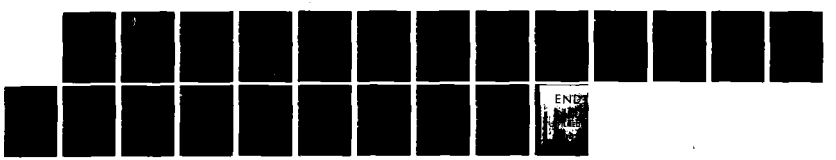
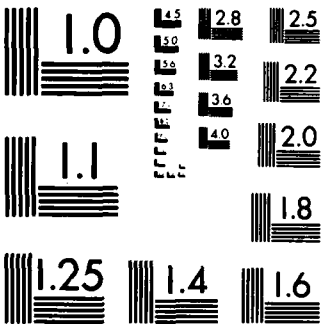
AD-A134 368 A PORTABLE INPUT/OUTPUT PACKAGE FOR CORAL BASED 8080 1/1
SYSTEMS(U) ROYAL SIGNALS AND RADAR ESTABLISHMENT
MALVERN (ENGLAND) M P GRIFFITHS MAY 83 RSRE-MEMO-3589
UNCLASSIFIED DRIC-BR-88546 F/G 9/2 NL





MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

RSRE
MEMORANDUM No. 3589

ROYAL SIGNALS & RADAR ESTABLISHMENT

A PORTABLE INPUT/OUTPUT PACKAGE FOR CORAL BASED 3080 SYSTEMS

Author: M P Griffiths

PROCUREMENT EXECUTIVE,
MINISTRY OF DEFENCE,
RSRE MALVERN,
WORCS.

ROYAL SIGNALS AND RADAR ESTABLISHMENT

MEMO 3589

Title: A PORTABLE INPUT/OUTPUT PACKAGE FOR CORAL BASED 8080 SYSTEMS

Author: M P Griffiths

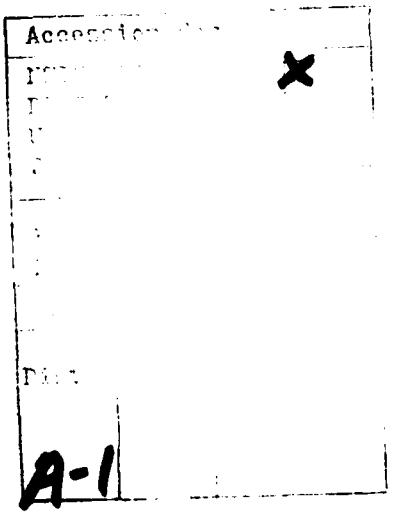
Date: May 1983

SUMMARY

This memo describes a set of procedures for use with the RCC80 Coral compiler, which enable user programs to access the host operating system and transform integer and floating point numbers to and from ASCII strings and their binary representations. The structure of the library is described, and this shows how the procedures may be easily transported onto a new hardware configuration.

CONTENTS

APPENDIX



1 INTRODUCTION

The Official Definition of Coral 66[1] does not specify any input or output procedures for the language and so it has been left to implementors to invent their own facilities for each installation. This memo describes a library of procedures to operate with the RCC80 compiler (originally supplied by GEC but now supported by MicroFocus) and provides facilities to interface the user's program to the machine's operating system to enable him to open and close files, read and write characters to them and read and print integer and floating point numbers.

The RCC80 compiler is hosted on the Intel MDS, and compiles Coral source to Intel 8080/8085 assembly language. Consequently, the MDS is the most common target for user programs and the major use of this library is to communicate with the MDS's operating system ISIS-II. However, in order to increase the number of applications for which this library may be used, the procedures are written in such a way as to make the transportation of them onto other 8080 based systems as easy as possible.

Since different hardware and software configurations have different facilities, all programs will not always work on all systems, (for example, attempting to use a disc file depends on having a file handling system present) but it was felt to be important that INOUT should always present exactly the same interface to the user program. This means that in order to change systems, it is only necessary to relink the compiled program with the appropriate relocatable library file, and if the configuration cannot do what the program requests then an error will be flagged at run time.

All the following procedure descriptions, therefore, are written without assuming a specific target system (although the examples given assume an MDS with ISIS-II as a target), and you should refer to Appendix A to find out what facilities will operate with your chosen version. At the time of writing four versions of INOUT have been implemented. A section is also provided to give some guidelines for those wishing to write their own version.

Most procedures are of type integer and return a value indicating the success or failure of the operation. A value of zero indicating success, and any other value the reason for the failure. As these values may be dependent on the operating system in use, you should again refer to Appendix A for the possible error numbers and their interpretation.

All input/output files are assumed to be byte sequential, referred to as streams throughout this memo. That is, a file is a string of characters which must be read or written from start to finish.

2 STARTING AND STOPPING

There are two untyped procedures for initialising INOUT and exiting when finished, these are USESTREAMS and STOP.

2.1 USESTREAMS

This procedure must be called before any attempt is made to use the other procedures described below.

```
procedure USESTREAMS;
```

USESTREAMS has no parameters and does not return a result. Its function is to set up the defaults for the rest of the package, and initialise any required buffers. It is a good policy to always put it as the first statement in any program.

2.2 STOP

This is used to terminate processing and exit to the operating system (if any).

```
procedure STOP;
```

STOP has no parameters, and may be called from any point in the program. It closes all open streams in their current state, and then exits to the operating system. If the configuration does not have a command level then the processor will halt.

3 FILE HANDLING

Procedures are provided to open files, read or write data to them and close them when finished.

3.1 OPENINSTREAM

This opens a file for input (read).

```
integer procedure  OPENINSTREAM (location integer FILE;
                                 value integer STRING)
```

where FILE is an integer reference in which the stream number is returned and STRING is a string containing the name of the file.

For example:

```
integer STATUS, READER;
STATUS:=OPENINSTREAM(READER,":HR:");
if STATUS<>0 then ERROR(STATUS);
```

opens the paper tape reader. Note that in this example (for ISIS-II) that devices are referred to by a similar convention to file names, except that a special format is used with colons around the name.

3.2 OPENOUTSTREAM

This opens a file for output (write).

```
integer procedure OPENOUTSTREAM(location integer FILE;
                                value integer STRING)
```

where FILE is an integer reference in which the stream number is returned and STRING is a string containing the name of the file.

For example:

```
STATUS:=OPENOUTSTREAM(PRINTER,":LP:");
```

opens the line printer.

3.3 OPENADDSTREAM

This opens a disc file for append, i.e. addition of bytes to the end of a previously created file.

```
integer procedure OPENADDSTREAM(location integer FILE;
                                value integer STRING);
```

where FILE is an integer reference in which the stream number is returned and STRING is a string containing the name of the file.

For example:

```
STATUS:=OPENADDSTREAM(TABLE, "TABLE")
```

will open the file :F0:TABLE for append.

3.4 GETSTREAM

This reads the next character (byte) from the specified file.

```
integer procedure GETSTREAM(value integer FILE;
                            location byte DATA);
```

where FILE is an integer returned from a previous OPENINSTREAM call, and DATA is a byte reference to put the character in.

Continuing the OPENINSTREAM example:

```
STATUS:=GETSTREAM(READER, CHAR);
```

gets the next character from the paper tape reader and puts it in CHAR.

3.5 PUTSTREAM

This writes a character to a given file.

```
integer procedure  PUTSTREAM(value integer FILE;
                             value byte DATA)
```

where FILE is an integer returned from a previous OPENOUTSTREAM or OPENADDSTREAM call, and DATA is the character to be put.

Continuing the OPENADDSTREAM example:

```
STATUS:=PUTSTREAM(TABLE, literal(A));
```

puts the ASCII code for A on to the end of the disc file TABLE.

3.6 MESSAGE

This writes a character string to a given file.

```
integer procedure MESSAGE(value integer FILE, STRING)
```

where FILE is an integer returned by a previous OPENOUTSTREAM or OPENADDSTREAM call, and STRING is a string containing the characters to be written.

Continuing the OPENOUTSTREAM example:

```
STATUS:=MESSAGE(PRINTER, "NOW IS THE HOUR");
```

which writes the message NOW IS THE HOUR on the line printer.

3.7 RESETSTREAM

This restarts a disc file which was opened for read from the beginning, so that it can be re-read.

```
integer procedure RESETSTREAM(value integer FILE)
```

where FILE is an integer returned from a previous OPENINSTREAM call.

3.8 CLOSESTREAM

This closes a stream and releases the number for further use in another OPEN call.

```
integer procedure CLOSESTREAM(value integer FILE)
```

where FILE is an integer returned by a previous OPENINSTREAM, OPENOUTSTREAM or OPENADDSTREAM call.

Continuing the GETSTREAM example:

```
STATUS:=CLOSESTREAM (READER);
```

closes the paper tape reader.

4 INTEGER INPUT AND OUTPUT

Four procedures are provided for integer input and output: INTIN and INTOUT, which convert ASCII strings to and from binary integers, and READI and PRINTI which use INTIN and INTOUT to transfer the results to or from a file.

4.1 INTIN

This converts an ASCII string to a binary integer.

```
integer procedure INTIN (location integer  VALUE;
                         value integer PTR;
                         location integer  DIST)
```

where VALUE is an integer reference for the return of the value input, PTR is the address of the character of which to start looking for the number and DIST is the number of bytes the procedure searched before it reached the end of the number. Thus PTR + DIST will be the address of the character that terminated the number.

The rules that the character string must obey before being classed as a number are:

(a) A number starts with a plus sign, minus sign or a digit 0 to 9.

(b) A number is terminated by a character other than a digit 0 to 9, a letter A to F, a letter H, a letter O, or a letter Q.

(c) The base is assumed to be 10 unless the last character is B, when the base is 2; O or Q, when the base is 8; or H, when the base is 16. A last character of D may be used to explicitly specify base 10.

(d) The value input must be greater than -32767 and less than +32767.

Thus valid numbers are:

```
23, -4, 24Q, 0FH, 00011B
```

but not:

```
40000, 8000H,  BFH, 385Q
```

For example:

```
STATUS:=INTIN(VALUE, "0FFH,", DIST);
```

will set VALUE to 255 and DIST to 5.

4.2 READI

This uses INTIN to read a number from a file.

```
integer procedure READI(value integer FILE;
                        location integer VALUE);
```

where FILE is an integer returned from a previous OPENINSTREAM call and VALUE is an integer reference to put the number in.

For example:

```
STATUS:=READI(READER, A);
```

will read the next number and put it in A.

4.3 INTOUT

This converts a binary integer to an ASCII string.

```
integer procedure INTOUT(value integer VALUE, PTR);
```

where VALUE is the number to be converted and PTR is the address where the first character of the string will be put. Succeeding characters will be placed in PTR + 1, PTR + 2 and so on. The format of the output string is controlled by 6 parameters, each being set by its own procedure of the form:

```
integer procedure FORMAT VARIABLE (value byte SETTING)
```

where FORMAT VARIABLE is selected from the following table and SETTING is any allowed value.

(1) WIDTH

This specifies the number of characters in the output field. The characters will be stored in PTR to PTR + WIDTH -1, and the procedure always delivers WIDTH characters. WIDTH must be between 1 and 18, if the number is too big to go into the available space, then WIDTH asterisks are output and a status of 103 returned.

Default: 7

(2) BASE

This specifies the base in which the number is printed. BASE must be 2, 8, 10, or 16.

Default: 10

(3) PLUSSIGN

This specifies the character to be used for printing a plus sign. It is normally literal(+) but may be any printing character or -1 meaning suppress the plus sign altogether.

Default: -1

(4) LEADZERO

This specifies the character to be used for leading zeroes. It is normally literal(0) but may be any printing character or -1 meaning suppress leading zeroes altogether.

Default: -1

(5) SIGNED

If this is set to 1, then the number will be interpreted as signed two's complement. If 0, then it will be interpreted as positive straight binary with no sign output.

Default: 1

(6) POSTBASED

If this is set to 1, then the last character of the number will be B, Q, D, or H depending on the setting of BASE. If 0, then the letter will be suppressed.

Default: 1

To set any of the above it is necessary to call the corr[illegible]ding procedure. For example, to set WIDTH to 12:

```
STATUS:=WIDTH(12);
```

Any attempt to set a forbidden value will result in a STATUS of 104 and the call will be ignored.

For example:

```
CODE:=5;
WIDTH(8);
BASE(2);
LEADZERO(literal(0));
SIGNED(0);
POSTBASED(0);
STATUS:=INTOUT(CODE,location(BUFF[0]);
```

will fill BUFF[0] to BUFF[7] with the ASCII string 00000101.

4.4 PRINTI

This uses INTOUT to write the string to a file.

```
integer procedure PRINTI(value integer FILE, VALUE)
```

where FILE is an integer returned from a previous OPENOUTSTREAM or OPENADDSTREAM call and VALUE is the number to be printed.

Continuing the above example:

```
STATUS:=PRINTI(PRINTER, CODE)
```

will print 00000101 on the printer.

5 FLOATING INPUT AND OUTPUT

Four procedures are provided for floating point input and output: FLTIN, READF, FLTOUT and PRINTF which work is a similar manner to their integer counterparts.

Note that throughout this section the @ sign means "times 10 to the power".

5.1 FLTIN

This converts an ASCII string to a binary floating.

```
integer procedure FLTIN(location floating VALUE;
                        value integer PTR;
                        location integer DIST)
```

where VALUE is a floating reference for the return of the value input, PTR is the address of the character of which to start looking for the number and DIST is the number of bytes the procedure searched before it reached the end of the number. Thus PTR + DIST will be the address of the character that terminated the number.

The corresponding rules for floating character strings are:

(a) A number must start with a plus sign, minus sign, or digit 0 to 9.

(b) A number is terminated by a character other than a digit 0 to 9, a plus sign, a minus sign, an @ sign or a decimal point.

(c) The base is always 10.

(d) The value input must be greater than -1.7@38 and less than +1.7@38.

Thus valid numbers are:

```
123, -4.5, -3.65@-4, 6@3
```

but not:

```
@6, .4, 1@40
```

5.2 READF

This uses FLTIN to read a number from a file.

```
integer procedure READF(value integer FILE;
                        location floating VALUE)
```

where FILE is an integer turned from a previous OPENINSTREAM call and VALUE is a floating reference to put the number in.

5.3 FLTOUT

This converts a binary floating to an ASCII string.

```
integer procedure FLTOUT(value floating VALUE;
                         value integer PTR)
```

where VALUE is the number to be converted and PTR is the address where the first character of the string will be put. Succeeding characters will be placed in PTR + 1, PTR + 2 and so on.

The format of the output string is set by 4 parameters in a similar manner as in INTOUT. These are:

(1) WIDTH

This specifies the number of characters in the output field. The characters will be stored in PTR to PTR + WIDTH -1, and the procedure always delivers WIDTH characters. WIDTH must be between 1 and 18, if the number is too big to go into the available space, then WIDTH asterisks are output and a status of 103 returned.

Default: 7

(2) PLUSSIGN

This specifies the character to be used for printing a plus sign. It is normally literal(+) but may be any printing character or -1 meaning suppress the plus sign altogether.

Default: -1

(3) DECIPLACE

Specifies the number of digits to be printed after the decimal point. It must be between 1 and 7. As the representation of number is accurate to 7 significant figures, it is possible that some digits will not be printed. In this case the last printed digit is rounded up if the unprinted part starts with 5 or more, e.g. if DECIPLACE = 3 and VALUE = 1.3455, this would come out as 1.346.

Default: 7

(4) AUTOFLOAT

If this is set to 1 then the decimal point is floated, if it is set to 0 then scientific notation is used i.e. the number is printed in the form x.xxx@xx . However, even if AUTOFLOAT is set to 1 there are two situations in which FLTOUT will revert to scientific. These are (a) if the number is greater than 9999999, and (b) if DECIPLACE is set such that the number to be printed would be truncated to zero. For example, in the situation DECIPLACE = 2 and VALUE = 0.0004 then if scientific notation were not used, this would be printed as zero with a consequent loss of accuracy. Note that whichever notation is used you still get DECIPLACE digits after the decimal point.

Default: 1

For example:

```
NUM:=1.34567;
WIDTH(10);
DECIPLACE(4);
STATUS:=FLTOUT(NUM, location(BUFF[0]);
```

will fill BUFF[0] to BUFF[9] with the ASCII string:

sss1.34567 where s means space.

5.4 PRINTF

This uses FLTOUT to write the string to a file.

```
integer procedure PRINTF(value integer FILE;
                         value floating VALUE)
```

where FILE is an integer returned from a previous OPENOUTSTREAM or OPENADDSTREAM call and VALUE is the number to be printed.

6 USING THE LIBRARY

As mentioned in the introduction, there is more than one version of INOUT depending on which system the program is to be run. At the time of writing there are four versions of INOUT, the differences between these versions are described in detail in Appendix A.

6.1 Source Linkage

In order to enable a program to use the package, it must have the necessary external linkages to the INOUT procedures. This is achieved by placing a library call to the file INOUT.EXT at the top of any segment which uses the procedures:

```
coral PROG

library "INOUT.EXT"

begin

        etc.
```

This file is the same for all versions of INOUT, therefore it is not necessary to recompile a program to run with another version provided, of course, that both configurations can provide the required facilities.

6.2 Object Code Linkage

When the program has been compiled it must be linked with the appropriate relocatable library file (and SYSTEM.LIB if on the MDS) before being run. Thus the stages to go through to compile and link a single segment program would be:

```
CORAL PROG.CRL$S
ASM80 PROG.MC8 NOPRINT
LINK PROG.OBJ,INOUT#.LIB,CORAL.LIB,SYSTEM.LIB TO PROG.LNK
LOCATE PROG.LNK
```

Where the INOUT#.LIB refers to one of the INOUT libraries as selected from those available as documented in Appendix A. Note that SYSTEM.LIB is only required if you are using INOUT.LIB or INOUTF.LIB as these are the versions which have the MDS as a target.

Due to the way the compiler works, if INOUT.EXT is inserted as it stands, you always get all the procedures included in the final program, not just the ones you actually use. So, to reduce the amount of code and link/locate time, you can edit INOUT.EXT to remove the external references to procedures you do not use. Removing all the floating procedures, for example, for just integer working.

7 IMPLEMENTATION OF NEW VERSIONS

This section is provided to give some guidance to those users who wish to transport the package onto a new system. I will start off with a brief description of the structure of the library, indicating what needs to be done in order to write a new version. This is followed by some hints on what the altered versions of the procedures should contain so as to maintain the maximum of compatibility with existing versions.

7.1 Structure of the Package

Each version of INOUT is composed of a number of files, these are:-

(1) The Coral source of USESTREAMS
(2) The Coral source of the three OPENSTREAM procedures which are implemented as a single procedure with an additional parameter
(3) The Coral source of CLOSESTREAM
(4) The Coral source of GETSTREAM
(5) The Coral source of PUTSTREAM
(6) The Coral source of RESETSTREAM
(7) The Coral source of MESSAGE
(8) The Coral source of STOP
(9) An assembly language program to declare the external variables as required by the compiler

(10) The Coral source of 4 utility character checking procedures (not directly accessable as part of INOUT)
(11) The Coral source of the procedure that handles the setting of the format variables
(12) The Coral source of INTIN
(13) The Coral source of READI
(14) The Coral source of INTOUT
(15) The Coral source of PRINTI
(16) The Coral source of FLTIN
(17) The Coral source of READF
(18) The Coral source of FLTIN
(19) The Coral source of PRINTF
(20) The file to be inserted at the top of every program that contains the external declarations of the procedures (INOUT.EXT)

The last 11 of these files (numbers 10 to 20) are exactly the same regardless of the version used. Consequently, the implementation of a new version of INOUT consists of writing new versions of the first 9 files to interface either with an existing operating system or directly with the hardware of the target machine. Each INOUT#.LIB is composed of the compiled code of all of the above files (except (20), of course) put together by the Intel LIB program. Due to the fact that the ISIS assembler will only accept symbols of 6 characters or less, the names of procedures longer that this limit are abbreviated to 6 characters by the use of a macro in INOUT.EXT (e.g. USESTREAMS is abbreviated to QUSEST).

7.2 Hints on Implementation

In order to maintain the ability to change versions by merely relinking the object code of the user's program with a different library file, it is necessary to provide dummy routines in certain cases so as to tie off the loose ends of a facility which is not implementable in the new configuration. To take a simple example, RESETSTREAM is not usable without a disc filing

7.3 Final Remarks

Although this memo has assumed that the RCC80 compiler hosted on the MDS will be used, there is no reason why other Coral compilers on different hosts should not be able to use most of the principles and code contained in this package. In order to do this, the host compiler must be able to compile programs containing the byte type, be able to generate code to address bytes and the target must use the ASCII character set.

8 REFERENCES

[1] "Official Definition of Coral 66",
HMSO, 1970

[2] H S Field-Richards,
"The DISCUS Hardware System",
RSRE Report 82010

[3] Intel Corp,
"ISIS-II User's Guide",
Intel 9800306

[4] H S Field-Richards, M P Griffiths,
"DISCUS Multiprocessor Enhancements",
RSRE Memo 3591

APPENDIX A

Currently Implemented Versions

A.1 INOUT.LIB

This is the standard library for use with the MDS and ISIS-II. Consequently, the following restrictions apply:

(1) Device and file names correspond to the normal ISIS conventions, that is a file name is a string of up to six characters with an optional three character extension. Device names are enclosed in colons, e.g. :HR: for the paper tape reader.

(2) STOP closes all open files except the current console and returns to the ISIS command level.

(3) Stream numbers 0 and 1 are always open and refer to the current console output and input respectively

(4) GETSTREAM and the other procedures that use it (READI, READF) properly return a status number of 101 at the end of file.

(5) MESSAGE is more efficient than a repeated call of PUTSTREAM.

Note that in this version all transfers to and from the peripherals are unbuffered. This has the advantage that the status of the transfer is checked at every procedure call, as ISIS is invoked every time. However, it does mean that programs with lots of disc access run very slowly. Use this version for debugging and the faster version (INOUTF.LIB) for running if you suffer from this problem.

A.2 INOUTF.LIB

This version also runs on the MDS with ISIS-II and provides the same facilities as INOUT.LIB, but is provided with large buffers for access to all peripherals (except the console). Thus programs with large amounts of input and output run much faster (factors of 7 are typical), but the price paid for this is the reduced frequency of error checking and the enormous memory overhead (about 24k). However, for most programs of normal size this is not too much of a problem.

Note that MESSAGE is not more efficient than PUTSTREAM in this version.

Caution! Due to the internal buffers that INOUTF uses, it is extremely dangerous to use the ISIS system calls OPEN, CLOSE, READ, WRITE, and SEEK in a program using this version.

A.3 INOUTD.LIB

This version runs on a DISCUS 8080 style processor[2] together with the 8080 peripheral card with software version 2.1 or greater. Since this configuration does not have discs or any resident system software the following restrictions apply:

(1) A call of STOP halts the processor.

(2) Calls of OPENADDSTREAM and RESETSTREAM and OPENIN/OUTSTREAM with non-device type file names produce non-zero error returns and the call is ignored.

(3) Available devices names are:- :CO:, :CI:, :HR:, :HP: and :LP:. As with ISIS, :CO: and :CI: are given the stream numbers of 0 and 1, and are always open.

(4) GETSTREAM only gives a proper end of file indication for :HR:.

(5) MESSAGE is more efficient than repeated calls of PUTSTREAM.

A.4 INOUTZ.LIB

This version runs on a DISCUS 8085 style processor[4] together with the Z80 peripheral card with software version 2.0 or greater and an intelligent bus supervisor card. Again, this configuration does not have discs or any resident system software so the following restrictions apply:

(1) A call of STOP halts the processor.

(2) Calls of OPENADDSTREAM and RESETSTREAM and OPENIN/OUTSTREAM with non-device type file names produce non-zero error returns and the call is ignored.

(3) Available device names are:- :CO:, :CI:, :HR:, :HP:, :FP: and :LP:. As with ISIS, :CO: and :CI: are given the stream numbers of 0 and 1, and are always open. The device :FP: refers to the LED front panel display which enables a message of up to 16 characters to be displayed.

(4) GETSTREAM only gives a proper end of file indication for :HR:.

(5) MESSAGE is more efficient than repeated calls of PUTSTREAM.

A.5 Error Numbers

As stated previously, these error numbers are consistent throughout all the versions of the package. Numbers less than 100 are derived from the equivalent ISIS error number[3]. New numbers should be assigned in ascending order from the last number in this list.

100 Character too big for the indicated base, e.g. the 8 in 438Q.
101 Attempt to read off the end of a file.
102 Overflow, the number is too big.
103 WIDTH is not wide enough to accommodate the number.
104 Attempt to set an invalid value for a format variable.
105 Plus or minus sign with no following characters detected.

DOCUMENT CONTROL SHEET

Overall security classification of sheet ..... UNCLASSIFIED .....

(As far as possible this sheet should contain only unclassified information. If it is necessary to enter classified information, the box concerned must be marked to indicate the classification eg (R) (C) or (S))

1. DRIC Reference (if known)	2. Originator's Reference	3. Agency Reference	4. Report Security Classification
	MEMORANDUM 3589		UNCLASS-IFIED

5. Originator's Code (if known)	6. Originator (Corporate Author) Name and Location
	ROYAL SIGNALS AND RADAR ESTABLISHMENT
5a. Sponsoring Agency's Code (if known)	6a. Sponsoring Agency (Contract Authority) Name and Location

7. Title

A PORTABLE INPUT/OUTPUT PACKAGE FOR CORAL BASED 8080 SYSTEMS

7a. Title in Foreign Language (in the case of translations)

7b. Presented at (for conference papers) Title, place and date of conference

8. Author 1 Surname, initials	9(a) Author 2	9(b) Authors 3,4...	10. Date	pp. ref.
GRIFFITHS, M.P.				

11. Contract Number	12. Period	13. Project	14. Other Reference

15. Distribution statement

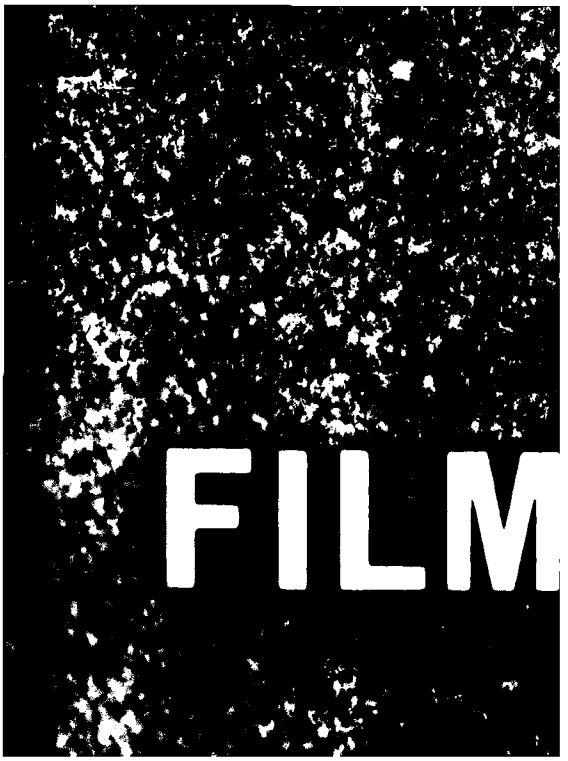
UNLIMITED

Descriptors (or keywords)

continue on separate piece of paper

Abstract

This memo describes a set of procedures for use with the RCC 80 Coral compiler, which enable user programs to access the host operating system and transform integer and floating point numbers to and from ASCII strings and their binary representations. The structure of the library is described, and this shows how the procedures may be easily transported onto a new hardware configuration.


FILM