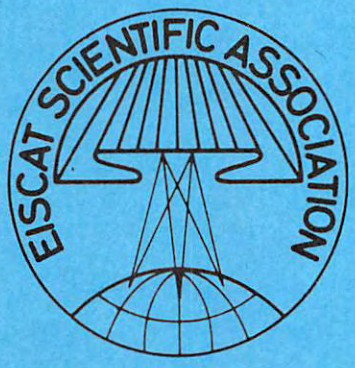
82/36



EISCAT
TECHNICAL
NOTE

CORLAN
(CORrelator LANguage)

by
B W Törustad

KIRUNA
Sweden

C O R L A N

(CORrelator LANguage)

by

Bård Willy Törustad

EISCAT Scientific Association
S-981 27 Kiruna, Sweden
October 1982
EISCAT Technical Note 82/36
Printed in Sweden
ISSN 0349-2710

CORLAN

(CORrelator LANguage)

An attempt to simplify the programming of the EISCAT digital correlator.

By

BRD WLLY TRSTD

Tromso JULY 1982

ABSTRACT

Present development of programs for the EISCAT correlator is being done by manipulating bits and numbers in a dedicated 'editor'. This in spite of an earlier attempt to induce the bitmagicians to use an assembler developed for the correlator.
This attempt probably failed because the only simplification offered was giving the bits and numbers mnemonic names restricted to a length of 4 characters. This made the writing of programs tedious and difficult still.

The main problems with the present system (altho good enough for some people) are:

1) Already developed programs are difficult (impossible) to read and understand.
2) Thus difficult for others to learn from existing programs to write their own.
3) Hence only 2 or 3 persons as a whole (and none within EISCAT) can be said to have skill in writing such programs.
4) The correlator has, because of this, been left as a black magic box (it has put the spell on us) nobody 'understands'.

It is the hope that this report and CORLAN will improve this situation and make a more varied and effective use of the correlator possible.

TABLE OF CONTENTS

0. INTRODUCTION

This report describes a programming language for the correlator: CORLAN. The purpose of this language is to ease the development of correlator-programs as to complexity and make the programming process more efficient, less timeconsuming and easier to learn.
The language consists sometimes of long words (like PREPARETRANSFER). This because it makes the source code easy to read and thus the program itself becomes a good documentation of what is happening inside it. There is also a comment facility available in addition.
Some checking is done by the translator, but the responsibility as to the logical correctness of the program rests mainly with the programmer.
The programmer has the same possibilities to control the correlator in CORLAN as with the traditional way of programming. This is necessary because the program-memory of the correlator is only 63 instructions long.
The correlator instruction word is divided into fields which applies to different parts of the correlator. Only those fields which are of interest at any time need be programmed. The other fields are set to sensible defaults.
The generated object code is compatible with already existing software (EROS and CORRSIM).
As an introduction the report gives a very simple review of the correlator hardware.

The translator is coded in PS-PASCAL (from CERN) and was written as an exercise in that language.

1. GENERAL

Notations used in the report:

::= means 'defined as'
/ means 'or'
<> anything enclosed in these brackets will be defined more specifically later
.. Signifies an inclusive range

For example:

<number> ::= <digit> / <number><digit>
<digit> ::= 1/2/3/4/5/6/7/8/9/0

That is a digit is 1 or 2 or .. or 0 and a number is one or more digits.

1..100 means all numbers from 1 to 100 inclusive.

Anything not appearing inside <>, is to be taken as it is written.

'<statement> ::=' is to be understood as 'all statements used to program the feature pertaining to the present chapter are as follows'.

A CORLAN-program must be created and written onto a disk-file by means of the NORD program editor (QED) before being submitted to the CORLAN-translator.

The only separator in CORLAN is the SPACE. (Apart from two cases where the COMMA is used). End-of-line is treated as a SPACE and has no significance other than that of a SPACE and as END OF COMMENT LINE. This means that there is virtually no restrictions as to how the source code can be written onto a file.

2. A SHORT AND INCOMPLETE REVIEW OF THE CORRELATOR HARDWARE

2.1 The correlator's place in the radar

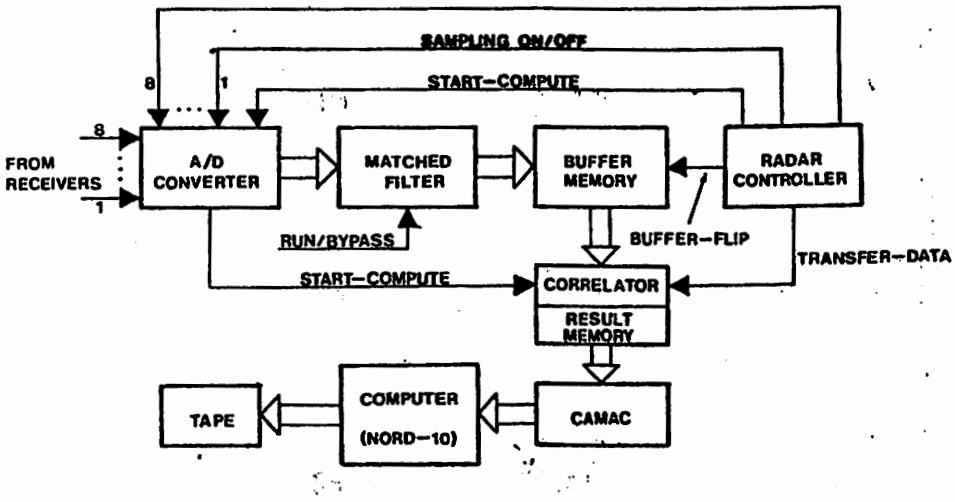


Figure 2.1 the correlator's place in the radar

Comments to figure:

1) About the start compute: by setting the status word in the correlator appropriately it is possible to start it from the computer. By means of the statusword one can choose to read data from an internal buffer or from the external (normal) buffer memory.

2) The start compute signal will not reach the correlator if there has been an overflow in the ADC. The overflow will be reset by the next start compute from the radar controller.

3) The correlator reads from 1 half of the buffer memory while the ADC writes to the other half. At the end of a scan the buffer memory is flipped, that is the correlator starts reading from the half that the ADC just wrote to etc.

4) The TRANSFER-DATA signal forces the correlator to enter a program which dumps the result memory into the computer. This process has to be programmed by the user who decides what and how much to transfer to the computer. The dump rate is approximately 0.75 MHz.

5) Each half of the buffer memory is at present 4k*2*8 bit words

6) The result memory is at present 2k*2*32 bit words.

2.2 Data and instruction flow in the correlator

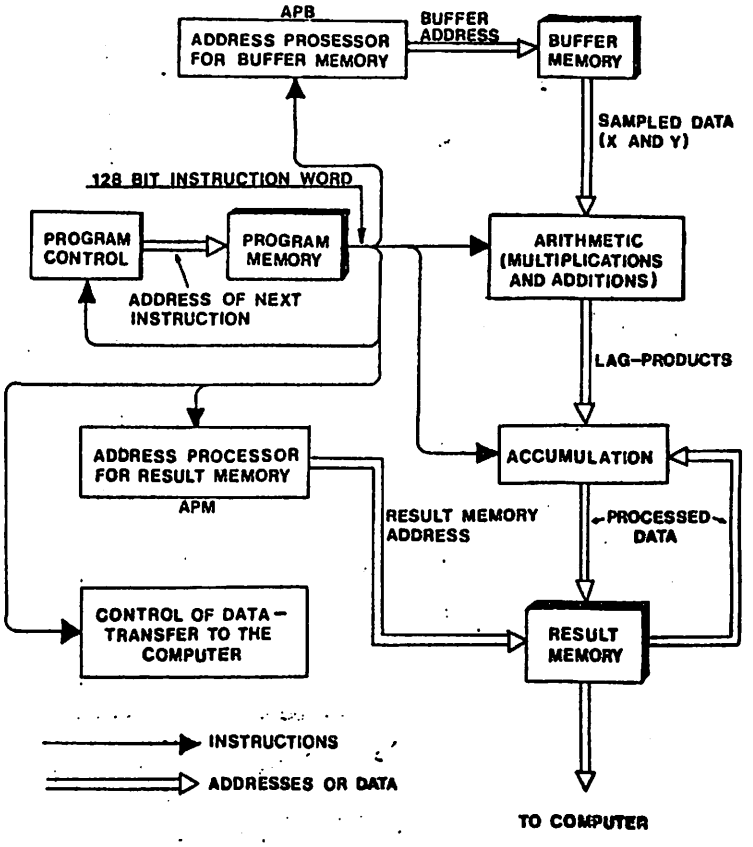


Figure 2.2 data and instruction flow in the correlator

See figure 2.2. 1 correlator-instruction is 128 bits wide. It consists of different instruction fields going to different parts of the correlator as indicated by the simple lines in figure 2.2.
The manipulations of 1 datasample from the buffermemory to the resultmemory is programmed in one instruction. It is however, clear that the samples have to be read before they can be treated by the arithmetic and the accumulator. The instructions for the arithmetic and accumulators have to be delayed in relation to the ones for the APB-processor. This is done automatically by the correlator and the user need not worry about it. (There is however one special case. For that refer to sections 4.4 and 4.5)
Further it is clear that this increases the thruput of the data because Xvergold and Yvergold can be in the accumulator whereas Xold and Yold are in the arithmetic while Xnew and Ynew are being read from the buffer memory. The technique of having all this happening at the same time is called 'pipelining'.
The program memory of the correlator is only 64 words long.

2.3 Description of each major part of the correlator

2.3.1 Address processor for the buffermemory

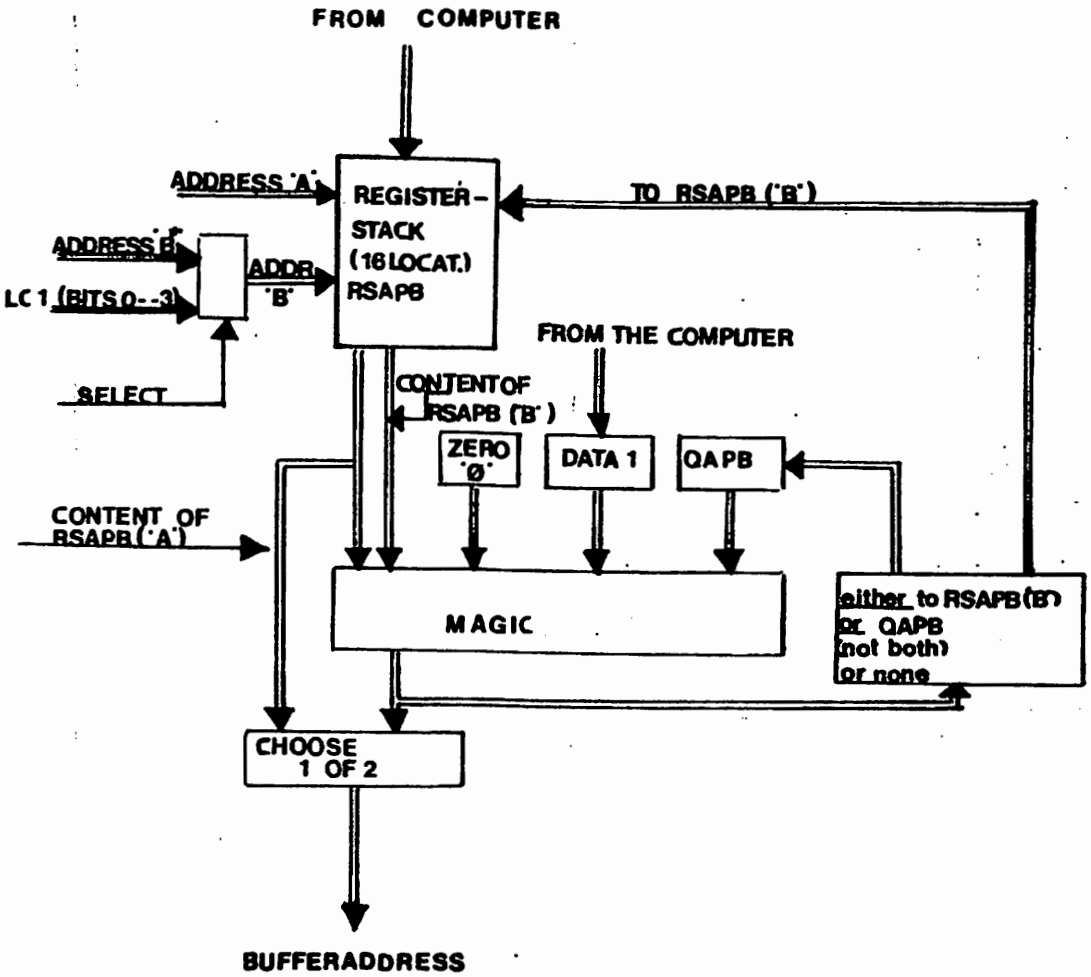


Figure 2.3 address processor for the buffermemory

Output of MAGIC is the sum or difference of 2 of the inputs (the 2 beins 2 different ones) or simply 1 of the inputs. The '0' must always occur 'alone'.

Examples:

```
BUFFERADDRESS=0
BUFFERADDRESS=-DATA1
QAPB=-DATA1
RSAPB(B)=RSAPB(B)+RSAPB(A)
BUFFERADDRESS=RSAPB(LC1)
```

A- and B-addresses refer to fields in the instruction-word. In the case of B-address on figure 2.3, it is possible to select it as bits0..3 of LC1 OR B-address from the instruction word (called B* on the figure).

As can be seen from figure 2.3, only RSAPB(B) can be written to and read from. RSAPB(A) can only be read from. On the bit-level one has to remember this when programming, whereas CORLAN will allocate the 'A' and 'B' addresses automatically. On the bit-level a single bit has to be set if LC1 (loopcounter1) is wanted as 'B' address. CORLAN will automatically generate this bit when RSAPB(LC1) is programmed. Note that LC1 as address only

refer to 'B' and select is valid with all references to RSAPB(B). That means it is not possible to refer to RSAPB(LC1) and RSAPB(B) in the same instruction. These things are checked by CORLAN.

2.3.2 Address processor for result memory

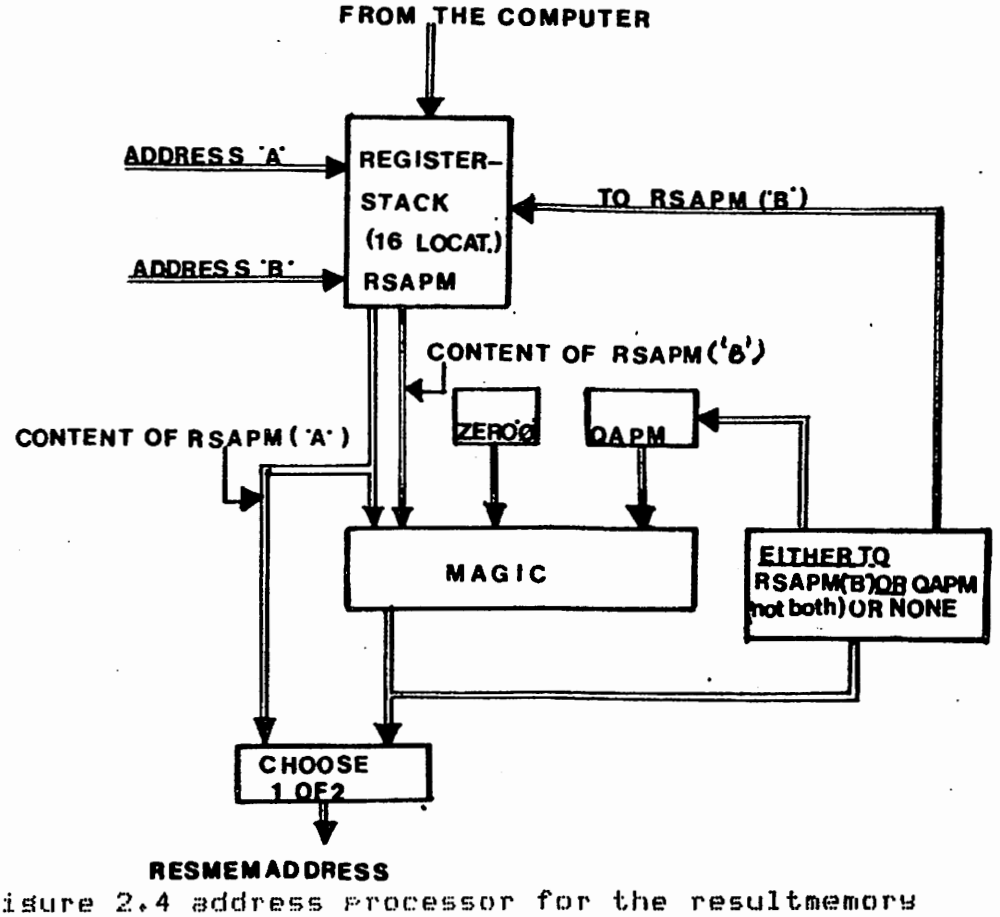


Figure 2.4 address processor for the resultmemory

Output of MAGIC is the sum or the difference of 2 of the inputs (the 2 being 2 different ones) or simply 1 of the inputs. '0' Must always be 'alone'.

Examples:

```
RESMEMADDRESS=0
RESMEMADDRESS=-RSAPM(0)
RESMEMADDRESS=-QAPM
```

All explanations to figure 2.3 are valid for figure 2.4 apart from the possibility to select LC1 as 'B' address.

2.3.3 The arithmetic

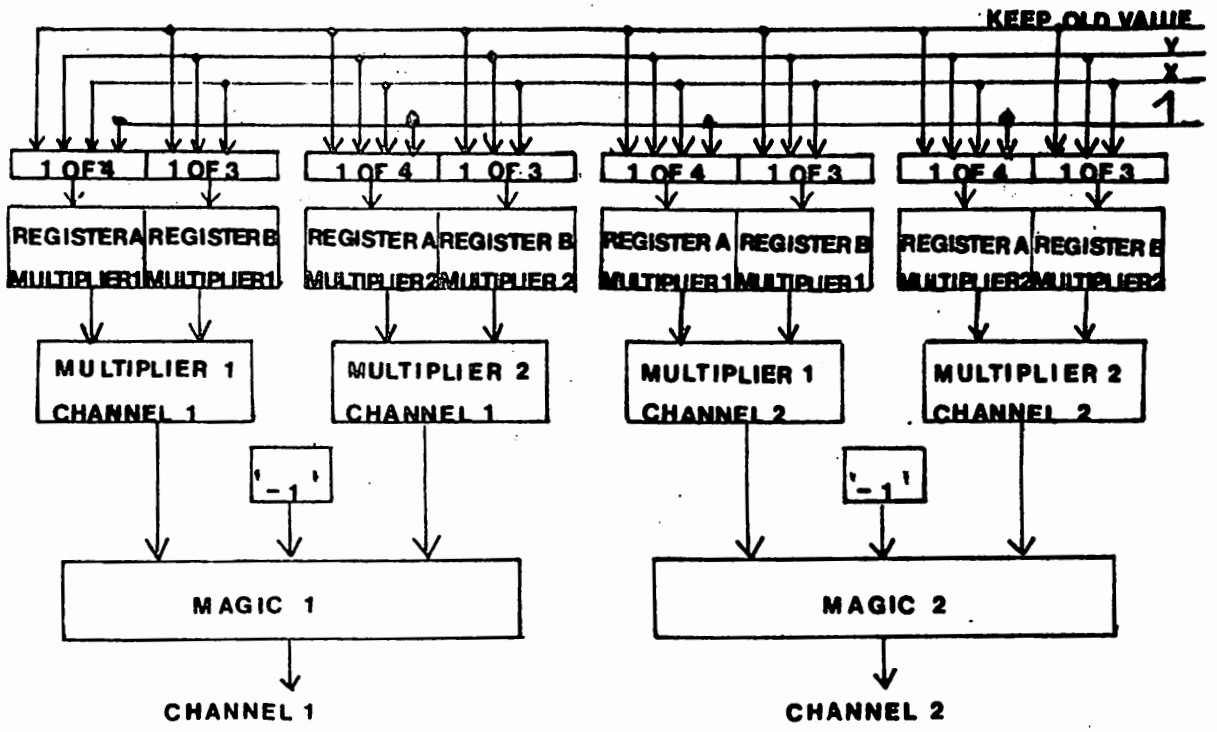


Figure 2.5 the arithmetic

X and Y are samples from the buffer memory (X = INPHASE and Y = QUADRATURE). The possible outputs of MAGIC1 and MAGIC2 are the same, but can be programmed independantly.

2.3.4 The accumulator

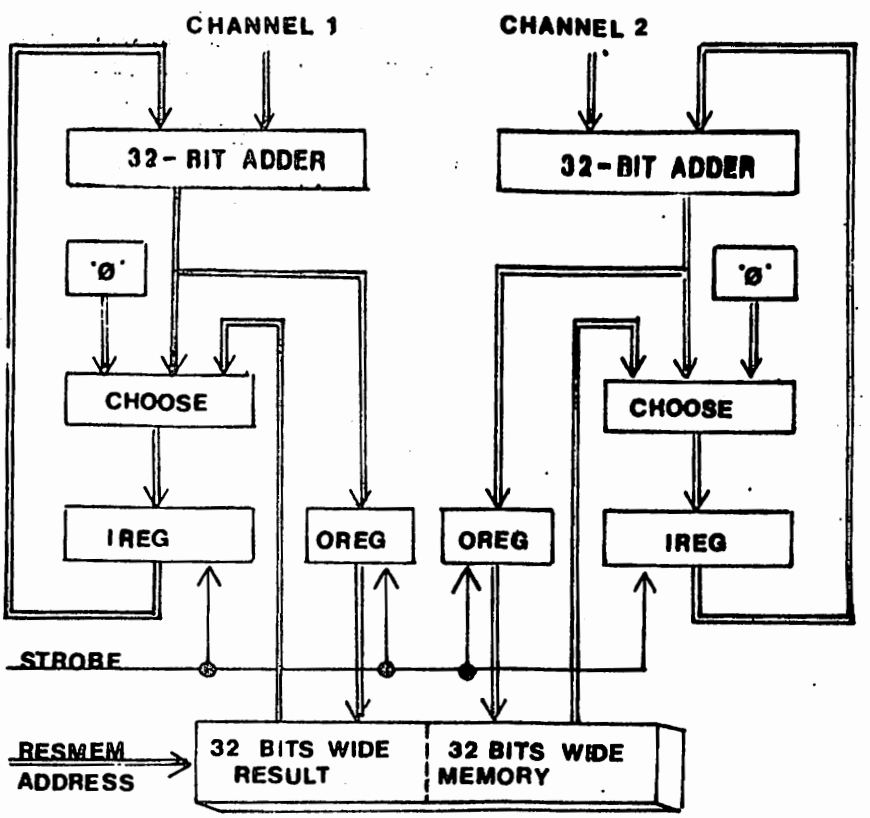


Figure 2.6 the accumulator

About the block called CHOOSE:
Assume START-EXPERIMENT MODE:
Having done INITIALIZE ACCUMULATOR and then doing a
LOAD IREG ==> IREG := 0
Having done ACCUMULATE and then doing a
LOAD IREG ==> IREG := value from the result memory
Indicated by the RESMEMADDRESS.

Doing only STROBE IREG ==> IREG := output of the
Accumulator, that is internal accumulation.
If in CONTINUE-EXPERIMENT MODE:
Doing a LOAD IREG ==> IREG:= value from the result
Memory (always) indicated by the RESMEMADDRESS.

IMPORTANT: after the last STORE OREG an extra STROBE OREG is required in the next instruction to set correct result.

The START-EXPERIMENT MODE and the CONTINUE-EXPERIMENT MODE are controlled by the so called flip-flop 1 (FF1).
The INITIALIZE ACCUMULATOR and the ACCUMULATE modes are controlled by another flip-flop, the famous flip-flop 2 (FF2).
Hence:

SET START-EXPERIMENT MODE ==> FF1:=0
SET CONTINUE-EXPERIMENT MODE ==> FF1:=1
INITIALIZE ACCUMULATOR ==> FF2:=0
ACCUMULATE ==> FF2:=1

Note that after a program has been loaded into the correlator by EROS, it is in 'START-EXPERIMENT-MODE'. It is the user's responsibility to set the correlator in 'START-EXPERIMENT-MODE' before a new integration periode is started. Setting this mode is usually done in the dma-dump routine.

2.3.5 The program flow control

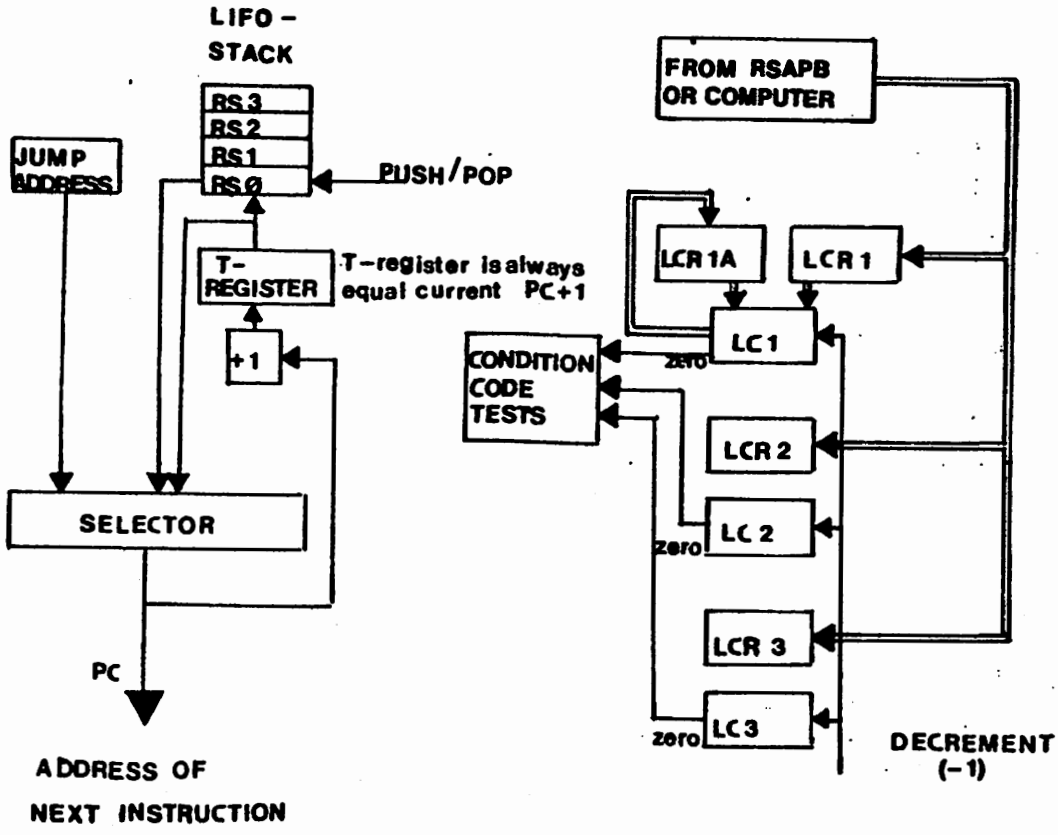


Figure 2.7 program flow control

Program memory is 64 instructions long and 128 bits wide. Like in any computer there are several ways to decide which instruction to execute next. In our case they are:

1) Continue to next instruction (PC:=PC+1)
2) Jump to a label (PC:=Jump address)
3) Return (PC:=RS0)
4) According to the state of the loopcounters (LC1,LC2 and LC3) select one of 1), 2) or 3).

3. PROGRAMMING IN CORLAN
3.1 Program flow control

```
<statement>::= <unconditionals> / <type1cond> /
               <type2cond> / <type3cond>

<type1cond>::= IF (<condition1>) THEN <labelless>
                                 ELSE <labelless> /
               IF (<condition1>) THEN <labelless>
                                 ELSE <withlabel> /
               IF (<condition1>) THEN <withlabel>
                                 ELSE <labelless>

<type2cond>::= IF (<condition2>) THEN <labelless>
               ELSEIF (<condition3>) THEN <labelless>
               OTHERWISE CONTINUE /

               IF (<condition2>) THEN <labelless>
               ELSEIF (<condition3>) THEN <withlabel>
               OTHERWISE CONTINUE /

               IF (<condition2>) THEN <withlabel>
               ELSEIF (<condition3>) THEN <labelless>
               OTHERWISE CONTINUE

<type3cond>::= IF (<condition4>) THEN <unconditionals>
               OTHERWISE CONTINUE

<condition1>::= LC1=0 / LC2=0 / LC3=0 / LC2#0 / LC3#0
                LC1=0 OR LC2=0 / LC1=0 OR LC3=0 /
                LC2=0 OR LC3=0 / LC1=0 OR LC2#0 /
                LC2#0 OR LC3=0 / LC1=0 OR LC3#0 /
                LC2=0 OR LC3#0 / LC2#0 OR LC3#0 /
                LC1=0 OR LC2=0 OR LC3=0 /
                LC1=0 OR LC2#0 OR LC3=0 /
                LC1=0 OR LC2=0 OR LC3#0 /
                LC1=0 OR LC2#0 OR LC3#0
```

<condition2> and <condition3> are allowed in certain combinations; a list of the legal combinations follows:

<condition2>::=	<condition3>::=
LC1=0 /	LC2=0 /
LC1=0 /	LC2#0 /
LC1#0 /	LC2=0 /
LC1#0 /	LC2#0 /
LC3=0 /	LC2=0 /
LC3=0 /	LC2#0 /
LC3#0 /	LC2=0 /
LC3#0 /	LC2#0 /
LC1=0 OR LC3=0 /	LC2=0 /
LC1#0 OR LC3#0 /	LC2=0 /
LC1=0 OR LC3=0 /	LC2#0 /
LC1#0 OR LC3#0	LC2#0

```
<condition4>::=LC1=0 OR LC3=0 / LC1#0 OR LC3#0
```

<unconditionals>::=<labelless> / <withlabel>

<labelless>::= CONTINUE / CONTINUEANDPOP /
CONTINUEANDPUSH /
GOBACK / GOBACKANDPOP / GOBACKANDPUSH /
RETURN / LOOP / ENDLOOP

<withlabel>::=GOTO <label> / GOTOANDPOP <label> /
GOTOANDPUSH <label> / CALL <label>

<label>::= SAR / as defined by means of LABEL or
SUBROUTINE directives.

3.1.1 About the program stack (LIFO-stack)

To help in the program control there is a LIFO-stack (last in first out) with 4 locations. (See figure under). POP means to take 1 entry off the stack. PUSH means to 'put' 1 entry onto the stack.

```
    ________                          ________
RS0 !__A__!                       RS0 !__B__!
RS1 !__B__!      POP ==>          RS1 !__C__!
RS2 !__C__!                       RS2 !__D__!
RS3 !__D__!                       RS3 !__X__!
```

X is undefined

```
    ________                          ________
RS0 !__A__!                       RS0 !_NEW_!
RS1 !__B__!     PUSH ==>          RS1 !__A__!
RS2 !__C__!                       RS2 !__B__!
RS3 !__D__!                       RS3 !__C__!
```

D is lost.

3.1.2 Explanation of the sequencing instructions

Refer to section 3.1.1 and figure 2.7.
PC is program-counter.

CONTINUE	PC:=PC+1
CONTINUEANDPOP	PC:=PC+1; POP stack
CONTINUEANDPUSH	PUSH T-REGISTER onto stack; PC:=PC+1
GOBACK	PC:=RS0
GOBACKANDPOP	PC:=RS0; POP stack
GOBACKANDPUSH	PC:=RS0; PUSH T-REGISTER onto stack
GOTO <label>	PC:=address of label
GOTOANDPOP <label>	PC:=address of label; POP stack
GOTOANDPUSH <label>	PUSH T-REGISTER onto stack; PC:=address of label
RETURN	same as 'GOBACKANDPOP'
CALL <label>	same as 'GOTOANDPUSH'
LOOP	same as 'CONTINUEANDPUSH'
ENDLOOP	same as 'GOBACK'

Examples of programming the 'program flow control':

```
IF (LC1 = 0) THEN CALL POWERPROFILE ELSE CONTINUE
IF (LC1 = 0 OR LC2 = 0) THEN GOTO LAB1 ELSE CONTINUE
IF (LC1 = 0) THEN CALL POWER ELSEIF (LC2 = 0) THEN
                  RETURN OTHERWISE CONTINUE
IF (LC1=0 OR LC3=0) THEN CONTINUE OTHERWISE RETURN
IF(LC1#0 OR LC3#0) THEN GOTO ABARYSTWYTH
                   ELSEIF (LC2#0) THEN GOBACK OTHERWISE CONTINUE
CONTINUEANDPOP
RETURN
CALL SUBR1
```

3.2 Programming the loopcounters

```
<lc1statement>::=
LC1=LC1-1 / LC1=LCR1A /
IF (LC1=0) THEN LC1=LCR1 LC2=LC2-1 ELSE LC1=LC1-1 /
IF (LC1=0 OR LC3=0) THEN LC1=LCR1A ELSE LC1 LC1-1 /
IF (LC1=0) THEN LC1=LCR1 ELSE LC1=LC1-1 /
IF (LC1=0) THEN LC1=LCR1A ELSE LC1=LC1-1

<lc2statement>:: LC2=LC2-1 / LC2=LCR2

<lc3statement>::= LC3=LC3-1 / LC3=LCR3 /
                  IF (LC3=0) THEN LC3=LCR3 ELSE LC3=LC3-1

<lcr1astatement>::= LCR1A=LC1
```

IMPORTANT:within one instruction the LCR1A=LC1 (if used) and testing on loopcounters is done BEFORE the other operations on the loopcounters.

3.3 Reloading registers from the APB-stack

```
<statement>::= RELOAD <register>
               RELOADVALUE=<output from apb>

<register>::=SAR / LCR1 / LCR2 / LCR3

<output from apb>::= RELOADVALUE has the same meaning as
                     BUFFERADDRESS (see section 3.4)
```

Examples of use:

```
RELOAD LCR1   RELOADVALUE=DATAI+QAPB
RELOAD LCR2   RELOADVALUE=QAPB
```

IMPORTANT:before using the reloaded register or reloading another register 1 DUMMY instruction is needed between.

```
CONTINUE
RELOAD LCR1   RELOADVALUE=DATAI

NEXT
CONTINUE              %THE DUMMY-INSTRUCTION

NEXT
LC1=LCR1
RELOAD LCR2
```

3.4 Programming the address processor for the buffermemory

```
<apbstatement>::=BUFFERADDRESS=<magic out>    /
                 BUFFERADDRESS=<magic out> <simple feedback> /
                 BUFFERADDRESS=RSAPB(n) <feedback>          /
                 <feedback>

<simplefeedback>::=QAPB=BUFFERADDRESS   /
                   RSAPB(n)=BUFFERADDRESS

<feedback>::=QAPB=<magic out>   / RSAPB(n)=<magic out>

<magic out>::=<operand> / <operand><operator><operand> / 0 /
              - <operand>

<operand>::=RSAPB(n) / RSAPB(m) / QAPB / DATAI

<operator>::=+ / -

<n>::=0..15 / <index>
<m>::=0..15 / <index>
<index>::=as defined by the INDEX-directive
```

IMPORTANT: only 2 different addresses to the stack may be referred in 1 instruction, hence

RSAPB(A)=RSAPB(B)+RSAPB(C)

Is illegal if A#B#C

Examples of use:

```
BUFFERADDRESS=DATAI
BUFFERADDRESS=QAPB+DATAI          QAPB=BUFFERADDRESS
BUFFERADDRESS=RSAPB(10)           RSAPB(9)=RSAPB(10)
QAPB=DATAI+RSAPB(0)
```

"BUFFERADDRESS=RSAPB(10) RSAPB(10)=RSAPB(11)"
is an illegal instruction even tho there are only two references to the stack. (Refer to figure 2.3)
Because we don't have RSAPB(10)=BUFFERADDRESS the first part of the statement must use the facility
BUFFERADDRESS=RSAPB(A) (A=10)
Only B-address can be written to so in
RSAPB(B)=RSAPB(11) (B=10)
There is no room for the address 11, so in effect this statement represents 3 references to APB-stack and is thus wrong.

3.5 Programming the address processor for the resultmemory

```
<apmstatement>::=RESMEMADDRESS=<magic out>    /
                 RESMEMADDRESS=<magic out>
                 <simple feedback>            /
                 RESMEMADDRESS=RSAPM(n)  <feedback> /
                 <feedback>

<simplefeedback>::=QAPM=RESMEMADDRESS   /
                   RSAPM(n)=RESMEMADDRESS
```

```
<feedback>::=QAPM=<magic out> / RSAPM(n)=<magic out>

<magic out>::=<operand> / <operand><operator><operand> / 0 /
                - <operand>

<operand>::=RSAPM(n) / RSAPM(m) / QAPM /

<operator>::=+ / -

<n>::=0..15 / <index>
<m>::=0..15 / <index>
<index>::=as defined by the INDEX-directive
```

IMPORTANT: only 2 different addresses to the stack may be referred in 1 instruction, hence

```
RSAPM(A)=RSAPM(B)+RSAPM(C)
```

Is illegal if A#B#C

Examples of use:

```
RESMEMADDRESS=QAPM                  QAPM=RESMEMADDRESS
RESMEMADDRESS=RSAPM(10)             RSAPM(9)=RSAPM(10)
QAPM=RSAPM(0)
```

IMPORTANT:see section 3.4 for example of illegal use of the stack. The same applies to the APM-processor.

3.6 Programming the arithmetic

```
<statement>::=CHANNEL1=<magic out>  / CHANNEL2=<magic out> /
               <register assignments>  /
               <multiplier definitions>

<magic out>::=-1 / <direct sumproduct> /
               <indirect sumproduct>

<direct sumproduct>::=<product> /
                      <product><operator><product>

<multiplier definitions>::= <multiplier><channel>=<product>

<product>::=<operand> / <operand>*<operand>

<register assignments>::=
                 <register><multiplier><channel> = <operand>

<register>::=REGISTERA / REGISTERB

<multiplier>::=MULTIPLIER1 / MULTIPLIER2

<channel>::=CHANNEL1 / CHANNEL2

<operand>::=X / OLDVALUE / Y

<indirect product> ::= MULTIPLIER1 / MULTIPLIER2 /
                       MULTIPLIER1+MULTIPLIER2 /
                       MULTIPLIER1-MULTIPLIER2
```

<operator>::=+ / -

Examples of use:
CHANNEL2=-1
CHANNEL1=Y*Y + X*X CHANNEL2=X-Y
REGISTERA MULTIPLIER1 CHANNEL1 = X
CHANNEL1=OLDVALUE*X + OLDVALUE*Y
CHANNEL1=MULTIPLIER1+MULTIPLIER2

3.7 Programming the accumulator

<statement>::= LOAD IREG / STORE OREG /
ACCUMULATE /
INITIALIZE ACCUMULATOR /
SET <mode> MODE /
STROBE IREG / STROBE OREG

<mode>::= START-EXPERIMENT / CONTINUE-EXPERIMENT

LOAD IREG is a bit 'queer':
Having done SET START-EXPERIMENT MODE:
the sequence INITIALIZE ACCUMULATOR
LOAD IREG
will cause IREG:=0

And the sequence ACCUMULATE
LOAD IREG
will cause IREG:=RESULTMEMORY(RESMEMADDRESS)

Having done SET CONTINUE-EXPERIMENT MODE:

Both sequencies above will cause
IREG:=RESULTMEMORY(RESMEMADDRESS)

Doing only STROBE IREG will cause IRED:= the output of the accumulator. This is called internal accumulation.

STORE OREG will cause the OREG to written to the result memory so that:

RESULTMEMORY(RESMEMADDRESS):=OREG

IMPORTANT:when the last STORE OREG has been done an extra single STROBE OREG is required to terminate the writing process correctly.

3.8 Transfer of data to the computer

<statement>::=PREPARETRANSFER / FINISHTRANSFER /
TRANSFER <what to transfer>

<what to transfer> ::= CONTROLWORD/ STATUSWORD /
TESTWORD1 / TESTWORD2 /
<channel> <part>

<channel>::=CHANNEL1 / CHANNEL2

<part>::=LSPART / MSPART

Examples of use:
PREPARETRANSFER
FINISHTRANSFER
TRANSFER CONTROLWORD

When running the correlator under EROS control the dump program has to be located from address 32 and upwards. This because the START-TRANSFER signal from the radar controller forces the program execution to start at location 32.
Note especially that 2 PREPARETRANSFER are needed before any TRANSFER .... is attempted. Similarly 4 FINISHTRANSFER are necessary after having dumped all the wanted data to terminate correctly.
See sample dump program for correct procedure.

4. DIRECTIVES FOR THE TRANSLATOR

4.1 NEXT and LOCATION

Since the correlator instruction word consists of instruction fields for many parts of the correlator which must be programmed individually, many statements mite be needed to program 1 full instruction.
From this arises a need to be able to tell the translator to which program location the following statements applies.
These are:

<directives>::=NEXT / LOCATION=<n>

<n>::=0..63

Examples of use:

```
******
******        %CODE PERTAINING TO LOCATION N
******
NEXT
******
******        %CODE FOR LOCATION N+1
******
NEXT
******
******        %CODE FOR LOCATION N+2
******

******
******        %CODE FOR LOCATION X
******
LOCATION=32
******
******        %CODE FOR LOCATION 32
******
```

4.2 INCLUDE

<statement>::= INCLUDE <file name>

<file name>::= any NORD-10 file with a CORLAN program on it. Extension :CLAN is assumed if none specified.

INCLUDE instructs the translator to suspend translation of the current file and continue with the one specified in INCLUDE-directive. After having finished this file, it will resume the translation of the original one.

INCLUDE cannot be nested, that is an includefile cannot contain an INCLUDE directive. Note that this is not checked by the translator.

Example of use:
INCLUDE DUMP-TO-NORD10

4.3 Defining labels (LABEL and SUBROUTINE directives)

```
<statement>::=LABEL <identifier> /
              SUBROUTINE <identifier>

<identifier>::=a name of maximum 15 characters, starting
               with a letter (A..Z) and thereafter
               consisting of any combination of letters,
               digits and underscores.
```

Examples of use:
LABEL ZERO
LABEL LOOP1
SUBROUTINE POWER_PROFILE

4.4 Assigning names to locations in the APB- and the APM-register stacks (INDEX)

```
<statement>::=INDEX <identifier>=<value>
              [{,<Identifier>=<value>}]

<identifier>::=<letter> / <identifier><alfanumeric>

<letter>::=A..Z
<alfanumeric>::=0..9 / A..Z / _
<value>::=0..15
```

Examples of use:
INDEX NLAGS=10, INCREMENT=4, ONE=14, NUMBER_OF_LAGS=3
Hence the following is possible:
RSAPB(TEMPORARY)=RSAPB(INCREMENT)+RSAPB(TEMPORARY)
BUFFERADDRESS=RSAPB(NUMBER_OF_LAGS)
CONSTANT RSAPB(ONE)=1

NOTE that a sequence enclosed in {} can be repeated indefinitely and a sequence enclosed in [] is optional.

4.5 Defining the correlator datafield (CONSTANT)

```
<statement>::=CONSTANT <target>=<number>
              [{,<target>=<number>}]
<target>::=SAR / DATAI / STATUS / RSAPM(<m>) / RSAPB(<m>)
<m>::=0..15 / <index>
<index>::= as defined by INDEX-directive
<number>::= for SAR    : 1..63
            for DATAI  : -32767..+32767
            for STATUS : -32767..+32767
            for RSAPB  : -32767..+32767
            for RSAPM  :  -4097..+4096
```

By default the DATAI, RSAPB(0..15) and the RSAPM(0..15) are all set to zero.

SAR is set to 1 and STATUS to octal 12000 by the translator.

Example of use:

```
CONSTANT SAR=1, DATAI=1024, RSAPB(10)=0, RSAPB(11)=4444,
         RSAPH(14)=999
```

(See explanation of brackets under 4.4)

4.6 The comment facility

A comment can be put in anywhere after a '%' (per-cent) sign. A comment is delimited by '%' and 'end of line'.

Example of use:

```
CONTINUE   %THIS COMMENT IS UP TO END OF LINE
```

5. SAMPLE CORLAN-PROGRAMS

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                             %
%       ROUTINE TO DUMP DATA FROM RESULT-MEMORY TO THE        %
%       COMPUTER.  MUST BE FROM LOCATION 32 BECAUSE THE       %
%       CORRELATOR STARTS EXECUTING THERE ON RECEIVING A      %
%       'START-TRANSFER' SIGNAL FROM THE RADAR-CONTROLLER.    %
%       (BAARD W. TORUSTAD 30.06.1982)                        %
%                                                             %
%       DATAI MUST CONTAIN NUMBER OF WORDS TO TRANSFER        %
%       RSAPM(0) = 1 (INCREMENT)                              %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
LOCATION=32
SUBROUTINE DUMPTONORD10
   RELOAD LCR1
   RELOADVALUE=DATAI
   CONTINUE

   NEXT
   CONTINUE
   PREPARETRANSFER

   NEXT
   CONTINUE
   PREPARETRANSFER

   NEXT
   CONTINUE
   TRANSFER STATUSWORD
   QAPM=0

   NEXT
   CONTINUE
   TRANSFER CONTROLWORD
   QAPM=QAPM-RSAPM(0)
   LC1=LCR1

   NEXT
   LABEL LOOP1
       CONTINUE
       LC1=LC1-1
       RESMEMADDRESS=RSAPM(0)+QAPM     QAPM=RESMEMADDRESS
       TRANSFER CHANNEL1 MSPART

       NEXT
       CONTINUE
       RESMEMADDRESS=QAPM
       TRANSFER CHANNEL1 LSPART

       NEXT
       CONTINUE
       RESMEMADDRESS=QAPM
       TRANSFER CHANNEL2 MSPART

       NEXT
       IF (LC1=0) THEN CONTINUE ELSE GOTO LOOP1
       RESMEMADDRESS=QAPM
       TRANSFER CHANNEL2 LSPART
```

```
    NEXT
    CONTINUE
    FINISHTRANSFER

    NEXT
    CONTINUE,
    FINISHTRANSFER

    NEXT
    CONTINUE
    FINISHTRANSFER

    NEXT
    GOTO ZERO
    FINISHTRANSFER
    SET START-EXPERIMENT MODE
END
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                    %
%       FIRST PROGRAM WRITTEN IN CORLAN (29.03.1981)                 %
%       IT TRANSFERS DATA FROM BUFFER-MEMORY TO RESULT-              %
%       MEMORY AND THEN DUMPS IT INTO THE COMPUTER.                  %
%                                                                    %
%       RSAPB(0) = MEMORY INCREMENT (=1)                             %
%       BMAPB(0) = MEMORY INCREMENT (=1)                             %
%       DATAI    = NUMBER OF WORDS TO TRANSFER                       %
%                                                                    %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
INDEX BUFFER_INCR=0, RESULT_INCR=0
%
CONSTANT RSAPB(BUFFER_INCR)=1, RSAPM(RESULT_INCR)=1, DATAI=1024,
%
LOCATION = 0
LABEL ZERO
CONTINUE
NEXT
GOTO ZERO       %ON NORMAL 'START-COMPUTE' GO BACK TO IDLE-LOOP
                %WITHOUT DOING ANYTHING
%
LOCATION=32     %ON 'START-TRANSFER' FROM RADAR-CONTROLLER THE
                %CORRELATOR STARTS EXECUTING IN LOCATION 32
CALL MOVEBUFRES
NEXT
CALL DUMPTONORD10
NEXT
GOTO ZERO
NEXT
SUBROUTINE MOVEBUFRES
    RELOAD LCR1
    RELOADVALUE=DATAI-RSAPB(BUFFER_INCR)
    CONTINUE
    NEXT
    CONTINUE
    QAPB=0    QAPM=0
    INITIALIZE ACCUMULATOR
    SET START-EXPERIMENT MODE

    NEXT
    CONTINUE
    QAPB=QAPB-RSAPB(BUFFER_INCR)    QAPM=QAPM-RSAPM(RESULT_INCR)
    LC1=LCR1

    NEXT
    LABEL LOOPONE
        IF (LC1 = 0) THEN CONTINUE ELSE GOTO LOOPONE
        LC1=LC1-1
        BUFFERADDRESS=QAPB+RSAPB(BUFFER_INCR)   QAPB=BUFFERADDRESS
        RESMEMADDRESS=QAPM+RSAPM(RESULT_INCR)   QAPM=RESMEMADDRESS

        CHANNEL1=X    CHANNEL2=Y
        LOAD IREG
        STORE OREG

        NEXT
        RETURN
        STROBE OREG              %STROBE LAST SAMPLE INTO MEMORY
                                 %CONFER PIPE-LINE...
```

```
NEXT
SUBROUTINE JUMPTONORD10
    RELOAD LCR1
    RELOADVALUE=DATAI
    CONTINUE

    NEXT
    CONTINUE
    PREPARETRANSFER

    NEXT
    CONTINUE
    PREPARETRANSFER

    NEXT
    CONTINUE
    TRANSFER STATUSWORD
    QAPM=0

    NEXT
    CONTINUE
    TRANSFER CONTROLWORD
    QAPM=QAPM.RSAPM(0)
    LC1=LCR1

    NEXT
    LABEL LOOP1
        CONTINUE
        LC1=LC1-1
        RESMEMADDRESS=RSAPM(0)+QAPM     QAPM=RESMEMADDRESS
        TRANSFER CHANNEL1 MSPART

        NEXT
        CONTINUE
        RESMEMADDRESS=QAPM
        TRANSFER CHANNEL1 LSPART

        NEXT
        CONTINUE
        RESMEMADDRESS=QAPM
        TRANSFER CHANNEL2 MSPART

        NEXT
        IF (LC1=0) THEN CONTINUE ELSE GOTO LOOP1
        RESMEMADDRESS=QAPM
        TRANSFER CHANNEL2 LSPART

    NEXT
    CONTINUE
    FINISHTRANSFER

    NEXT
    CONTINUE
    FINISHTRANSFER

    NEXT
    CONTINUE
    FINISHTRANSFER

    NEXT
    RETURN
    FINISHTRANSFER
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
%  CORLAN PROGRAM TO TEST RECEIVERS:    RXTEST-CORLAN:CLAN
%
%        APR(15) = GATES-1
%        APR(14) = BUFF. MEM. INCREMENT = 1
%
%        APM(0 ) = RES. MEM. INCREMENT  = 1
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
LOCATION=0
LABEL ZERO
   CONTINUE
NEXT
   CONTINUE
   RELOAD LCR1     RELOADVALUE=RSAPB(15)
NEXT
   QAPB=0    QAPM=0      CONTINUE
NEXT
   CONTINUE    INITIALIZE ACCUMULATOR    LC1=LCR1
   QAPB=QAPB-RSAPB(14)     QAPM=QAPM-RSAPM(0)
NEXT
LABEL DC
   IF(LC1=0)THEN CONTINUE ELSE GOTO DC       LC1=LC1-1
   BUFFERADDRESS=QAPB+RSAPB(14)    RESMEMADDRESS=QAPM+RSAPM(0)
   QAPB=BUFFERADDRESS              QAPM=RESMEMADDRESS
   CHANNEL1=X%X                    CHANNEL2=X
   LOAD IREG    STORE OREG
NEXT
   CONTINUE    QAPB=0 STROBE OREG
NEXT
   CONTINUE    LC1=LCR1    QAPB=QAPB-RSAPB(14)
NEXT
LABEL POWER
    IF(LC1=0)THEN CONTINUE ELSE GOTO POWER     LC1=LC1-1
   BUFFERADDRESS=QAPB+RSAPB(14)    RESMEMADDRESS=QAPM+RSAPM(0)
   QAPB=BUFFERADDRESS              QAPM=RESMEMADDRESS
   CHANNEL1=Y%Y                    CHANNEL2=Y
   LOAD IREG    STORE OREG
NEXT
   CONTINUE    QAPB=0    STROBE OREG
NEXT
   CONTINUE    LC1=LCR1    QAPB=QAPB-RSAPB(14)
NEXT
LABEL XY
   IF(LC1=0)THEN CONTINUE ELSE GOTO XY      LC1=LC1-1
   BUFFERADDRESS=QAPB+RSAPB(14)    RESMEMADDRESS=QAPM+RSAPM(0)
   QAPB=BUFFERADDRESS              QAPM=RESMEMADDRESS
   CHANNEL1=X%Y                    CHANNEL2=Y%X
   LOAD IREG    STORE OREG
NEXT
LABEL SCANCOUNT
   CONTINUE    RESMEMADDRESS=QAPM+RSAPM(0)
   CHANNEL1=-1    CHANNEL2=-1
   LOAD IREG STORE OREG
NEXT
   GOTO ZERO    SET CONTINUE-EXPERIMENT MODE    STROBE IREG
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
   INCLUDE (TORU)DUMP-TO-NORD10
END
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%                                                                 %%%
%%%    AURORA-CORLAN:CLAN PROGRAM (CESAR LA HOZ)                    %%%
%%%                                                                 %%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
LOCATION=0
LABEL ZERO
   CONTINUE
NEXT
   CONTINUE
   RELOAD LCR1    RELOADVALUE=RSAPB(8)
NEXT
   CALL POWER
   QAPB=0    QAPM=0
NEXT
   CONTINUE
   RELOAD LCR1    RELOADVALUE=RSAPB(11)
NEXT
   CONTINUE
NEXT
   CONTINUE
   RELOAD LCR2    RELOADVALUE=RSAPB(10)
NEXT
   CONTINUE
NEXT
   CONTINUE
   RELOAD LCR3    RELOADVALUE=RSAPB(9)
NEXT
   CALL SINGLEPULSE
   QAPB=QAPB+RSAPB(13)    QAPM=QAPM+RSAPM(0)
NEXT
   CONTINUE
   RELOAD LCR1    RELOADVALUE=RSAPB(15)
NEXT
   CONTINUE
NEXT
   CONTINUE
   RELOAD LCR2    RELOADVALUE=RSAPB(14)
NEXT
   CALL MULTIPULSE
   QAPB=QAPB+RSAPB(13)    QAPM=QAPM+RSAPM(15)
NEXT
   CONTINUE
   RELOAD LCR2    RELOADVALUE=RSAPB(7)
NEXT
   CALL MULTIPULSE
   QAPB=QAPB+RSAPB(13)    QAPM=QAPM+RSAPM(0)
NEXT
   CALL MULTIPULSE
   QAPB=QAPB+RSAPB(13)    QAPM=QAPM+RSAPM(0)
NEXT
   CONTINUE
   RESMEMADDRESS=RSAPM(0)+QAPM
   CHANNEL1=-1    CHANNEL2=-1    STROBE IREG    LOAD TREG    STORE OREG
   INITIALIZE ACCUMULATOR
NEXT
   GOTO ZERO
   SET CONTINUE-EXPERIMENT MODE
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
%
NEXT
 SUBROUTINE POWER
%
%    CALCULATES CH1=X**X AND CH2=Y**Y
%
%        (LCR1) <===  APB(B ) = HEIGHTS-1
%                     APB(13) = SAMPLE INCREMENT (=1)
%
%                     APM(0 ) = RESMEM INCREMENT (=1)
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
    CONTINUE
    LC1=LCR1
    QAPB=QAPB-RSAPB(13)    QAPM=QAPM-RSAPM(0)
    INITIALIZE ACCUMULATOR
NEXT
LABEL  POWLOC2
    IF(LC1=0)THEN RETURN ELSE GOTO POWLOC2
    LC1=LC1-1
    BUFFERADDRESS=QAPB+RSAPB(13)    QAPB=BUFFERADDRESS
    RESMEMADDRESS=QAPM+RSAPM(0 )    QAPM=RESMEMADDRESS
    CHANNEL1=X**X    CHANNEL2=Y**Y    STROBE IREG LOAD IREG STORE OREG
NEXT
%%%    PROG3: SINGLE PULSE NO. LAGS .LE. NO. SAMPLES  %%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%    (LCR3) <===  APB(9 ) = LAGS-1
%%%    (LCR2) <===  APB(10) = HEIGHTS-1
%%%    (LCR1) <===  APB(11) = SAMPLES-1 (IN RANGECELL)
%%%                 APB(12) = TEMPORARY STORAGE
%%%                 APB(13) = RANGECELL INCREMENT = 1 (NO OVERLAP)
%%%                         = SAMPLE    INCREMENT = 1
%%%
%%%
%%%                 APM(0)   = INCREMENT = 1
%%%                 APM(14) = TEMPORARY STORAGE
%%%                 APM(15) = NO. OF LAGS CALCULATED
%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%
SUBROUTINE SINGLEPULSE
    CONTINUE
    LC1=LCR1    LC2=LCR2    LC3=LCR3
    BUFFERADDRESS=QAPB-RSAPB(13)    QAPB=BUFFERADDRESS
    RESMEMADDRESS=QAPM-RSAPM(15)    QAPM=RESMEMADDRESS
NEXT
LABEL P3LOC2
    IF(LC1=0 OR LC3=0)THEN GOTO P3LOC7 ELSE CONTINUE
    LC1=LC1-1    LC3=LC3-1
    BUFFERADDRESS=RSAPB(13)+QAPB    QAPB=BUFFERADDRESS
    RESMEMADDRESS=RSAPM(15)+QAPM    QAPM=RESMEMADDRESS
    CHANNEL1=X**X+Y**Y    CHANNEL2=Y**X+X**Y  STROBE IREG LOAD IREG STORE OREG
    INITIALIZE ACCUMULATOR
NEXT
LABEL P3LOC3
    IF(LC1=0 OR LC3=0)THEN GOTO P3LOC5 ELSE CONTINUE
    LC1=LC1-1    LC3=LC3-1    LCR1A=LC1
    BUFFERADDRESS=RSAPB(13)+QAPB    RSAPB(12)=BUFFERADDRESS
    RESMEMADDRESS=RSAPM(0)+QAPM    RSAPM(14)=RESMEMADDRESS
    CHANNEL1=OLDVALUE**X+OLDVALUE**Y    CHANNEL2=OLDVALUE**X-OLDVALUE**Y
```

```
   STROBE IREG   LOAD IREG   STORE OREG
NEXT
 LABEL P3LOC4
   IF(LC1=0 OR LC3=0)THEN CONTINUE ELSE GOTO P3LOC4
   LC1=LC1-1   LC3=LC3-1
   BUFFERADDRESS=RSAPB(13)+RSAPB(12)   RSAPB(12)=BUFFERADDRESS
   RESMEMADDRESS=RSAPM(0)+RSAPM(14)   RSAPM(14)=RESMEMADDRESS
   CHANNEL1=OLDVALUE*X+OLDVALUE*Y   CHANNEL2=OLDVALUE*X-OLDVALUE*Y
   STROBE IREG   LOAD IREG   STORE OREG
NEXT
LABEL P3LOC5
   CONTINUE
   LC1=LCR1A   LC3=LCR3
NEXT
   IF(LC1=0)THEN CONTINUE ELSE GOTO P3LOC3
   IF(LC1=0)THEN LC1=LCR1 ELSE LC1=LC1-1
   LC3=LC3-1
   BUFFERADDRESS=RSAPB(13)+DAPB   DAPB=BUFFERADDRESS
   RESMEMADDRESS=DAPM
   CHANNEL1=X*X+Y*Y   CHANNEL2=Y*X+X*Y  STROBE IREG LOAD IREG STORE OREG
   ACCUMULATE
NEXT
 LABEL P3LOC7
   IF(LC2=0)THEN RETURN ELSE GOTO P3LOC2
   LC1=LCR1   LC2=LC2-1   LC3=LCR3
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
          INCLUDE (TORO)DUMP-TO-NORD10
%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%
LOCATION=5J
SUBROUTINE MULTIPULSE
%
%      (LCR1) <===  APB(15) = PULSES-1
%      (LCR2) <===  APB(14) = HEIGHTS-1
%                   APB(13) = SAMPLE INCREMENT
%                   APB(12) = TEMPORARY STORAGE
%%     (LCR1) <===  APB(7)  = (CAL-1)=(BACK-1) = 0 FOR 1 OF EACH
%   FOR FIVE PULSES:
%
%                   APB(3 ) = SAMPLE DISTANCE BETWEEN FIRST AND SECOND
%                   APB(2 ) =                         FIRST AND THIRD
%                   APB(1 ) =                         FIRST AND FOURTH
%                   APB(0 ) =                         FIRST AND FIFTH
%
%                   APM(0 ) = INCREMENT IN RESMEM (=1)
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
      LC1=LCR1
      LC2=LCR2
      RSAPB(12)=DAPB-RSAPB(13)
      DAPM=DAPM-RSAPM(0)
      INITIALIZE ACCUMULATOR
      CONTINUE
NEXT
 LABEL LOC2
      IF(LC1=0) THEN GOTO LOC6 ELSE CONTINUE
      IF(LC1=0) THEN LC1=LCR1  ELSE LC1=LC1-1
      BUFFERADDRESS=RSAPB(13)+RSAPB(12)   RSAPB(12)=BUFFERADDRESS
      CHANNEL1=X*X+Y*Y   CHANNEL2=Y*X+X*Y   STROBE IREG
```

```
NEXT
LABEL LOC3
      IF(LC1=0) THEN GOTO LOC5 ELSE CONTINUE
      IF(LC1=0) THEN LC1=LCR1A ELSE LC1=LC1-1
      LCR1A=LC1
      BUFFERADDRESS=RSAPB(12)+RSAPB(LC1)
      RESMEMADDRESS=RSAPM(0 )+QAPM    QAPM=RESMEMADDRESS
      CHANNEL1=OLDVALUE*X+OLDVALUE*Y    CHANNEL2=OLDVALUE*X-OLDVALUE*Y
      STROBE IREG    LOAD IREG    STORE OREG
NEXT
LABEL LOC4
      IF(LC1=0) THEN CONTINUE ELSE GOTO LOC4
      IF(LC1=0) THEN LC1=LCR1A ELSE LC1=LC1-1
      BUFFERADDRESS=RSAPB(12)+RSAPB(LC1)
      RESMEMADDRESS=RSAPM(0 )+QAPM    QAPM=RESMEMADDRESS
      CHANNEL1=OLDVALUE*X+OLDVALUE*Y    CHANNEL2=OLDVALUE*X-OLDVALUE*Y
      STROBE IREG    LOAD IREG    STORE OREG
NEXT
LABEL LOC5
      IF(LC1=0) THEN CONTINUE ELSE GOTO LOC3
      IF(LC1=0) THEN LC1=LCR1 ELSE LC1=LC1-1
      BUFFERADDRESS=RSAPB(12)+RSAPB(LC1)    QAPB=BUFFERADDRESS
      CHANNEL1=X*X+Y*Y        CHANNEL2=Y*X+X*Y     STROBE IREG
NEXT
      LABEL LOC6
      IF(LC2=0) THEN RETURN ELSE GOTO LOC2
      LC2=LC2-1
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
END
```

6. RUNNING THE TRANSLATOR

The translator is invoked in one of the following ways: (small letters are user-input, big letters prompts from the computer)

```
@(PREP)CORLAN
CORLAN - V04.82
SOURCE-FILE:<source-file>
LIST-FILE:<list-file>
CODE-FILE:<code-file>
or
@(PREP)CORLAN <source-file> <list-file> <object-file>
or
@(PREP)CORLAN <source-file>,,<object-file>
or
@(PREP)CORLAN <source-file> <list-file>,,
or
@(PREP)CORLAN <source-file>,,,
```

Where <source-file> is the name of the file where you have put your CORLAN-program. Extension :CLAN is assumed if none is specified. Thus if P1 is given, a file P1:CLAN must exist. However e.g P1:SYMB is also legal.
The <list-file> is the name of the file where you want the listing of the program and the possible error-messages to appear. The <list-file> can be TERMINAL, LINE-PRINTER or any other file-name. The <code-file> is where the code readable by EROS and CORRSIM is put. Extension CCOD is assumed if none given. If P1 is specified, a file P1:CCOD must exist. However e.g. P1:OBJ is also legal.
If neither or one of <list-file> and <code-file> is unwanted typing RETURN for the file-name will do. Typing RETURN for <list-file> will cause possible error-messages to appear on the terminal.
By enclosing the names for <list-file> and <code-file> in quotes (""), a new file will automatically be created.

The listing-output from CORLAN has three numbers in front of each line. The first number is the program-memory location in the correlator being programmed. The second number is number of source-code lines compiled so far. The third number gives the line number of this line on the source-file indicated by the heading on each page. If no INCLUDE is used, the second and the third number will be equal.

The following shows a sample run with a 'dummy'-program:

```
@CORLAN DEMO,TERM,,

CORLAN - V04.82  FILE:          DEMO TIME 11:26:03 DATE 22/JUL/82     PAGE   1
 0       1       1 >%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
 0       2       2 >%
 0       3       3 >%        A VERY LITTLE DEMONSTRATION-PROGRAM
 0       4       4 >%
 0       5       5 >%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
 0       6       6 >%
 0       7       7 >INDEX LOCATION_1=1, LOCATION_2=2
 0       8       8 >%
```

```
0      9      9 >CONSTANT DATAI=333, RSAPB(LOCATION_2)=-30
0     10     10 >%
0     11     11 >LOCATION=0                    %START PROGRAMMING IN LOCATION 0
0     12     12 >LABEL ZERO
0     13     13 >NEXT
1     14     14 >%
1     15     15 >CONTINUE
1     16     16 >RELOAD LCR1
1     17     17 >RELOADVALUE=RSAPB(LOCATION_1)
1     18     18 >%
1     19     19 >NEXT
2     20     20 >CONTINUE                      %A DUMMY STATEMENT AWAITING THE
2     21     21 >                              %RELOAD TO BE FULFILLED
2     22     22 >NEXT
3     23     23 >%
3     24     24 >LC1=LCR1
3     25     25 >QAPB=0
3     26     26 >QAPM=0
3     27     27 >NEXT
4     28     28 >%
4     29     29 >IF(LC1=0) THEN GOTO ZERO ELLERS CONTINUE
4     30     30 >NEXT
5     31     31 >%
5     32     32 >GOTO ZERO
5     33     33 >END

# ERROR IN LINE   29: UNEXPECTED SYMBOL
```

As can be seen there is an error in line 29. The unexpected symbol is ELLERS, which of course must be ELSE. (ELLERS is norwegian for ELSE).

EISCAT publications

F. du Castel, O. Holt, B. Hultqvist, H. Kohl and M. Tiuri:
A European Incoherent Scatter Facility in the Auroral Zone (EISCAT).
A Feasibility Study ("The Green Report") June 1971. (Out of print).

O. Bratteng and A. Haug:
Model Ionosphere at High Latitude, EISCAT Feasibility Study, Report No. 9.
The Auroral Observatory, Tromsö July 1971. (Out of print).

A European Incoherent Scatter Facility in the Auroral Zone, UHF System and Organization ("The Yellow Report"), June 1974.

EISCAT Annual Report 1976. (Out of print).

P.S. Kildal and T. Hagfors:
Balance between investment in reflector and feed in the VHF cylindrical antenna.
EISCAT Technical Notes No. 77/1, 1977.

T. Hagfors:
Least mean square fitting of data to physical models.
EISCAT Technical Notes No. 78/2, 1978.

T. Hagfors:
The effect of ice on an antenna reflector.
EISCAT Technical Notes No. 78/3, 1978.

T. Hagfors:
The bandwidth of a linear phased array with stepped delay corrections.
EISCAT Technical Notes No. 78/4, 1978.

Data Group meeting in Kiruna, Sweden, 18-20 Jan. 1978
EISCAT Meetings No. 78/1, 1978

EISCAT Annual Report 1977

H-J. Alker:
Measurement principles in the EISCAT system
EISCAT Technical Notes No. 78/5, 1978

EISCAT Data Group meeting in Tromsö, Norway 30-31 May, 1978
EISCAT Meetings No. 78/2, 1978.

P-S. Kildal:
Discrete phase steering by permuting precut phase cables.
EISCAT Technical Notes No. 78/6, 1978

EISCAT UHF antenna acceptance test.
EISCAT Technical Notes No. 78/7, 1978.

P-S. Kildal:
Feeder elements for the EISCAT VHF parabolic cylinder antenna.
EISCAT Technical Notes No. 78/8, 1978.

H-J. Alker:
Program CORRSIM: System for program development and software simulation of EISCAT digital correlator, User's Manual.
EISCAT Technical Notes No. 79/9, 1979.

H-J. Alker:
Instruction manual for EISCAT digital correlator.
EISCAT Technical Notes No. 79/10, 1979

H-J. Alker:
A programmable correlator module for the EISCAT radar system.
EISCAT Technical Notes No. 79/11, 1979.

T. Ho and H-J. Alker:
Scientific programming of the EISCAT digital correlator.
EISCAT Technical Notes No. 79/12, 1979.

S. Westerlund (editor):
Proceedings EISCAT Annual Review Meeting 1969. Part I and II,
Abisko, Sweden, 12-16 March 1979.
EISCAT Meetings No. 79/3, 1979.

J. Murdin:
EISCAT UHF Geometry.
EISCAT Technical Notes No. 79/13, 1979.

T. Hagfors:
Transmitter Polarization Control in the EISCAT UHF System.
EISCAT Technical Notes No. 79/14, 1979.

B. Törnstad:
A description of the assembly language for the EISCAT digital correlator.
EISCAT Technical Notes No. 79/15, 1979.

J. Murdin:
Errors in incoherent scatter radar measurements.
EISCAT Technical Notes No. 79/16, 1979.

EISCAT Digital Correlator. TEST MANUAL.
EISCAT Technical Notes No. 79/17, 1979.

G. Lejeune:
A program library for incoherent scatter calculation.
EISCAT Technical Notes No. 79/18, 1979.

K. Folkestad:
Lectures for EISCAT Personnel, Volume I
EISCAT Technical Notes No. 79/19, 1979.

Svein A. Kvalvik:
Correlator Buffer-Memory for the EISCAT Radar system
EISCAT Technical Notes. No. 80/20.

P-S. Kildal:
EISCAT VHF Antenna Tests
EISCAT Technical Notes No. 80/21

J. Armstrong:
EISCAT Experiment Preparation Manual
EISCAT Technical Notes No. 80/22

A. Farmer:
EISCAT Data Gathering and Dissemination
EISCAT Technical Note 80/23

Terrance Ho and Hans-Jørgen Alker:
Scientific Programming of the EISCAT Digital Correlator (Revised)
EISCAT Technical Note 81/24

Terrance Ho:
Programs Corrsim, Corrtest: System for Program Development and Software Simulation of EISCAT Digital Correlator. User's manual.
EISCAT Technical Note 81/25

Terrance Ho:
Instruction Manual for EISCAT Digital Correlator (Revised).
EISCAT Technical Note 81/26

Terrance Ho:
Standard Subroutines and Programs for EISCAT Digital Correlator.
EISCAT Technical Note 81/27

Terrance Ho:
Pocket Manual for Programming the EISCAT Digital Correlator.
EISCAT Technical Note 81/28

K. Folkestad:
Lectures for EISCAT Personnel, Volume II.
EISCAT Technical Note 81/29

M. Lehtinen och Anna-Liisa Turunen:
EISCAT UHF antenna direction calibration
EISCAT Technical Note 81/30

K. Folkestad:
Use of the EISCAT Radar as a supplement to rocket measurements.
EISCAT Technical Note 81/31

T. Turunen, T Mustonen and P J S Williams:
EISCAT UHF RECEIVERS: Report and Recommendations
EISCAT Technical Note 81/32

Phil Williams:
Polarisers in the EISCAT System
EISCAT Technical Note 81/33

R. Gras:
THE EISCAT CORRELATOR
EISCAT Technical Note 82/34

W. Kofman:
A MESOSPHERIC EXPERIMENT: Measurement Scheme and Program Implementation
EISCAT Technical Note 82/35