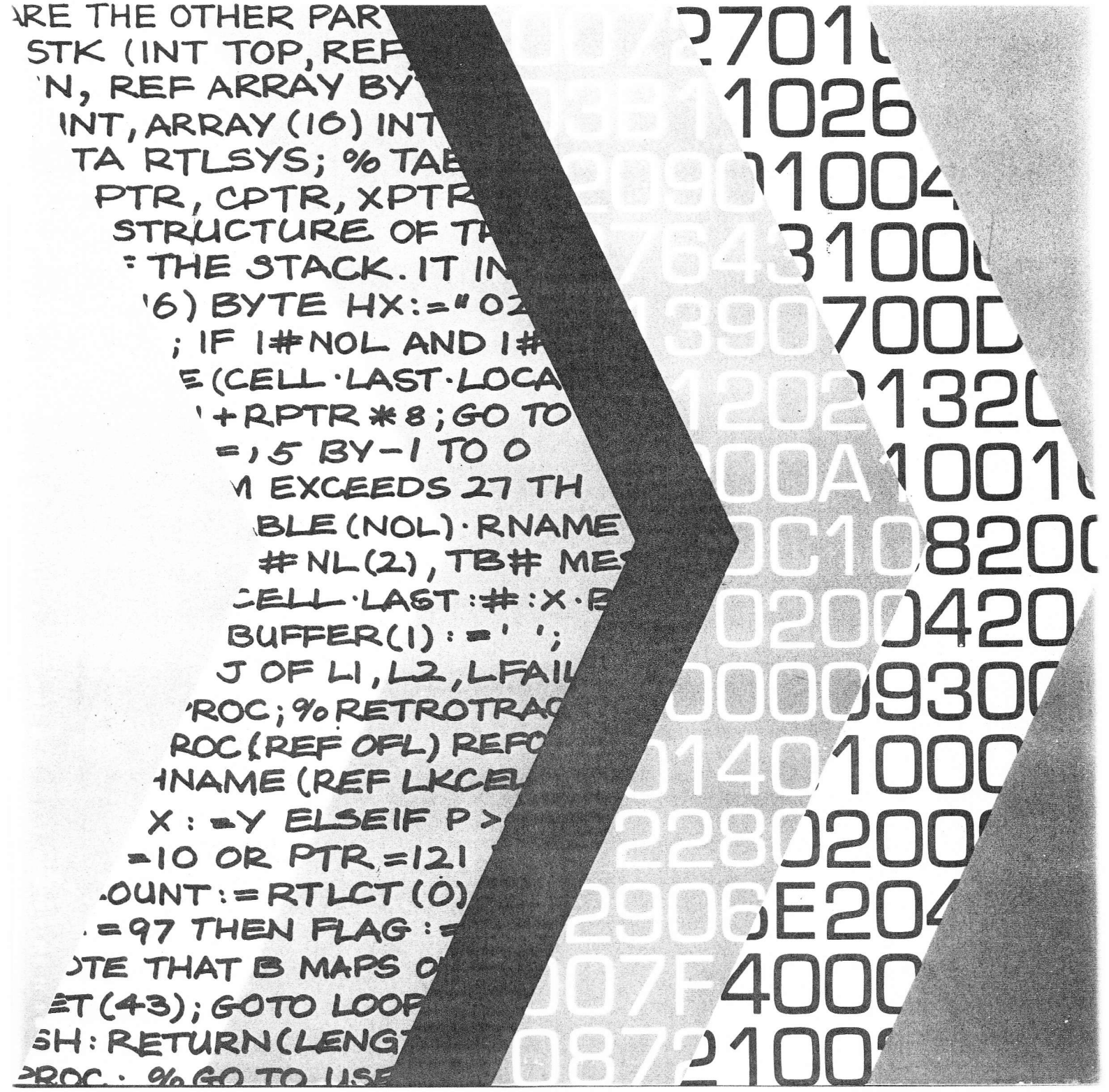
RTL/2

P800 DOM RTL/2 User Manual



PHILIPS P800 DOM RTL/2 USER MANUAL

RTL/2 Reference: 73 Version: 2

Authors: A.P. Porter
G.C. Stevenson

Date: 14th March 1977

Philips Data Systems Publ. Nr. 5122 991 28112

©
Systems Programming Ltd.
1970
All rights reserved

PHILIPS P800 DOM RTL/2 USER MANUAL

INDEX

PHILIPS P800 DOM RTL/2 USER MANUAL

SECTION 1

INTRODUCTION

CONTENTS

1.1 RTL/2 UNDER DOM

RTL/2 is a compact, easy to understand, machine independent, high level systems programming language developed by ICI and marketed by SPL International. DOM is a single-program operating system developed by PHILIPS for the P800 range of computers.

Because RTL/2 is a system programming language, e.g. has part-word manipulation features as standard, it can be regarded as a direct competitor to assembly language for low-level programming. It is also suitable for writing applications programs being simple, easy to learn and lending itself to good structural design. It is pitched at a slightly higher level than FORTRAN. Unlike many systems languages, it supports real arithmetic.

The aim has been not only to supply a compiler, but to incorporate many other features into a package of software useful in generating systems. For this reason the package contains two free-standing utility programs and several items of run-time support software for the user to incorporate in his own system.

The utility programs are released in object format together with catalogued procedures for building the programs into the user's system. The run-time support software is released in source format so that the user may either modify it to suit his own needs or use it as an example for a private implementation.

We have tried to pitch our support at two levels. Because DOM I/O is complicated we have supplied a stream I/O support package which hides the operating system almost completely. If the user follows instructions he will be able

to write programs which perform I/O without having to 'learn' DOM at all. On the other hand, if the user needs the full power of the operating system at his disposal he can, once he has become familiar with the system facilities, call upon the monitor directly from his RTL/2 program without dropping into code. As with all independently designed, complicated entities, the interface is not perfect, but with two levels of compromise we hope to satisfy most requirements. The user may always write new interfacing software if he wishes.

The host machine configuration must support the DOM operating system. The limiting factor on core size is the compiler which needs 20 K words. The RTL/2 software does not require the P857 floating point instructions, because they can be simulated by run-time routines. The compiler can generate floating point instructions if they are available.

1.2 CONTENTS OF THE RELEASE PACKAGE

The RTL/2 system consists of two utility programs and a number of items of run-time support software. The utility programs are supplied in object format together with catalogued procedures for building them on the host system. The run-time support software is supplied in source, RTL/2 or assembler.

1.2.1 The Utility Programs

1.2.1.1 The RTL/2 Compiler

The compiler generates three types of output from RTL/2 source input. Firstly it generates assembler source from the RTL/2 source input. Secondly the user may have produced a line numbered listing of the RTL/2 source. Thirdly, an output file which contains cross-reference information, that may be used by the next program, the Linkage Verifier, to perform inter-module consistency checks.

The compiler's syntax analyser is identical to that utilised by all other RTL/2 compilers, guaranteeing that there is only one legal version of the language. Most of the code generation section is common to other RTL/2 compilers generating P800 object code,so a module compiled, say, on an IBM System/370 will be compatible with the P800.

The Compiler is described in Section 2.

1.2.1.2 The Linkage Verifier

This program is used to check that references between compilation modules are consistent, i.e. that ENT and EXT

brick specifications match. To exploit it, the compiler must be used to generate cross-reference files for every module which will ultimately be link-edited together. These cross-reference files are submitted to the verifier, which performs the check and prints relevant diagnostics if a mis-match is found. The RTL/2 compiler performs intra-modular consistency checks, while the linkage verifier enhances the error detection facilities by performing inter-modular checks across separately compiled modules.

The Linkage Verifier is described in Section 3.

1.2.2 The Run-Time Support

There are a number of items in the package. Some of these are concerned with the fundamental RTL/2 environment and are mandatory. Others may be used as required.

1.2.2.1 The 'Base Program'

This is constructed from modules performing a variety of functions. The primary function is that of initialising the stack for a task prior to entering the RTL/2 Compiler generated code. The program is entered directly from the operating system when a program is run, and returns control to the monitor when the program exits.

The Base Program performs various initialisations and provides system standard functions such as RRNUL and RRGEL. Section 4.2 describes the base program modules in detail.

1.2.2.2 Control Routines

These provide intimate run-time support to the compiled object code. Operations which would be too bulky as in line code are performed by the control routines. Procedure entry and exit, global GOTO's, array bound checking and instruction simulations are the main categories. The latter is a particularly important area since the compiler can generate code to handle various operations in line or by control routine call. Provided the routines are in the object library, they are selected automatically by the DOM linkage editor.

The use of the control routines is described in Section 4.4. Further technical information is contained in RTL/2 Reference 69, The RTL/2 Run-Time Environment on the Philips P800 Series.

1.2.2.3 Stream I/O

RTL/2 Standard Stream I/O (ref. 5) specifies a mechanism for character stream I/O and define a set of formatting routines

based on it. This set of procedures has been augmented by a set peculiar to the DOM system for opening and closing files.

Stream I/O is introduced in Section 4.3 and Appendix II describes the procedures in detail. It is a useful package since the basic facilities it provides makes DOM easier to use, especially for the newcomer to DOM.

1.2.2.4 DOM Interface Library

The stream I/O support package is designed to give the user an easy way of performing I/O operations. Obviously, the implementation will not necessarily be very efficient and is not intended to cover every user requirement. Non-I/O operations are not supported at all. For this the user must go to the operating system more directly. To help him to do this without having to drop into code, a library of RTL/2 procedures has been created which call the operating system directives on a one-to-one basis.

In order to utilise these procedures sensibly, the user must be familiar with DOM. The use of RTL/2 cannot hide the underlying complexity. Section 4.5 contains a description of the interfacing philosophy and Appendix I describes each procedure in detail.

1.3 CONTENTS OF THIS MANUAL

Each program in the package has a section of its own. This contains a general description, a specification of its inputs and outputs, including error messages, and operating instructions under DOM.

Run-time support has a section to itself. Most of the technical details have been put in appendices to enhance clarity.

Finally, a section has been devoted to user program generation.

PHILIPS P800 DOM RTL/2 USER MANUAL

SECTION 2

THE RTL/2 COMPILER

CONTENTS PAGE

2.1 GENERAL DESCRIPTION AND FRONT-END INFORMATION

2.1.1 Description

An RTL/2 Compiler is a program (written in RTL/2) which accepts as input modules of RTL/2 text and produces some form of object code (machine code for assembly or binary for direct loading or linking) for a particular machine - the object machine - on which the modules are to be run.

The machine on which the compiler itself is running may or may not be the object machine and is referred to as the host machine.

An RTL/2 Compiler is formed from four major parts, brief descriptions of which are given below.

Front-End

The front-end is a machine-independent set of modules which performs all the necessary syntax checks on the source text and creates a version of the module in a machine-independent format. It produces a list of error messages and other relevant information. Only one version of the front-end is required to compile RTL/2 on any host machine for any object machine.

Back-End

If no errros are discovered by the front-end, its output is passed to the back-end which produces the object code program and which may optimise the code. It is independent of the host machine, but is clearly dependent on the object machine and may also be dependent on the particular system on the object machine under which modules are to run.

Interface

This collection of modules provides a communication area between the front-end and back-end, and means of accessing the various data relating to the compilation. In addition to specifying

various table sizes, an implementation can use the host machine's capabilities to organise the data and access to it in an efficient manner. This compiler contains a version of the interface the restrictions of which are described in the following pages.

Base

The base program is not part of the compiler but controls the running of it, calling the various phases and performing any overlaying. It creates the necessary environment for the compiler, including the provision of I/O procedures, file handling for source text and object code, the presentation of compile-time options to the compiler and the presentation of the error messages and other information to the user.

2.1.2 Front-End Restrictions

The following limitations (mainly of size) are imposed by the front-end. They will not normally cause any problem for the average program. Further restrictions are imposed by particular implementations and details of these may be found elsewhere in this document.

i) Name : maximum of 31 characters

ii) Constants : a) maximum integral value is $2^{224}-1$
maximum of some 67 significant decimal digits
b) -128<number of decial places - exponent <127
c) -128< binary scale < 127

iii) Options : integer key must be in range 1,15

iv) Arrays : a) maximum dimension is 16
b) maximum bound is 32767

v) Repetition factors : must be in the range 0,32767

vi) Blocks : a) maximum number is 255
b) maximum depth of nesting is 15

vii) External References : maximum number of EXT bricks referenced from within a brick is 50

viii) External Specification : maximum number of distinct MODE names used in an externally known brick is 15

ix) Security : some scope checks cannot be performed absolutely and potential danger can only be warned

Other restrictions involve the maximum depth of nesting of statements, the complexity of expressions and the complexity of modes, but no simple rules can be given.

2.1.3 Front-End Error Mechanism

The error messages produced by the front-end are intended to be self-explanatory and to enable a syntactically correct module to be produced with a small number of compilation runs. For each error detected, two pieces of information are generally given:

i) The line number on which the error was detected; this line count includes blank lines. This line number should be exact, but inaccuracies can occur when the failure is detected at a newline character or in a complex statement or expression spread over more than one line.

ii) An error number and/or an error message. This identifies the nature of the failure. A table of numbers and corresponding messages is given below.

Three types of failures are distinguished:

I) Catastrophic Errors: as the name implies, it is not feasible to continue attempting to compile, and processing of the module is aborted. Clearly only one such message can occur. Such messages indicate an internal compiler fault or a violation

of some size limitation. Normally only the line number and message will be given. It is important that internal errors are reported to the RTL/2 support team. A copy of the source text and the error and line number will suffice.

II) Program Errors: this is the main class of error. The compiler takes some recovery action and continues to process the module; but entry to the back-end is inhibited. As the front-end is a multi-pass process, the recovery action may give rise to side-effect errors later, but detailed consideration of the recovery action should only be necessary when an error message cannot immediately be understood. The general philosophy here is to reduce to a minimum the number of side-effect or repeated errors whilst enabling the compiler to proceed safely and report as many errors as it can find.

III) Warning Messages: these can be suppressed by the use of suitable options (but this should only be done if really necessary) and do not inhibit compilation; no recovery action is necessary.

Recovery

Various recovery actions are taken for program errors (II above); these normally involve ignoring the subsequent source text until a suitable re-start point is found. In particular matching keywords are used in this, and an attempt made to ensure that matching is complete by inserting any missing keywords. It is possible that this will be done at the wrong point and this may lead to side-effect errors.

The broad classes of recovery action are detailed below and referred to by their number in the later table of failure messages. Where more than one is given (or "various") the recovery depends on the level of processing (basically brick level, data or procedure brick):

1. No recovery; processing continues at the next symbol; missing keywords may be flagged and inserted at this point.

2. Recover to the next sensible item at brick level (PROC, STACK, DATA, ENDPROC, ENDDATA, SVC, EXT, ENT, MODE, LET, TITLE, OPTION). Note that the dual use of PROC, STACK can lead to side-effect errors.

3. Recover as 2 or to a semi-colon and continue processing data brick declarations; intervening declarations will not be processed and may give rise to subsequent "identifier not declared" messages.

4. Recover as 2 or to a semi-colon (processing any further local declarations with the same warning as in 3) or to a statement delimiter, i.e. RETURN, GOTO, SWITCH, FOR, TO (when not in FOR statement), IF, WHILE, BLOCK, REP, END, ENDBLOCK.

5. Recover to a statement delimiter or semi-colon and continue processing statements or initialisations as appropriate. Note that in skipping to IF it will always be treated as opening a statement, so that if it is textually opening an expression further (spurious) errors are likely.

Note that errors in LET definitions may cause side-effect errors because occurrences of the name may be replaced by null or erroneous sequences. LET defined names are removed on replacement and may not occur as the "last identifier" in a message; the current symbol can also be confusing in these error circumstances.

It is also helpful to remember that the method of driving the compiler leads it to expect certain items or sequences; a message reporting something missing may seem ludicrous when that item is obviously present on that line; however, examination of the supposed structure at, or just before, this point should reveal the source of the error.

2.1.4 FRONT-END FAILURE MESSAGES

NO.	MESSAGE	RECOVERY	EXTRA INFORMATION
CATASTROPHIC ERRORS			
1	Program too long	-	
2	Blocks nested too deep	-	
3	Too many blocks	-	
4	Statement structure nested too deep	-	
5	Compiler error - please report	-	
6	Mode too complex	-	
7	Compiler error - please report	-	
8	Compiler error - please report	-	
10	Compiler error - please report	-	
11	Expression nested too deep	-	
12	Compiler error - please report	-	
20	Too many name characters	-	
21	Too many names	-	
22	Too many identifiers	-	
23	Mode information overflow	-	
24	Array pool overflow	-	
-	Too many failures	-	This may be reported; see interface details
PROGRAM ERRORS			
101	Non-RTL character	1	Treated as layout and hence may terminate item
102	Illegal stringchar	1	Can give later length mismatches
103	Newline/tab in string or " missing	1	Concatenation still attempted with adjacent strings
104	Illegal symbol in inner sequence / # missing	1	(i) Not a byte constant or repetition factor syntax error (ii) If fails on " then # missing probably & concatenation attempted
105	Mismatched # symbols in string	1	Source representations differ.
106	String pool overflow - null string inserted	1	Later strings may still get into pool. Can cause length errors
107	Terminator in item	1	Module reading terminated: any subsequent text not processed
108	Newline in comment/ % missing	1	
109	Non stringchar / ' missing	1	Byte constant inserted; newline and terminator treated as such
110	: # illegal	1	: # : assumed which may give side errors

NO.	MESSAGE	RECOVERY	EXTRA INFORMATION
111	Illegal macro-item	1	Keyword is ignored
112	Compile-time option error	1	Subsequent opitems ignored
113	Option syntax error	1	Refers to option in text; subsequent opitems ignored
114	Code in applications	1	Treat as if systems module to give maximum error information
115	Digit missing	1	
116	Constant too long	1	Remaining digits skipped: may cause later mode errors
117	Illegal exponent	1	
118	Exponent/binary scale out of range	1	
119	Illegal binary scale	1	
120	Illegal repetition factor	(i)1	Inner string sequence ignored: could cause length errors
		(ii)3	Array initialisation
121	) missing	various	Can lead to length, mode and parameter errors
122	Name too long	1	Subsequent alpha-numerics ignored
123	Constant pool overflow-zero inserted	1	Zero is of same mode so later mismatches should not occur
124	Identifier missing	various	Whenever a name is expected; can cause side errors
125	Identifier is keyword	1	(i) RTL encountered outside code item; removed (ii) Attempting to define LET
126	= missing	1	After LET identifier
127	Identifier already declared	various	Subsequent identifiers in a list will not be declared. This message also used in mismatch situations for redundant EXT/ENT.
128	Code section error	NIL	(i) Parameter syntax error; (ii) layout following trip character; (iii) No name after second trip; (iv) no identifier preceding brick name
129	Illegal item in data brick	NIL	Only occurs on recovery from failure
130	Illegal item(s) at brick level	2	Only one such message of a group of items are misinterpreted
131	; missing	NIL or 2	Syntactic semi-colons assumed
132	EXT/ENT descriptions mismatch	(i) 3	Data brick case: more than one message likely
		(ii)4	ENT procedure specification
133	Stack length missing/not integer	2	
134	(missing	(i) 2	After MODE name: component names not declared
		(ii)4	Literal procedure becomes PROC()
		(iii)various	Syntactic omission can lead to mode mismatches
135	Illegal result mode	2,3,4	Recovery depends on situation
136	Illegal mode specification	2,3,4	May be mode missing or use of REF REF; leads to non declared names
137	Array/record declaration illegal	2,3,4	
138	Bound illegal/too big	2,3	Can cause length mismatch
139	Too many bounds	2,3,4	

NO.	MESSAGE	RECOVERY	EXTRA INFORMATION
140	: = missing	NIL	Only possible after FOR-loop control variable
141	DO missing/illegality in loop heading	4,5	
142	Empty mode	2	Mode name declared
143	Illegal initialisation	(i)2	Within MODE definition } attempting to initialise label
		(ii)4	Within PROC parameters } variables in DATA not failed here
144	Illegal item in declaration	various	A semi-colon may be inserted which may cause side-errors
145	Misplaced ENDPROC	3	Two messages if ENDPROC caused original error (in data)
146	Illegal item at statement level	4,5	
147	REP missing (matching TO)	NIL	
148	REP missing (matching FOR)	NIL	
149	ENDBLOCK missing	NIL	The appropriate keyword is inserted: this may be in the wrong
150	REP missing (matching WHILE)	NIL	place and cause later errors. Can also be side-effect errors
151	END missing	NIL	caused by misplaced keywords.
152	ENDPROC missing	NIL	
153	ENDDATA missing	NIL	
154	Illegal use of :	4	No label identifier: meant for ; or compound item ?
155	Misplaced REP	4	Item removed: this can give rise to or be caused by "XXX missing"
156	Misplaced END	4,5	messages or other errors particularly in case of END and
157	Misplaced ENDBLOCK	4	expression/statement ambiguity
158	Variable not initialised	NIL	Declaration is still successful
159	Identifier not declared/out of scope	(i)NIL	in code sequence
		(ii)5	Use of variable in dynamic initialisation before its declaration
		(iii)3,5	general access of variable name
160	Too many modes in specification	NIL	Unlikely; too many MODE example in external specification
161	Expression of wrong mode	3,5	LHS in multiple assignments or RHS expression or initialisation. Can be a side-error. It is common and includes wrong mode terms supplied to operators and passing SVC name to variable
162	Illegal byte constant	3	+ or - sign supplied
163	Array length mismatch	3	Can be side error in initialisation
164	Illegal item in initialisation	3	
165	External references list full	3,5	Too many external bricks referenced by local brick
166	Identifier not in module	3	Reference to variable in SVC data brick in data initialisation
167	Identifier of wrong mode	3,5	Attempting to access brick or component in expression
168	Invalid selector for record	3,5	
169	Non integer subscript	3	Must be constant in data

NO.		RECOVERY	
170	Array bound violation	3,5	Constant subscript case only
171	Dereferencing in data	3	
172	FOR control variable read only	5	Includes passing to reference
173	Too many subscripts/) missing	5	Always fails on comma
174	Not a procedure	5	Operator comitted before (?
175	Parameter mismatch	5	Should not occur and may indicate compiler fault
176	Too few parameters/ , missing	5	
177	ELSE missing/expression ends incorrectly	5	
178	END missing/expression ends incorrectly	5	
179	Illegal item in expression	5	
180	Illegal use of VAL	5	Includes cases where dereferencing is automatic
181	:= missing/destination invalid	5	
182	Host brick error	NIL	(i) brick name absent or unnecessary for brick (ii)variable does not belong
183	; missing/statement ends incorrectly	5	Can cause side-errors on recovery
184	Illegal use of ELSEIF	5	May be logically impossible (already had an ELSE part)
185	DO missing/condition ends incorrectly	5	
186	THEN missing/condition ends incorrectly	5	
187	RESULT not returned	NIL	ENDPROC reached without result; can occur as side effect of inserted keywords; also if each branch of an IF statement contains a RETURN statement - compiler not clever enough!
188	OF missing/illegal item in switch	5	
189	Illegal label in switch	5	Labels must be literals
190	TO missing/illegality in loop heading	5	
191	Illegal use of ELSE	5	May be logically impossible (already had an ELSE part)
192	Bracketed comparison	NIL	Syntactic error: may also have ") missing "= logical error
WARNING MESSAGES			
201	Taking out of scope	-	Any potentially dangerous action with non-plain variables; in particular passing of results and assignments of locals to globals (especially label variables) will be flagged
202	Floating constant at run-time	-	Run time float inefficient; ensure that if this constant is supplied via a LET definition, all uses are the same mode
203	Procedure result lost	-	Check function is used for side-effects only
204	Silly TO loop	-	Limit is 0 ; executed zero times via run-time check

2.2 P800 BACK-END INFORMATION

2.2.1 Options

In addition to the standard 'opitems' (NW, NS, BC and BS) applicable to all RTL/2 implementations, further opitems specific to this compiler are available. In any OPTION not containing a particular opitem, the default value will be taken. The full set of opitems is as follows:

Opitem	Interpretation	Default Value
NW	inhibit warning messages	all warning messages given
NS	inhibit scope warning messages	all scope warning messages
BC	array bound checks applied in 'safe' cases (gives full checks in application language)	array bound checks omitted in 'safe' cases
BS	array bound checks applied in 'unsafe' cases in systems language	array bound checks omitted in 'unsafe' cases in systems language
NC	minimal comments only generated in object code	full comments generated
TR	run time line number monitoring	no run time line number monitoring
FP	coding to use P857 floating point unit will be generated where applicable	all floating point by control routines
CM,QK	ignored	
OV	fixed-point overflow detected by control routines	overflow ignored

2.2.1 Cont..

Notes:

NW and NS should only be used where a module cannot be compiled because the messages overflow the message pool.

CM and QK have been included for compatibility with earlier compilers.

BC and BS give the user control over two types of run time array bound checks. A 'safe' case is one where even if the bounds are violated no corruption of the software structures can occur - an example is reading out of an integer array. An 'unsafe' case is one where corruption can occur - any store operation is unsafe and reading from an array of references is also unsafe because the value obtained may, when used as an address, cause corruption.

In the applications language checks are always applied in the unsafe case; BC allows the user to apply them in safe cases also. In the systems language checks are not normally applied; BS allows the user to apply them in the unsafe cases to bring security up to the level of the applications language and BC allows the user to apply the remaining checks.

It should be noted that in the case of a constant subscript of an array accessed directly (that is not via a ref array variable) checks are always carried out at compile time and never at run time.

OPTIONs may be altered at Compile time as detailed in section 2.4.2. However, it is only possible to affect OPTION statements within the source program, and it is therefore good practice to include at least one OPTION statement in every RTL/2 module.

2.2.2 CODE sequences

The syntax follows the overall standard as described in the RTL/2 Specification Manual thus:

```
Codeseq      ::=  codeheading codeitem....

Codeheading  ::=  CODE digitlist, digitlist;

Codeitem     ::=  ISO7-character-other-than-& on @
                  |& variable-name| @ data-brick-name
```

Thus in this implementation the characters 'trip 1' and 'trip 2' of the specification manual are & and @ respectively.

The two values denoted by "digitlist" in the heading denote, in bytes, the core space required by the code itself and additional stack workspace required at run time.

The forms & variable-name and
& variable-name @ data-brick-name
are transformed by the compiler as follows:

RTL/2 TEXT	CORRESPONDING ASSEMBLER
&integer	literal value in hexadecimal
&fraction	"
&string	symbolic address of conceptual zero element of string in pool
&name	label followed by colon
&&, &@	&, @ respectively
&identifier	depends on use of identifier as below
&modename	literal value of length of mode in decimal
&brickname	symbolic address of start of brick
&literal-label-name	symbolic address of label
&localname	displacement of variable from current linkcell
&component@mode	displacement of component from start of record
&globalname@databrick	symbolic address of variable
$$	offset from A12 of the lowest addressed workspace word available to CODE section

For further information on code statements and the run-time representation of RTL/2 programs on the P800, see RTL/2 Reference 69, "The RTL/2 Run Time Environment on the P800" (5122 991 2813x).

2.2.3 Back-End Restrictions

The following additional restrictions are imposed by this back-end (independently of those of the front-end). They are unlikely to be encountered in practice.

i) Generated labels (that is labels implied by conditional statements/expressions and repetition statements) : maximum of 32367

ii) Strings: : maximum of 3000 distinct strings in pool

iii) Arrays: : maximum dimension is 8

iv) Blocks: : maximum depth of nesting is 15

v) Procedure calls: : maximum depth of lexically nested procedure calls is 24

There are also natural restrictions on the size of integer and real constants imposed by the nature of the object machine.

Reals have a maximum binary exponent of $\pm$ 32767.

2.2.4 Back-End Error Mechanism

The mechanism employed by the back-end is similar to that of the front-end. Few errors are detected by the back-end - most are internal compiler errors (which should be reported to the RTL/2 Support Team) and violations of the restrictions of the previous section.

For each error detected, two pieces of information are given:

i) The line number as for the front-end.

ii) An error number. Note that the numbers will not be distinct from those used by the front-end and an indication of whether the messages originate from the front-end or back-end will be given if necessary. A table of numbers and corresponding messages is given below.

Three types of failure are distinguished as for the front end (see 2.1.3). The only significant difference is that no particular recovery action is visible to the user in the case of Program Errors. Note that in the case of Catastrophic Errors (which terminate the compilation) and Program Errors (which do not) the assembler module produced by the compiler will be of no value.

2.2.5 BACK-END FAILURE MESSAGES

NO.	MESSAGE	EXTRA INFORMATION
CATASTROPHIC ERRORS		
1	Program too long	
2	Too many generated labels	
3	Too many strings	
4	Compiler error - please report	
5	Array of too many dimensions	
6	Compiler error - please report	
7	Compiler error - please report	
8	Too many levels	Blocks nested too deeply
9	Compiler error - please report	
10	Compiler error - please report	
11		Not used
12	Compiler error - please report	
13	Compiler error - please report	
14	Compiler error - please report	
15	Compiler error - please report	
16	Procedure calls nested too deeply	
17	Compiler error - please report	
18	Compiler error - please report	
19	Compiler error - please report	
20	Compiler error - please report	

2.2.5 Cont...

No.	MESSAGE
PROGRAM ERRORS	
101	Integer constant overflow
102	Fraction constant overflow
103	Real constant overflow
104	Compiler error - please report
107	Array exceeds 32767 bytes
WARNING MESSAGES	
201	Unknown option
202	Real constant underflow
203	Integer too big?

2.2.6 Hints

There is little extra cost in space in applying array bound checks. This is because the routines which perform the checks also do other operations (such as subscript alignment for non byte arrays) which are normally done inline.

The compiler contains certain memory features and, other things being equal, it will remember a simple subscript, array base or record base from operation to operation. Statements using the same bases or subscripts should therefore be grouped together.

Conversions between normal and fine forms of integers and fractions may generate up to 14 bytes per operation.

2.3 CONFIGURATION AND SIZE LIMITATIONS

2.3.1 Installation

The RTL/2 Compiler is built as a self contained program running under DOM. With dynamic memory buffers it requires 20 K of core and optionally a line-printer for source listing, error messages and report. Alternatively, the console can be used for this purpose.

Information on building the Compiler is contained in Appendix III.

2.3.2 Source Module Size Constraints

The maximum limits for the resources are:

Resource	Maximum
Identifiers	400
At brick level	250
Identifier names	250
Name Characters	1500
Generated labels	32367
Constant Pool	1000
String Pool	3000
Array bounds	50
Mode Information Pool	500

A table showing these limits and the resources actually used can be printed at compile-time.

2.4 RUNNING THE COMPILER

2.4.1 Command Format

The compiler is stored in the system library as a load module named RTL. There is a catalogued procedure $RTL in the system procedure file M:PROC which sets up the file environment for the compiler. The compiler can only be run using this procedure.

```
$RTL [FN = file name [,U = userid]] [,CN = control]
[,INFO = n] [,LS = file code] [,OP1 = option] [,OP2 = option]
... [,OP5 = option] [,XREF = file name] [,OB = KPF]
```

2.4.2 Parameters

FN specifies a library source file containing the RTL source to be compiled. If U is specified the file may be in another user's library.

If FN is not specified, the temporary source file /S is taken. However this file will be overwritten by the output so /S must have been saved previously.

CN is set to F [ULL] if the source is to be compiled in systems mode. If CN is omitted or is set to anything else, the applications mode will be assumed.

INFO specifies the degree of reporting (see 2.4.8).

If INFO = 1 only minimal reports are output, titles, errors and a success or failure message.

If INFO = 2 in addition, the use of resources, brick sizes and a list of unidentified variables is output.

If INFO = 3 in addition, a concordance table is output showing the line numbers on which each variable occurs.

If INFO = 9999 in addition, an interphase print is produced. This is a compiler debugging aid.

LS specifies the file code to which the listing of source will be sent. The file code must be assigned. Default is O, (zero) for no listing.

By means of OP1 to OP5, a source statement of the form;

OPTION (n) opitem, opitem, ...

may be modified by specifying the OPTION number in brackets followed by opitems separated by semi-colons. The number in the OPn parameter has no significance.

e.g. a source statement

OPTION (2) NS;

is modified by specifying

OP1 = (2) NW;TR

to OPTION (2) NW,TR;

The source listing, however, is not modified. XREF specifies the name of the data file under which the cross reference data generated by the compiler will be stored. Default, the data is scratched.

If OB = KPF, a command, KPF /O will be executed after assembly, to store the object code in the object library.

2.4.3 Action of the command procedure $RTL

The command procedure uses filecodes /BD, /BE,/BF, /L and /S. They are released after use except for /S. The user should not use these codes for local files. If the compilation is successful, the procedure will assemble the output, and if requested, keep the cross-reference data and object data.

The user may wish to alter the procedure, for example, to change the defaults or to add extra OPn parameters.

The user should note that blank lines and the % signs are significant.

If the compilation is unsuccessful, the command file is skipped until the second % sign is read. This ensures that commands like Assemble or keep cross-reference data are not attempted when no assembler output is produced.

2.4.4 IDENT statement

All RTL/2 modules must have an IDENT statement on line 0 in this form:

␣IDENT␣<name of up to 6 characters>%<comments>%

The number of spaces in the statement is optional, but the form above will enable the source module to be kept in the library under the name in the IDENT statement using KPF /S.

This statement is copied to the assembler output file in the form:

\IDENT\ <name>

Care should be taken to keep the assembler output in a different library file from the RTL/2 source, if the output is required.

2.4.5 Examples

To compile PROG with a listing to lineprinter, in applications mode, use;

```
$RTL FN=PROG,LS=2,XREF=XPROG,OB=KPF
```

If, however, the compilation fails because of a syntax error, the user might use the following sequence of commands;

```
LED PROG,/S
   - - -
!! EN
KPF/S
$RTL INFO=1,XREF=XPROG,OB=KPF
```

2.4.6 Workfile

The compiler uses a random access workfile of up to 80 sectors. The sectors are allocated one at a time during compilation. If the disc becomes full, compilation is abandoned. Filecode /BE is assigned to the workfile while the compiler is running.

2.4.7 Environmental Errors

These are described in Appendix VI.

2.4.8 Report Format

The compiler, during the compilation of programs, outputs a report to filecode 2. This report in the full form contains:

(a) Compiler identity.

(b) The source program with line numbers, if LS ≠ 0.

(c) The operands of RTL/2 TITLE statements at their point of occurrence.

(d) Compiler error diagnostics and warning messages in the format:

$$\left\{\begin{matrix}F\\B\end{matrix}\right\} \langle\text{error number}\rangle \quad \text{LINE} \quad \langle\text{line number}\rangle$$

where F or B is printed to indicate whether the compiler Front or Back end found the error. The significance of the <error number> is defined in section 2.1.4 of this manual if it originates in the front end or section 2.2.5 in the case of the back end. The <line number> refers to the number of the line, in the source program, where the error is detected.

(e) A warning if compiled as systems module.

(f) A resource table is INFO > 1.

(g) A concordance table of identifiers and their line numbers if INFO>2.

(h) If the compilation is successful, a list of brick names and sizes if INFO>1.

(i) If the compilation fails due to a fatal environmental error (described in Appendix VI) or a standard error (e.g. array bound check or stack overflow), the message:

"FAILS <error number> "

(j) The Final Message "COMPILATION OK" or "COMPILATION FAILS" which is also printed on the console.

PHILIPS P800 DOM RTL/2 USER MANUAL

SECTION 3

THE RTL/2 LINKAGE VERIFIER

CONTENTS PAGE

3.1 INTRODUCTION

A runnable RTL/2 program is produced in three stages:

(1) Compile RTL/2 modules into assembly code.
(2) Assemble the modules.
(3) Build the modules into a loadable program with base programs and system libraries.

Any undefined labels, multi-defined procedures, etc. discovered by stage 3, will necessitate the source being edited and the three stages being repeated. Some errors will not be identified by any of the stages, e.g. an 'EXT' brick specification differing from the 'ENT' definition of that brick.

This document describes a program which can be run between stage (1) and stage (2) and will report all errors including those not detected by the above three stages.

Note that the use of the Linkage Verifier is not mandatory. However, its use is highly recommended when a new program is being built for the first time or after substantial changes have been made to a functioning system.

3.2 CROSS REFERENCE INFORMATION

This information is output by the RTL/2 compiler to the data file specified by XREF in the compiler's command string.

It is arranged as a series of separate items, each starting a new line, and each being written as an assembler comment. Each type of item is indicated by a key letter, which is the first non-layout character to be found after the asterisk. An item of data looks thus:

* <key letter> <data item list>

Spaces embedded within the data of an item will be ignored.

3.2.1 Key letter interpretation

The Verifier interprets cross-reference items which have the key-letter N, R or X:

⟨name⟩, ⟨int⟩ and ⟨spec⟩ are defined in 3.2.2.

3.2.1.1 X - External references

Items with the key letter 'X' appear for every EXT or SVC declaration in an RTL/2 module, and for control routines and any other 'R' number labels used. There are five forms:

XP, ⟨name⟩ , ⟨spec⟩ which defines an EXT PROC brick
XS, ⟨name⟩ , ⟨spec⟩ which defines an EXT STACK brick
XU, ⟨name⟩ , ⟨spec⟩ which defines an SVC DATA brick
XV, ⟨name⟩ , ⟨spec⟩ which defines an SVC PROC brick
XY, ⟨name⟩ , ⟨spec⟩ which defines an EXT DATA brick
and XRnn which defines an external 'R' number

3.2.1.2 N - Entry points

Items with the key letter 'N' define the name and specification for every entry point in a module. There are five forms:

NP, ⟨name⟩ , ⟨spec⟩ for an ENT PROC brick
NS, ⟨name⟩ , ⟨spec⟩ for an ENT STACK brick
NU, ⟨name⟩ = ⟨int⟩ , ⟨spec⟩ for an SVC DATA brick
NV, ⟨name⟩ = ⟨int⟩ , ⟨spec⟩ for an SVC PROC
NY, ⟨name⟩ , ⟨spec⟩ for an ENT DATA brick

Note that the SVC DATA and PROC items define an entry integer value as well as a name and description; this is the value the name is to take in any module that refers to the SVC DATA or PROC thus defined. For an SVC DATA brick this integer is the address displacement (in bytes) of that data brick from the start of the SVC area. For an SVC PROC this integer is the data following the LKM instruction into which an SVC PROC call is compiled. The method used to include NU and NV items in

the cross-reference information is given below in section 3.2.5.

Note that R numbers are not defined by 'N' items. See 'R' below.

3.2.1.3 R - Control Routines

This item defines a control routine so that it may be referred to in this and other modules. The item may have five forms:

Rnn = <int>
Rnn = <name>
Rnn = <name> + <int>
Rnn = <Rmm>
Rnn = <Rmm> + <int>

where nn or mm are 2-digit decimal numbers.

The 'name' which is referred to must be a module name or entry point name.

'Rmm' which is referred to must be another 'R' number already declared in the same or a previously linked module.

The method used to include an 'R' declaration is a module is given below in Section 3.2.5.

3.2.2 Notation for data items

In the descriptions of items the following symbols are used:

<int> - this indicates a decimal integer.

<name> - this indicates the name of an RTL/2 brick or module. It should be noted that the Linkage Verifier, like the assembler, restricts names to a maximum of six characters. Longer RTL/2 names are, therefore, truncated.

< spec > - this is the brick specification string output by the compiler. It gives, in a coded form, a list of data types or parameters and results involved and is used by the Verifier to check that, for example, the description of an EXT data brick is correct. The symbols used in these strings are as follows:

B byte

I int

F frac

R real

P proc

L label

S stack

E ref

X ref array

A array. This will be followed by the bounds of the array (decimal) separated by a ',', if there are more than one.

M mode. This will be followed by a specification of the mode and terminated by an 'N', or, if this mode definition is already in the specification string, by a backward pointer 'Y'.

N end of mode.

Y backward mode pointer. This will be followed by an integer that indicates the position of the relevant mode definition in the string as a count of characters from the beginning of the string.

Q type or result of proc. Followed by the characters defining the result of the procedure, or by a 'Ø' if the procedure returns no result.

T repetition factor. This will be followed by an integer to indicate the number of repetitions. Not used in record specifications.

Z String terminator.

3.2.3 Examples

Two examples of cross-reference information are:

(1) ENT PROC OUTTP (BYTE X)
which would result in the following cross-reference data:
*N P,OUTTP,BQØZ

(2) ENT DATA DTX;
INT I,J;
REC HOLDER;
REF REC PTR;
REF ARRAY BYTE RAB;
ENDDATA;

which, if REC is defined by MODE REC (INT I1,I2,BYTE B3); would yield *N Y,DTX,IT2MIIBEY4XBZ

It can be seen that it is easily possible for these specification strings to be too long to fit on a single line. If this is the case they continue on as many continuation lines as necessary, each line starting with an asterisk.

3.2.4 Compiler Generated data

Cross-reference items are arranged by the compiler in two main groups. The first group describe the EXT, SVC or ENT declarations appearing in the module, i.e. the names and parameter or data layouts which the Verifier must check. The second group define the layout of the compiled brick structure of the module in sufficient detail to enable the program to be linked in other operating systems. The second group is however ignored by the Linkage Verifier.

In addition cross-reference information will contain details of non-RTL/2 external and entry references. These include:

(i) 'R' numbers, i.e. labels of the form Rnn (nn being two digit decimal numbers in the range ØØ to 99) which represent control routines or other system addresses and constants.

(ii) Entry values for SVC PROC and SVC DATA. The compiler will generate external reference calls for standard control routines. Other non-RTL/2 items are inserted by hand either via the compiler in CODE statements or in hand-written modules - see Section 3.2.5.

The order of cross-reference information is immaterial except that 'R' numbers used in defining others must be predefined.

3.2.5 Hand-coded Cross-Reference Information

A user may wish to include non-RTL/2 items of his own specification in the Cross-Reference Information. Such information is directly supplied to the linkage verifier.

3.2.5.1 Direct Hand Coding

Where a module has been written in assembler, the cross-reference data must be hand-coded according to rules in sections 3.2.1 and 3.2.2, and input directly to the linkage verifier.

3.3 VALIDATION ERRORS

Errors detected by the Linkage Verifier in the cross-reference information are reported to the user by a message of the form:

```
LNK ERROR  int
```

where the integer specifies the type of error. Some error messages will contain additional information as indicated below, which will follow the error number and be separated from it by a comma. On the following page there is a complete list of the error numbers and their meanings.

ERROR NO.	SIGNIFICANCE
1	An illegal item has been found on the cross-reference file being read.
2	The list used within the Verifier to hold 'ENT' brick information has overflowed. If this error occurs other side-effect errors may be generated as the Verifier resets its list pointer and overwrites the previously assembled information.
3	The list used within the Verifier to hold 'EXT' brick information has overflowed. If this error occurs, other side-effect errors may be generated as the Verifier resets its list pointer and overwrites the previously assembled information. The error is less likely to happen if the order of input of the cross-reference files is such that 'ENT' definitions precede their 'EXT' references.
4	Two 'ENT' bricks have been declared with the same name. The name is printed.
5	The specification of an 'EXT' brick does not conform to the 'ENT' definition of the brick. The name of the brick is printed.
6	An 'EXT' brick has been defined for which no corresponding 'ENT' exists. The name of the brick concerned is printed.
7	An 'R' number reference has been found that is not of the form 'R digit digit '. The two characters following the 'R' have been treated as digits and the value resulting from this treatment is printed.
8	An attempt has been made to define an 'R' number twice. The number concerned is printed.

ERROR NO.	SIGNIFICANCE
9	An 'R' number definition has been found which depends on another 'R' number which has not previously been defined in the cross-reference input. The number concerned is printed.
10	An external reference has been made to an 'R' number that is not defined. The number concerned is printed.

Notes:

(1) Error numbers 6 and 10 are only output during verification after all the cross-reference files have been input. Errors 1-4 and 7-9 are only output as the cross-reference information is read. Error number 5 may appear at either time.

(2) The limit for error 2 in the current version is 150 ENT bricks. The limit for error 3 is 300 unresolved EXT's at any one time.

3.4 RUNNING THE LINKAGE VERIFIER

3.4.1 Command Format

The catalogued procedure:

$LVE ⌈IN = file name⌉ ⌈,XF = file name⌉ ⌈,LS = file code⌉

is used to run the Linkage Verifier.
If IN is specified, the module names are read from the data file name supplied. Default is the system console.

LVE uses the line editor to build a source file of cross-reference data, starting with XF and adding the module names supplied. The default for XF is the standard file R:XREF containing all the system data. This file must be in the user's library.
LS is the file code to which the listing of source is sent. The default is 0 (Zero) for no listing.

When the procedure is run, the program asks for the names of the cross-reference data files for the user modules. Supply the names separated by commas or a new line. The list is terminated by replying with a new line. e.g.

```
MODULE NAMES?  X1, X2 (CR)
MODULE NAMES?  X3 (CR)
MODULE NAMES? (CR)
```

If the IN parameter is specified, the prompt is still output but replies are read from the data file specified.

Remember to add a blank line at the end of the file.

The procedure then runs the line editor. The editor will output warning messages on the console:

```
EOF IN AUXI INPUT
```

These are of no consequence.

3.4.1 Cont...

If a data file is not in the library, the editor will ask for the command to be re-entered on the console. Inspection of the line printer log will reveal which module name caused the error. If the cause is mis-typing a name in the first phase, the command may be re-entered as

ZZJN <name>,0,1000

After verification, a message

VERIFICATION OK/FAILS

is output on the console and any errors are reported on the line printer.

See section 2, Running the Compiler, for information on storing cross-reference data.

3.4.2 Example

A program is compiled as

$RTL FN=PROG, XREF=X:P

The compilation is successful so;

$LVE

MODULE NAMES? X:P
MODULE NAMES?

EOF IN AUXI INPUT

VERIFICATION OK (OR FAILS)

3.4.3 Environmental Errors

These are described in Appendix VI.

PHILIPS P800 DOM RTL/2 USER MANUAL

SECTION 4

THE RUN-TIME SUPPORT SOFTWARE

CONTENTS

4.1 INTRODUCTION

The RTL/2 package for DOM contains several run-time support items, some of which are mandatory. This section of the User Manual describes the modules at the user level. Further information concerning implementation may be found in Appendices. All run-time support is supplied in source format to enable the user to tailor any aspect to suit his particular requirements.

The mandatory items are the 'Control Routines' and the 'Base Program'. The control routines provide run-time support for the compiler-generated code and as such are bound intimately to the compiler's conventions. The base program provides stack initialisation, error handling and other miscellaneous features necessary to interface RTL/2 programs with the host environment.

The optional items are the components of the stream I/O support package. This obeys the RTL/2 standards defined in RTL/2 Reference 5. The character and formatted I/O routines defined there have been augmented by procedures which allow the user to open and close files and switch channels without having to be familiar with the rather complicated I/O interface. If the user wishes to utilise the full power of the operating system, which means interfacing at a lower level, procedures are available for making "link to Monitor" requests. To work at this level the user must be familiar with the operating system in detail.

Useful items of support documentation for this section are:

RTL/2 System Standards	RTL/2 Ref. 4 (5122 011 28961)
RTL/2 Standard Stream I/O	RTL/2 Ref. 5 (5122 011 28971)
The RTL/2 Environment on the P800	RTL/2 Ref. 69 (5122 991 28131

4.2 BASE PROGRAM

The base program, RRBPG,is written in assembler. There are conditional assembly instructions to generate long or short versions of the program.

4.2.1 Short version

This provides:

a) The SVC DATA bricks.

b) Data for the SVC procedures.

c) Code to get a dynamic buffer and initialise it for the run-time stack, initialise registers for RTL/2 and call the user's outermost procedure RRJOB.

d) The standard error procedure RRGEL (See RTL/2 Reference 4).

e) The standard procedures:

RRNUL - Null parameterless, resultless procedure.
RRIPF - Default input routine.
RROPF - Default output routine.

RRIPF and RROPF consist of a call to RRGEL with error numbers 98 and 99 respectively.

To assemble the short version set

PROGSZ \EQU \0

The stack length n in bytes is set by STLEN \EQU \ n

These assembler statements are just after the EXTRN statements in RRBPG.

4.2.2 Long version

The long version is as the short version but in addition RRGEL gives a register and stack dump on file code 2.

To assemble the long version set PROGSZ to any value except 0.

4.2.3 RRGEL

(see RTL/2 System Standards, RTL/2 Ref. 4) (5122 011 28961)

The DOM implementation performs the output of an error message and passes control to label ERL (if this is in scope). If ERL is not in scope, an RTL/2 error 3 is generated, the run-time stack is released and a program exit is executed (LKM 3).

The message is output on the line-printer and the operator console (file codes /2 and /01) and has the format:

RTL/2 ERROR ‹error number› LINE ‹line number›

Where ‹error number› is the parameter of RRGEL and ‹line number› is the number of the line in the source text of the last statement executed in a module which was compiled with the TR option.

When the long version has been specified, in addition to the above, register and memory dumps are output on filecode /2 in the following manner:

A1	A2	A3	A4	A5	A6	A7
8C60	0004	789C	0085	E8AC	3FD0	00B0
A8	A9	A10	A11	A12	A13	A14
885A	8F54	6C40	128E	D2E2	3FDC	3FD4

STACK DUMP

```
D2CO                                                    0000    D2D2
D2DO    789C    FAC1    8F54    FAC1    8956    0085    0002    0005
D2EO    88D4    D2E8    88C2    00EF    D2F2    7E82    05EF    FAC1
D2FO    00EF    D304    7A88    0001    79AB    79A7    2020    427E
D300    0000    0000    D308    789C    D30C    427E    D30C
```

CR AREA

```
3FCO                                                            0000
3FDO    0000    F9C1    78B4    0001    D2DO    FAC1    78F4
```

The dump format is the contents of eight memory words preceded by the address of the first word. In the above example the stack starts at D30C and the contents of this word is a pointer to itself. The stack is printed to the lowest point used so far in the program.

CRAREA is the Control Routine and fortran interphase stack pointed to by A14. It is only printed if it is in use.

For a description of the stack and register usage see "The RTL/2 Run-Time Environment on the Philips P800 Series" RTL/2 Ref. 69 (5122 991 28131)

4.3 STREAM I/O

Character stream I/O is supported according to the standards defined in RTL/2 Ref. 5, "RTL/2 Standard Stream I/O". The character and formatted I/O procedures defined therein are supported in the DOS implementation by procedures for opening, closing and switching I/O channels.

4.3.1 Standard Library Procedures

The following formatting procedures (defined in"RTL/2 Standard Stream I/O", RTL/2 Ref. 5) are available in source and object form:

FREAD, FWRT, FWRTF,
IREAD, IWRT, IWRTF,
RREAD, RWRT, RWRTF,
TREAD, TWRT, NLS, SPS

Errors from these procedures are detailed in RTL/2 Ref. 5.

The individual source Files supplied may be used to create an object library.

4.3.2 Stream I/O Support Procedures

These procedures permit the user to perform simple stream input and output without detailed knowledge of the DOM Library and the DOM monitor.

The standard I/O formatting procedures in the RTL/2 stream I/O library are supported as are the procedure variables IN and OUT in SVC DATA RRSIO.

4.3.3 Communication with Stream I/O Support Procedures

The parameter and result mechanism of RTL/2 is used exclusively for communication between the user and the support procedures.

4.3.4 Stream I/O Support Module

When building programs using I/O support, the user must also include the stream I/O support module, RRSIO. This may require regeneration to cover a greater number of streams in simultaneous use; the standard module as supplied is set for 8 streams.

4.3.5 The Detailed Specification

Appendix II contains detailed specifications of the stream I/O support module and the errors detected. An example program with the full sequence of input, compilation, assembly, linking and running commands is also included.

4.4 CONTROL ROUTINES

The control routines provide out-of-line functions in RTL/2 such as array bound checking and type conversions.

The P800 control routines will not in general operate in other environments.

4.4.1 Using control routines

Provided all the control routines are in the object library, the individual routines required by a program are picked out automatically by OLE when the program is linked.

For further information see "The RTL/2 Run-Time Environment on the Philips P800 Series", RTL/2 Ref. 69 (5122 991 28131)

4.4.2 Errors detected by the control routines

Errors detected by the control routines (which are normally unrecoverable) using RRGEL.

Error No.	Significance
1	Stack overflow (on procedure entry it is found that there will not be enough stack space to complete the procedure).
2	Label error (GOTO out-of-scope LABEL).
3	ERL out of scope.
4	Array bound error.
5	Fixed point overflow (when OPTION ()OV is in use.)
6	Floating point overflow.
7	Fixed point overflow on conversion.

4.5 USING MONITOR REQUESTS (LKMs)

4.5.1 Monitor Requests in RTL/2

SVC procedures are provided to enable the user to make Monitor Requests without having to drop into code. With the exception of the procedures for getting and releasing dynamic buffers, a knowledge of the operating system is necessary to set up the communication blocks required.

4.5.2 Procedures for getting and releasing dynamic buffers

There are two procedures in the module RRBUFF.

Their use does not require any detailed knowledge of the operating system but the following should be noted.

1. The procedure does not set an array length word and so the procedures can only be used in the systems language and care must be taken to avoid breaking array bounds.

2. When a buffer is released the gap in memory is not closed up and a future buffer request will only succeed if there is an available gap large enough to accept it.

If there is not enough room for a request an unrecoverable error 65 occurs.

The procedures are defined by

```
EXT PROC (INT) REF ARRAY BYTE RRGBF;      and
EXT PROC (REF ARRAY BYTE) RRRBF;
```

for getting and releasing buffers respectively. The parameter to RRGBF is the length of buffer required in bytes (remember to add 2 bytes for the array-length word).

Note that if the user wants a mode structure in dynamic memory, the procedures may be defined in the user's module with REF MODE instead of REF ARRAY BYTE. The error this will cause in Linkage Verification may be ignored.

4.5.3 Example of use

```
PROC USEBUFF;
   REF ARRAY BYTE P:= RRGBF (102);
   P(60):=0;  % CAN USE P(1) TO P(100)%
...etc...
   RRRBF(P);
ENDPROC;
```

Note that using P(101) or P(102) will destroy the buffer area chaining links.

4.5.4 Other Monitor Requests

These are invoked by SVC PROCs. When the compiler encounters an SVC Procedure, in-line code is generated which places the first parameter in A7, the second parameter in A8 and then does an LKM with data equal to the value of a symbol defined in the Base Program. If only one parameter is specified it is put in A8. If the procedure is defined as returning an Interger result, the contents of A7 after the LKM are returned. There is a list of procedures and their LKM numbers in Appendix I.

The procedures are defined in the user's program as for example:

```
SVC PROC (INT, REF INT) RRTIME;
```

The above procedure will generate an LKM 17 (under DRTM) and the procedure is defined in the Base Program by the assembler statement;

```
RRTIME  EQU  17
```

4.5.5 Monitor Requests not included

The SVC procedure table in the Base Program covers all LKM requests at time of writing. Extra procedures to cover user written LKMs can be included simply by including the name and the LKM number in the Base Program in the form

```
<name>      EQU    <LKM number>
```

LKM requests with scheduled labels are not included in the basic package because their use will normally corrupt the run-time stack. However if a user wishes to use scheduled labels after taking the necessary precautions, he can define the negative LKM number as above.

4.5.6 Example

To set the time under DRTM.

```
SVC PROC (INT, REF INT) RRTIME;
MODE TIME (INT HOUR, MINUTE, SECOND, TENTHS, FIFTYTHS, CTIME);

DATA TIMEDATA;
  TIME T;
ENDDATA;

PROC GTIME();

  RRTIME(1,T.HOUR); %COMPONENTS OF T NOW CONTAIN TIME%
  ...etc...

ENDPROC;
```

PHILIPS P800 DOM RTL/2 USER MANUAL

SECTION 5

USER PROGRAM GENERATION

CONTENTS PAGE

5.1 INTRODUCTION

This section describes in detail the process of compilation, assembly and linking of a program with the run-time support software described in section 4 in order to produce a runnable load module.

5.1.1 Other Target Operating Systems

Generation of operational programs for operating systems other than P800 DOM is not dealt with here. Users wishing to run their programs under other operating systems should consult RTL/2 Ref. No. 69 "The RTL/2 Run-Time Environment on the Philips P800 Series" for guidance on implementing their own support software. Note also that the DOM control routines, base program etc., are not interchangeable with other available implementations of RTL/2 (e.g. IBM 370), and may need modification for other systems, e.g. MAS though they may be used as a basis for user-written support software.

5.2 COMPILATION AND ASSEMBLY

The RTL/2 compiler is described fully in section 2. Output from the compiler will normally be assembled with the DOM assembler, although other compatible compilation/assembly environments may be used. The catalogued procedure $RTL automatically assembles the output if the compilation is successful.

5.3 LINKING

5.3.1 Non overlayed programs

Provided all the run-time software (Base program, Stream I/O, etc is in either the system or the user's object library, the Philips Linkage Editor will pick out the required modules and link them with the user's program.

Note that the start address in the base program, RRSTR, should be used as a parameter in the OLE command.

5.3.2 Example

A user has a module JOB1 in his object library which calls procedures contained in JOB2 and JOB3, also in his object library. The run-time software in the system library. The following commands are used to link the modules.

```
INC JOB1
OLE M, RRSTR
```

The overlay linkage editor will pick out JOB2 and JOB3 from the user library and the relevant control routines etc from the system library.

For a full description of the commands and parameters consult the relevant Philips Programmer's Reference Dates for your machine.

5.3.3 Overlayed programs

The overlay structure is defined using INC and NOD commands as in the Philips documentation. It should be remembered that the stream I/O control module, RRSIO, although it uses dynamic buffers, has various flags set within the module itself and so, if different paths are using stream I/O, RRSIO should be included at the root of the paths.

The stream formating library and the control routines can be picked out as required by the overlay linkage editor.

5.3.4 Stack

The size of the run-time stack is defined in the base program, RRBPG, by

```
STLEN  EQU  n
```

where n is the size in bytes. The stack size can be changed by modifying n and re-assembling the base program.

There is an SVC DATA brick

```
SVC DATA RRSTK
INT STKLO,
    STKLIM,
    WSPLO;
ENDDATA;
```

which contains, the lowest point so far encountered in the program, the lowest point in the stack and the lowest point in use at any particular moment. Note that addresses greater than 32K bytes will be stored as negative numbers.

RRSTK is updated by control routine R:ROl.

PHILIPS P800 DOM RTL/2 USER MANUAL

APPENDIX I

SVC PROCS

CONTENTS

I.1 SVC PROCS

```
RRIOLKM (INT, REF BYTE);
    LKM  1
I/O on a peripheral device.
Note:  most I/O operations, with the notable exceptions
       of the Extended Data Management Package, are
       supported by the stream I/O, described in Appendix II.
```

```
RRWAIT (REF BYTE);
    LKM  2
Wait for an event.
```

```
RREXIT (INT, INT);
    LKM  3
Exit from a program.
```

```
RRGBF (INT) REF ARRAY BYTE;
    LKM  4
RRRBF (REF ARRAY BYTE);
    LKM  5
Both described in 4.5.
```

```
RRPAUSE (INT, REF BYTE);
    LKM 6
Pause.
```

```
RRCNABT (REF INT,LABEL);
    LKM  7
Keep control on abort conditions.
Note:  RTL/2 labels consist of 3 words.
```

```
RRLDSEG (INT, REF INT) INT;
    LKM  9
Load and segment.
Note:  Not needed when OLE provided.
```

```
RRCNTIM (REF INT, REF INT) INT;
    LKM  10     (DRTM only)
Connect a program to a timer.
```

```
RRDCTIM (INT, REF INT) INT;
    LKM  11     (DRTM only)
Disconnect a program from a timer.
```

```
RRACTV (REF INT, REF INT) INT;
    LKM  12     (DRTM only)
Activate a program.
```

```
RRSWITCH (INT, INT);
    LKM  13     (DRTM only)
Switch inside a software level.
Note:  second parameter not used.
```

```
RRATDEV (INT, REF INT) INT;
    LKM  14     (DRTM only)
Attach a device to a program.
```

```
RRDTDEV (REF INT) INT;
    LKM  15     (DRTM only)
Detach a device from a program.
```

```
RRTIME (INT, REF INT);
    LKM  17     (DRTM only)
Get time.
```

```
RREVENT (REF BYTE);
    LKM  18     (DRTM only)
Set an event.
```

```
RRCNLEV (INT, REF INT) INT;
    LKM  20     (DRTM only)
Connect a program to a software level.
```

```
RRDCLEV (INT, REF INT) INT;
    LKM  21      (DRTM only)
Disconnect a program from a level.
```

```
RRWTIM (REF INT) INT;
    LKM  22      (DRTM only)
Wait for a given time.
```

```
RRASG (REF INT) INT;
    LKM  23      (DRTM only)
Assign a file code.
```

```
RRDEL (REF INT) INT;
    LKM  24      (DRTM only)
Delete a file code.
```

```
RROPMSG (REF INT);
    LKM  25      (DRTM only)
Read unsolicited operator message.
```

```
RRCANREQ (REF INT);
    LKM  26      (DRTM only)
Cancel request for an unsolicited operator message.
```

I.2 USE OF SVC PROCEDURES

The use of SVC procedures is explained in Section 4.5.

PHILIPS P800 DOM RTL/2 USER MANUAL

APPENDIX II

STREAM I/O SUPPORT SPECIFICATION

CONTENTS PAGE

CONTENTS CONTINUED PAGE

II.1 PROCEDURE SPECIFICATIONS

II.1.1 Basic Procedures for opening and closing streams

There are four basic procedures, defined in the user's module by:

```
EXT PROC (INT) PROC () BYTE RROPI;
EXT PROC (INT) PROC (BYTE)  RROPO;
EXT PROC (PROC () BYTE) RRCLSI;
EXT PROC (PROC (BYTE) ) RRCLSO;
RROPF and RROPO → RROPI and RROPO
```

RROPF and RROPO open input and output streams, RRCLSI and RRCLSO close input and output streams.

The parameter to RROPI and RROPO is the file code of the device or temporary file. For a list of the filecodes for the devices consult the relevant Philips "Programmer's Reference Data" for your machine.

The filecode supplied may be assigned to any device before run-time but if it is not assigned at all a run-time error will occur.

RROPI and RROPO return a procedure variable which is used for input or output. This procedure variable must be supplied to RRCLSI and RRCLSO to close the streams.

Although there are cases where it is not necessary to close a stream before exiting from a program it is strongly recommended to always close a stream when it is finished with. If a disc file contains strange data after a run it is probably because the stream has not been closed.

II.1.2 Example

A program to output a message on the system console and line printer.

```
     IDENT OUTMES
TITLE STREAM I/O EXAMPLE;
SVC DATA RRSIO;
    PROC ()BYTE IN;
    PROC (BYTE) OUT;
ENDDATA;
EXT PROC (INT) PROC (BYTE) RROPO;
EXT PROC (PROC (BYTE)) RRCLSO;
EXT PROC (REF ARRAY BYTE) TWRT;
LET NL = 10;

ENT PROC RRJOB();
    PROC (BYTE)  SCOUT: = RROPO (HEX EF),
                 LPOUT: = RROPO (2);
    OUT:= SCOUT;
    TWRT ("#NL#SYSTEM CONSOLE#NL#");
    OUT:= LPOUT;
    TWRT ("#NL#LINE PRINTER #NL#");
    RRCLSO (SCOUT);
    RRCLSO (LPOUT);
ENDPROC;
```

II.1.3 Effect of opening and closing a stream

When a stream is opened the following takes place:

1. Buffer space is obtained from the dynamic buffer area.

2. Input Disc files are rewound, except logical files /E0 and /EE

II.1.3 Cont...

When an output stream is opened,
No line feeds or page throws are issued, so if the program is writing to the line printer for example, a line feed should be issued before writing to avoid overprinting the program header.

When an input stream is closed

1. The dynamic buffer is released.

When an output stream is closed

1. The last buffer is output.

2. An :EOF mark is written.

3. Disc files are rewound, except logical files /EO and /EE.

4. The dynamic buffer is released.

II.2 EXTRA FACILITIES OF THE STREAM I/O PACKAGE

II.2.1 Single Buffering

The standard stream opening procedure for output streams generates a double buffering mechanism. There may be cases when a user would sooner have some extra memory rather than the extra speed of double buffering. For these cases an extra procedure, RROPOS, is provided which is identical to RROPO, except that it generates a single buffering mechanism. A stream opened by RROPOS is closed in the normal way by RRCLSO.

II.2.2 Supplying an I/O order

It is possible to supply an I/O order as part of the parameter when using RROPI or RROPO. Before doing this, the user should be familiar with the effects of different orders, as described in Philips "Programmer's Guide", Appendix C Peripheral Input/Output.

The parameter to RROPI or RROPO is an integer of 16 bits whereas the file code is a byte of 8 bits. The first 8 bits of the parameter may be used for an I/O order. This means that the 'S' bit cannot be set. The effects of the different orders on the stream I/O are explained fully later in this section. There are some subtle differences to the use of the orders in an ssembler environment.

II.2.3 Default I/O orders

The default I/O order for RROPI is HEX82, standard read with wait bit set.

The default order for RROPO is HEX 05, standard write with wait bit not set.

Thus RROPO(2) is equivalent to RROPO(HEX 0602)

II.2.4 Supplying I/O orders - Control bits

S bit

Cannot be set.

W bit (lst bit of parameter)

For output channels will cause a wait whenever a buffer is output. If the bit is not set the output procedure will take care of synchronisation. If the bit is set there is little point in using the double buffering procedure.

For input channels the wait bit should normally be set. If the bit is not set then the system will read in a fresh buffer as soon as the last character in the old buffer has been passed to the input stream.
To make the best use of this facility, the TREAD procedure should be used with NL as a terminating character to read a whole buffer in at once. The next buffer will be read in from the device while the buffer in memory is being processed.

R bit (2nd bit of parameter)

It is not possible to process abnormal conditions in the user program, however if the status word LAND HEX ID3≠0 the input procedure will return the end of stream character HEX80 and the user may wish to use the R bit in conjunction with this.

II.2.5 Supplying I/O orders - functions

Functions occupy bits 3 to 8 of the parameter.

Basic Read (1)

Characters will be read until the buffer is full. No special significance attached to control characters. May be used for reading binary data.

Basic Write (5)

The default order. The buffer is output whenever the buffer is full or a control character is encountered. The character NL = 10 will cause a new line,carriage return whatever the device is. The character VT = 11 may be used to print a buffer without moving the printhead, for example when an answer is required on the same line on the console.

Basic Write (4)

This is a dummy order and will cause the control characters to be output as any other. The buffer is output when it is full. It is used for outputting binary data. It should not be used on the printers as it will cause random form feeds and line feeds.

Standard Read (2)

The default order. For a full description of its effects see Philips "Programmer's Guide".

Standard Write (6)

See "Programmer's Guide".

Can be used, for example, on printers if a control code at the start of a buffer is required rather than a control character. This use should be avoided as it does not conform to RTL/2 Standards.
Can also be used to punch with the special sequence LF-XOFF - CR - Rubout at the end of a record. Basic write only outputs LF - CR.

The character VT = 11 should be used to signify end of record with this order.

Object Write (7 and 8)

See "Programmer's Guide".

It should not be necessary to use these orders.

Orders greater than 8

These should not be used when opening a stream. Their use is described in the next section.

II.3 FILE CONTROL PROCEDURES

II.3.1 Procedures available

There are two file control procedures available, defined in the user's module by:

```
EXT PROC (INT) INT RRORDI, RRORDO;
```

RRORDI is used for the file connected with the current input stream and RRORDO for the current output stream. The parameter is the order specifying which function is required. The result is zero except where order HEX 30 is specified when the result is the ASCII value of the characters representing the device. Note that this order is simulated to avoid destroying the control block.

A check is made that the function is compatible with the stream, for example, 'Write EOF mark' is not allowed on the input stream. No attempt is made to make sure that the order is only issued to relevant devices, for example, 'Rewind to load point' may be issued to a line-printer even though it may block the system. Remember to output the current buffer, if necessary, by outputting a control character before using these procedures.

II.3.2 Orders available

HEX14 Skip forward to EOS mark
HEX16 Skip forward to EOF mark
HEX22 Write EOF mark
HEX24 Write EOV mark
HEX26 Write EOS mark
} OUT stream only
HEX30 Return device
HEX31 Rewind to load point
HEX33 Backspace one block
HEX34 Space one block forward (not allowed for cassette)
HEX36 Skip backward to EOF mark
HEX38 Unlock

II.3.2 Cont

For further information see Philips "Programmer's Guide".

Note the functions carried out by the Stream opening and closing procedures in Section 1 of this appendix.

II.4 END OF STREAM

II.4.1 EOS Character

On input, the end of stream is signified by the RTL/2 character EOS = HEX 80 being returned, and IOFLAG being set to 2. End of stream is caused by :EOS, :EOF and any condition which leaves STATUS LAND HEX ID3#0.

On output, the EOS character has no significance and is output as any other. It is not treated as a control character.

II.4.2 End of stream on input

It is possible to read beyond the end of stream. If this is not desired, the user should look for EOS as in

```
WHILE CHAR#HEX80 DO etc.
```

or he should check IOFLAG for 2 at the end of every line as in

```
WHILE IOFLAG = 0 DO
    TREAD    etc.
```

IOFLAG is reset to 0 when a new buffer is successfully read. For binary streams the user should check for IOFLAG=2 after each character. Note that if data is read from the console either by an RTL/2 program or a system program, the end of stream is signified by typing :EOF on a new line.

II.4.3 End of stream on output

Normally end of stream is signified simply by closing the stream. The user may close the stream himself using the control procedure to write an :EOS or :EOF, if, for example, he does not

II.4.3 Cont...

want the file automatically rewound. The file will not be properly closed by simply outputting ":EOF". A stream I/O procedure must be used.

II.5 CONFIGURATION OF STREAM I/O PACKAGE

II.5.1 Maximum buffer size

The buffer size for a file is set to the value returned by DOM. The I/O module sets a maximum value for this buffer and disc file buffers are always set to this maximum value.

To make the most effective use of the dynamic area as many buffers as possible should be of the same size and it is recommended to set the maximum at or less than the maximum for a device. A good value is 136, the length of the line printer and some console buffers. 256 may be chosen if cassettes are used. The stream I/O package will print long buffers on two lines if necessary.

The maximum size n in bytes is set by

```
    LET MAXB = n;
in  RRSIO
```

The revised version should be macro-processed, compiled and assembled.

II.5.2 Number of Streams

The maximum number of streams is normally 8. These streams may be divided among input and output streams in any way.

If it is desired to have more than eight streams open at one time, RRSIO may be reconfigured. Change the macro statements:

```
[SET N 8 ]    and
[SET P 16]
```

to the number of streams and twice the number of streams respectively.

II.5.2 Cont...

The revised version should be macro-processed, compiled and assembled.

The number of streams may be reduced in the same way to save space. Each stream has an overhead of 120 bytes.

Note that the macro-processor may run out of stack space if the number of streams is large, because of the recursive nature of the macro-structure in RRSIO. See Section 4.

II.6 ERRORS

II.6.1 Error Reporting

Errors detected by the stream I/O procedures are unrecoverable and cause a call to RRGEL.

Error No.	Procedure	Significance
51	RROPI RROPO RROPOS	No streams left.
53	RROPI RROPO RROPOS	File code unassigned.
54	RROPI RROPO RROPOS	Illegal device for input order
55	RROPI RROPO RROPOS	Illegal device for output order
59	IN,RRORDI	Stream not open.
50	OUT,RRORDO	Stream not open.
61	RRCLSI RRCLSO	Stream already closed
63	RRCLSI	Trying to close an unknown stream
64	RRCLSO	Trying to close an unknown stream.
65	RROPI RROPO RROPOS	No dynamic memory available.
71	RRORDI	Illegal order for an input file.
72	RRORDO	Illegal order for an output file.

II.6.2 Error Correction

Error 51 is corrected by closing a stream already open or by rebuilding RRSIO for more streams. See II.4.

Error 53 is corrected by assigning the file code before running the program.

Errors 59, 6o are caused by using a stream after it has been closed. Using a stream before it has been opened, in effect, assigning IN or OUT to an uninitialised procedure variable, will leave the program in limbo.

Errors 61,63,64 should be self-evident.

Errors 71,72 are caused by illegal orders or orders such as 'write EOF' on an input stream.

Error 65 caused by RRGBF not finding a spare memory slot. Can be corrected by closing stream.

PHILIPS P800 DOM RTL/2 USER MANUAL

APPENDIX III

GENERATING THE RTL/2 UTILITIES

CONTENTS

III.1 INTRODUCTION

The RTL/2 Compiler and Linkage Verifier are supplied in object form for building in the User's environment. This Appendix covers the structure of these programs.

III.2 RTL/2 COMPILER

III.2.1 Overlay Structure

The compiler is built by using the Overlay Description Language facilities provided by OLE.

The RTL/2 compiler can only be linked using OLE.

The overlay tree structure has 21 segments including the root segment.

Each overlay segment is loaded once (if at all) during a compilation. None of the back-end overlay segments will be loaded when a compilation fails through front-end diagnostics.

III.2.2 Command file

The procedure $TREE will build the overlay structure and run OLE. All the compiler modules should be in the same userid, although fortran interface modules may be in another object library specified by a parameter U. This speeds up the access time for the object modules. Before building, the user should make sure that RRCIPD contains the required channel associations (See III.4).

III.3 LINKAGE VERIFIER

This is built as a single overlay structure to rebuild do:

```
INC   LV1
OLE   <parameters as desired>, RRSTR
KPF   /L, LVER
```

The linkage editor will pick out LV2 and the required run-time routines. Before linking, the user should make sure the RRCIPD contains the required channel associations (see III.4).

III.4 AUTOMATIC STREAM SELECTION BY THE CHANNEL INTERFACE PACKAGE

III.4.1 The utility programs use module RRCIP as an Interface to RRSIO. When the utilities set up an I/O stream they pass a question to RRCIP which prints it on the console and asks for the file code desired for this stream as in:

LISTING OF SOURCE TO?

However the file code may be stored permanently in module RRCIPD. In this case the question is not output and the file code is selected automatically. RRCIPD contains two arrays, INCODES and OUTCODES.

If an element is negative, the association is prompted at the console. If the element is 0, this corresponds to typing (CR) to a question. i.e. a null association. If the element is positive it is treated as the actual file code.
The channels are standard to all utilities

INCODES		OUTCODES	
1	Conversational	1	Conversational
2	Source input	2	Main output
3	Not used	3	Report and error
		4	Listing
		5	Not used

The user could, for example, change OUTCODES (3) to output to a VDU.

Note that if INCODES (1) = HEX EE, the catalogued procedure file, then OUTCODES(4) is set to an integer read in from file HEX EE. This means that the listing channel can be specified by a parameter in a catalogued procedure.

III.5 STANDARD RRCIPD VALUES

For the Compiler;

```
INCODES:= (HEX EE,HEX D4,0);
OUTCODES:=(HEX 01,HEX BF,2,0,HEX BD);
```

For the Linkage Verifier;

```
INCODES:= (HEX EE,HEX BF,0);
OUTCODES:= (HEX 01,0,2,0,0);
```

PHILIPS P800 DOM RTL/2 USER MANUAL

APPENDIX IV

SVC DATA BRICKS

CONTENTS

IV.1 INTRODUCTION

SVC DATA bricks are applicable only to multitasking operating systems, where each task has a private copy and procedures shared by different tasks are able to automatically access the appropriate area.

Strictly SVC DATA is not necessary within the single program environment of DOM. However, the inclusion of the standard SVC bricks greatly improves the ease of writing portable programs, and may be of particular interest to users developing (and perhaps testing) programs under DOM for eventual running in other environments.

The SVC DATA bricks have permanent space allocated in RRBPG, which initialises register A13 to the address of the area and defines the offsets for each brick name.

IV.2 STANDARD RTL/2 SVC DATA BRICKS

The following bricks are as defined in RTL/2 Systems Standards, and are given here for completeness only.

IV.2.1 Stream I/O

```
SVC DATA RRSIO;
    PROC()BYTE IN;
    PROC (BYTE) OUT;
ENDDATA;
SVC DATA RRSED;
    BYTE TERMCH,
         IOFLAG;
ENDDATA;
```

IV.2.2 Error Recovery

```
SVC DATA RRERR;
    LABEL ERL;
    INT ERN;
    PROC (INT) ERP;
ENDDATA;
```

IV.2.3 Default Settings of Standard SVC DATA

On entry to RRJOB the base program will have set up the following values:

```
IN:= RRIPF;
OUT:= RROPF;
TERMCH:= HEX 80; (end of stream)
IOFLAG:= 0;
ERL points to start of RRGEL;
ERN:= 0;
ERP:= RRGEL;
```

IV.3 DOM EXTENSION OF SVC DATA

The following additional bricks are included in the DOM implementation.

```
SVC DATA RRSTK;
  INT  STKLO,
       STKLIM,
       WSPLO;
ENDDATA;
```

This block should not be modified by the user but WSPLO contains the lowest point of the stack reached in the program and is of interest in assessing stack requirements.

```
SVC DATA RRERRX;
  INT  LINENO;
ENDDATA;
```

This block is used by the control routines when the TR option is used, to record the last traced line number.

IV.4 USER EXTENSION OF SVC DATA

Users implementing under other operating systems and wishing to include their own extensions of SVC DATA for program testing under DOM can modify RRBPG to include the necessary extra space, and definitions of further offsets.

PHILIPS P800 DOM RTL/2 USER MANUAL

APPENDIX V

MATHEMATICAL ROUTINES

CONTENTS

V.1 ROUTINES SUPPLIED

The routines supplied are:-

```
EXT PROC (REAL) REAL RSQRT;
```
returns positive root of a positive real number.

```
EXT PROC (FRAC) FRAC FSQRT;
```
returns positive root of a fraction.

```
EXT PROC (REAL) REAL RLOGE;
```
returns the log base e of a positive real number.

```
EXT PROC (REAL) REAL RLOG10;
```
returns the log base 10 of a positive real number.

```
EXT PROC (REAL) REAL REXP;
```
returns e to the power of a real number.

```
EXT PROC (REAL) REAL RSIN, RCOS;
```
returns sine or cosine of a real parameter in radians.

```
EXT PROC (REAL) REAL RATN;
```
returns the are tangent in radians of a real number.

V.2 ACCURACY

RSQRT

With NOIT=3 the error is less than 10^{-8} and is generally around 10^{-9}.

The error is given by

```
ABS  ((RSQRT(R) * RSQRT(R) - R) / (2.0 * R))
```

FSQRT

With NOIT=4 the error is less than 6×10^{-5}.

The error is given by

```
ABS ((FSQRT(F) * FSQRT(F) - F) / (2.0 * F))
```

RLOGE,RLOG10

The error is less than 2×10^{-9}.

The error is given by

```
ABS (RLOGE (R * R) - 2 * RLOGE(R)) / (4.0 * RLOGE(R));
```

REXP

The error is less than 8×10^{-9}.

The error is given by

```
ABS (REXP(R) - REXP (R * 0.5) * REXP (R*0.5)) / (3.0 * REXP(R))
```

RSIN,RCOS

The error is of the order of 10^{-9}.

The error is given by

```
ABS (RSIN(R) * RSIN(R) + RCOS(R) * RCOS(R) - 1) / 4.0
```

RATN

The error is of the order of 10^{-9}.

The error is given by

```
ABS ((RATN (2.0 * R / (1 - R*R)) - 2.0 * RATN(R)) / (4.0 * RATN(R)))
```

V.3 NOTES ON MATHEMATICAL TECHNIQUES

RSQRT

The number is reduced to the range $[\frac{1}{2},1)$ using sequential blocks.

A linear approximation is made and the result is iterated a a fixed number of times using the Newton-Raphson formula;

$$Un+1 = \frac{1}{2}(Un + R/Un)$$

The exponent is dealt with separatly.

FSQRT

A value N is found such that

$$C.2^{-2n} \leq F \leq C.2^{-2n+2} \quad \text{for } N = 0,1,2 \ldots.$$

A linear approximation is made in this range by

$$X = F.2^{N-1} + C.2^{-N}$$

and a fixed number of Newton-Raphson iterations are performed.

Note that divisions by 2 is performed by SRA 1.

RLOGE,RLOG10

The formula used is

$$\ln x = \log\sqrt{2} + 2\alpha + \frac{2}{3}\alpha^3 + \frac{2}{5}\alpha^5 + \frac{2}{7}\alpha^7 + \ldots.$$

$$\alpha = \frac{X - 1/\sqrt{2}}{X + 1/\sqrt{2}}, \quad |\alpha| < |.$$

The procedure uses sequential blocks to reduce R to $[\frac{1}{2},1)$ and the exponent is calculated as E*ln2. The coefficients are modified to reduce the error. RLOG10 simply multiples the result at RLOGE by $LOG_{10}e$.

REXP

EXPTOP is chosen so that overflow will not occur.

$e^x = 2^{\log_2 e^x} = 2^{x \log_2 e}$

So R is transformed to $R\log_2 e$.

This is decomposed to N + Y , $|Y| \leq |$

So $e^R = 2^n \cdot 2^Y$

Y is transformed to $Y/2 * \log_e 2$

and Cody & Ralston's Alogorithm is used to calculate 2^Y.

2^n is calculated by a simple loop.

RSIN,RCOS

INTEG returns the parameter module 2π
SIGTOP is chosen so that overflow will not occur.
SCALESIN returns $\sin \frac{R\pi}{2}$ where $R \in [-2.0, +3.0]$
It first adjusts R to be E [-1.0,1.0] using the periodic nature of the sine function. The sine is found using the Chebychev expansions.

RCOS(R) simply calls RSIN $(R + \pi/2)$

RATN

The range is reduced to (0,1)
and then to $(0, 2-\sqrt{3})$

In this area,convergence of the series, modified to improve accuracy,is rapid.

V.4 ERRORS

Errors are recoverable.

Error No.	Procedure	Cause
300	RSQRT,FSQRT	Negative Parameter
301	RLOGE,RLOG10	Negative Parameter
302	RSIN,RCOS	Number too large
303	REXP	Number too large

PHILIPS P800 DOM RTL/2 USER MANUAL

APPENDIX VI

ENVIRONMENTAL ERRORS DETECTED

BY UTILITIES

CONTENTS

VI.1 GENERAL

In addition to error diagnostics produced by the utilities, certain environmental errors may be detected by their supporting software. These are described in this appendix.

TABLE VI.1 UNRECOVERABLE ERRORS

Error No.	Meaning
9950	Should not occur
9955	
9956	
9965	
9966	
9980	Random Access disc file not assigned before Run-Time
9993	Should not occur
9994	
9995	
9996	

Errors that should not occur may occur as a result of recovery following an earlier error. Their occurance should only be reported if they are the first error in a run.

In addition, stream I/O errors may occur. Errors are usually caused by files not being assigned properly before run-time.

VI.2 FATAL ERRORS

Any of the errors listed in Table VI.1 cause the utility to abort, giving an RRGEL message.

PHILIPS P800 DOM RTL/2 USER MANUAL

APPENDIX VIII

SUMMARY OF ERROR NUMBERS

CONTENTS

VII.1 INTRODUCTION

This appendix is provided as a quick reference to aid users in locating the sections in the manual containing detailed error number description.

There are two categories or errors, those reported as RTL/2 Errors by calling RRGEL, and those reported by some other mechanism specific to the program involved.

VII.2 RRGEL REPORTED ERRORS

Error No.	Source	Reference	Applicability
1,2,4,5, 6,7	Control Routines	4.4.2	All RTL/2 programs
3	Base Program	4.2.3	All RTL/2 programs
98,99	RRIPF, RROPF	4.2.1	All RTL/2 programs
100-103	Stream I/O Formatting Procedures	RTL/2 Systems Standards	All RTL/2 programs
51-71 65	Stream I/O RRGBF	II 4.5.2	Programs using stream I/O
9950 to 9996	Environment for RTL/2 Utilities	VI	The RTL/2 compiler, or linkage verifier

WARNING The user is reminded to adhere to the recommendations regarding error numbers in"RTL/2 Systems Standards, RTL/2 Ref. 4"

VII.3 UTILITY REPORTED ERRORS

Error Identifier	Source	Reference
F1 thru F2O4	RTL/2 Compiler Front End	2.1.4
B1 thru B2O3	RTL/2 Compiler Back End	2.2.5
LINK ERROR 1 thru LINK ERROR 1O	RTL/2 Linkage Verifier	3.3