Introduction to RC COMAL

3600 / 7000

Introduction to
RC BASIC/COMAL
A Structured Programming Language

First Edition
April 1978
RCSL 42-i 0784

A/S REGNECENTRALEN
Information Department

Author: Thorkild Maaetoft and Børge Christensen
Text Editor: Inga Marcussen

KEY WORDS: RC 7000, RC 3600, RC COMAL, BASIC, Introduction, Brief language summary.

ABSTRACT: This introduction defines the extensions, regarding the BASIC generally used, which are implemented in RC COMAL. The special use of the floppy disc and card reader are also described. The final chapters contain a summary of RC COMAL.

REMARKS: Issued by A/S Regnecentralen in co-operation with the Computer Divison of the Tønder Statsseminarium.

Contents

1 WHAT IS RC COMAL Page 5

2 NAMES OF VARIABLES, ASSIGNMENT OF VARIABLES 7
- 2.1 Names of variables 7
- 2.2 Assignment of variables 7

3 ADVANCED CONTROL STRUCTURES 8
- 3.1 If p THEN .. ELSE .. ENDIF 8
- 3.2 If p THEN .. ENDIF 9
- 3.3 REPEAT .. UNTIL p 10
- 3.4 WHILE p DO .. ENDWHILE 11
- 3.5 CASE expr OF .. WHEN .. ENDCASE 12
- 3.6 Procedures 14

4 OPERATORS 16
- 4.1 Boolean expressions 16

5 TERMINAL COMMANDS 18

6 SYSTEMS WITH A CARD READER 19

7 SYSTEMS WITH FLOPPY DISCS 21

8 SUMMARY OF RC COMAL 23
- 8.1 RC COMAL statement types 24
- 8.2 RC COMAL standard functions 28
  - 8.2.1 Functions for processing text strings 30
- 8.3 Matrix operations in RC COMAL 30
- 8.4 Commands to logical discs 32
- 8.5 Statements for file processing 33
- 8.6 System commands in RC COMAL 35
- 8.7 Commands in connection with batch runs 37

1 What is RC COMAL

Since 1970 Regnecentralen has marketed the RC 7000 Minicomputer System, primarily for teaching institutions.

Today, in 1978, there are RC 7000's installed in over 100 schools, colleges, universities and other educational institutions, which makes it the most used mini-computer system for educational purposes.

Originally the RC 7000 was based on the hardware and software of the American firm Data General. The most frequently used programming language was BASIC, due to the many benefits contained in this language, especially for elementary teaching.

The RC 7000 is an all-Danish computer as both hardware and software are developed and produced in Denmark by Regnecentralen. As early as 1974, users of the RC 7000 expressed their wish for an extension of Data General's extended BASIC. It was found that BASIC had some general shortcomings which made it less suitable, for example, for procedure orientated structures.

The programming language COMAL was defined by Lecturer Benedict Løfstedt, Århus University, and Lecturer Børge Christensen, Tønder Statsseminarium (Teacher Training College). The language contains extended BASIC, as the aim was that COMAL should be a further development of BASIC.

A first version of COMAL was developed on an RC 7000 by Per Christiansen and Knud Christensen at the computer division of the Tønder Statsseminarium in 1975. More than three years experience with COMAL EDP teaching at commercial schools, teacher training colleges and in ordinary schools has shown that COMAL is easier to learn than BASIC, one of the reasons being that it results in programs which have a clearer structure and are therefore easier to read than the corresponding BASIC programs. By reading through this introduction to RC COMAL, the reader will understand why this is so.

RC COMAL is the final version of the COMAL language, and has been developed by A/S Regnecentralen in continuous co-operation with teachers from teacher training colleges, high schools and other schools. RC COMAL contains significant improvements in relation to the originally defined COMAL, and these improvements all aim at making the language into an easy and flexible tool for teaching in all places of education.

Incidentally, COMAL is an abbreviation of: COMmon Algorithmic Language. In the definition of the language, emphasis has been put on the basic algorithm structures being easy to describe and easy to recognise.

This introduction is not meant as a textbook of RC COMAL. The introduction has been written for readers who know the most important language elements in BASIC, and contains therefore no definitions or explanations of the most simple RC COMAL/BASIC sentences.

This book contains first a description of extensions and new language elements in RC COMAL in relation to BASIC. The last chapter is a total survey, in diagrammatic form, of RC COMAL.

Additional literature about RC COMAL:

"RC COMAL Programming Guide", published by A/S Regnecentralen.
Børge Christensen: "RUN COMAL", published by Studentlitteratur.
Børge Christensen: Problems for "RUN COMAL", by the same publisher.

2 Names of variables, assignment of variables

2.1 Names of variables

In RC COMAL, variables of all types are named according to the prescription:

$$t_1 t_2 \; \ldots\ldots. \; t_i, \; i \leqq 8,$$

where t_1 is a letter, while t_2, ... t_i are letters or digits.

Examples

```
LET INTEREST = CAPITAL * INTEREST RATE * NUMDAYS/360/100
LET NAME$ = "OLE OLSEN"
LET TABLE (NUMBER, YEAR) = 126
```

2.2 Assignment of variables

In RC COMAL, LET sentences may contain more than one assignment:

LET var_1 = $expr_1$; var_2 = $expr_2$; .....; var_n = $expr_n$

where var_1, var_2, ..., var_n are names of variables, while $expr_1$, $expr_2$, ..., $expr_n$ are constants, variables or formulae. A LET sentence may contain as many assignments as the length of the line permits.

Examples

```
LET INTEREST RATE = 12; CAPITAL = 15000; NAME$ = "OLE OLSEN"
```

This sentence has the same effect as three consecutive LET sentences, each with its own assignment.

3 Advanced control structures

In addition to the control structures in BASIC, the following are found in RC COMAL:

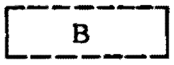
(1) IF p THEN .. ELSE .. ENDIF
(2) IF p THEN .. ENDIF
(3) REPEAT .. UNTIL p
(4) WHILE p DO .. ENDWHILE
(5) CASE expr OF .. WHEN .. ENDCASE

3.1 If p THEN .. ELSE .. ENDIF

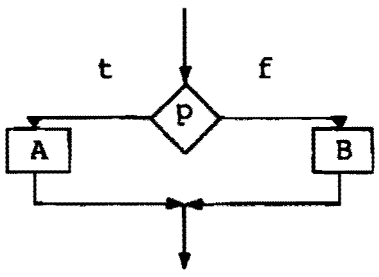
The structure is built up as follows:

IF p THEN

ELSE



ENDIF



p is a Boolean expression (an open statement); if p has the value "true", then the program portion A described between IF p THEN and ELSE is carried out, and if p has the value "false", the program portion B described between ELSE and ENDIF is carried out. When either A or B has been carried out, the program continues with the next sentence after ENDIF.

The portion of the program text positioned between the control sentences is inserted when a program list has been printed out (see FOR .. NEXT in BASIC).

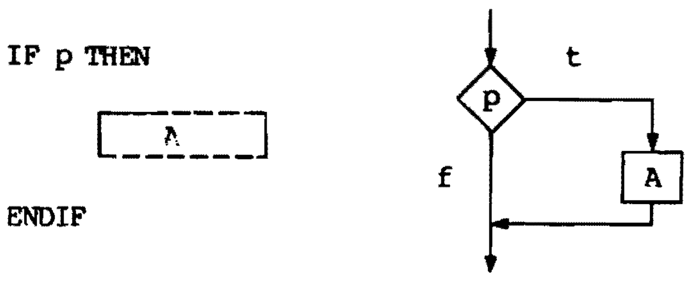
Examples

```
IF PRICE < 100 THEN
  PRINT "YOU MUST PAY (INCL. CHARGE): "; PRICE + 15;"KR."
ELSE
  PRINT "YOU MUST PAY: "; PRICE; "KR."
ENDIF

INPUT "CHARACTER: ", CHARAC
LET SUM = SUM + CHARAC; NUMCAR = NUMCAR + 1
IF CHARAC < MIN THEN
  LET NEXTMIN = MIN; MIN = CHARAC
ELSE
  IF CHARAC < NEXTMIN THEN LET NEXTMIN = CHARAC
ENDIF
```

3.2 If p THEN .. ENDIF

The structure is built up as follows:



p is a Boolean expression; if p has the value "true", then the program section A described between IF p THEN and ENDIF is carried out, and if p has the value "false", then A is skipped. In either case, the program continues with the next sentence after ENDIF. The text between IF 9 THEN and ENDIF is inserted when program lists are printed out.

In an RC COMAL program, up to seven IF p THEN .. (ELSE) .. ENDIF branches inside one another are allowed.

Examples

```
READ TAL1, TAL2
IF TAL1 > TAL2 THEN
  LET BUFFER = TAL1; TAL1 = TAL2 = BUFFER
ENDIF
PRINT TAL1, TAL2
```

```
IF RND (0) < 3/10 THEN
  LET H1 = H1 + 1
ELSE
  IF RND (0) < 3/9 THEN
    LET H2 = H2 + 1
  ELSE
    IF RND (0) < 3/8 THEN
      LET H3 = H3 + 1
    ELSE
      IF RND (0) < 3/7 THEN
        LET H4 = H4 + 1
      ELSE
        LET H5 = H5 + 1
      ENDIF
    ENDIF
  ENDIF
ENDIF
```

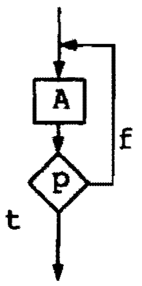
3.3 REPEAT .. UNTIL p

The structure is built up as follows:

```
REPEAT

      A

UNTIL P
```



p is a Boolean expression; the program section A described between REPEAT and UNTIL p is repeated until p has the value "true". When p obtains this value, the program continues with the next sentence after UNTIL p.

It should be noted that the program section A is carried out at least once if the interpreter reaches the REPEAT sentence, because the control of the loop starts from the UNTIL sentence. The text between REPEAT and UNTIL p is inserted when a program list is printed out.

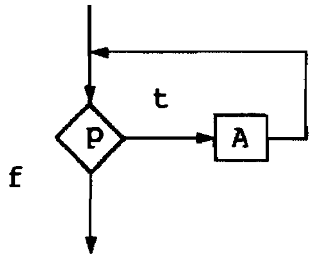
3.4 WHILE p DO .. ENDWHILE

The structure is built up as follows:

```
WHILE p DO

   [-----A-----]

ENDWHILE
```



p is a Boolean expression; the program section A described between WHILE p DO and ENDWHILE is repeated as long as p has the value "true". When p has the value "false" the program continues with the next sentence after ENDWHILE.

It should be noted that if p has the value "false" when the program reaches the WHILE sentence, the section A will not be carried out at all, and the interpreter will just go straight on to the next sentence after ENDWHILE. The text between WHILE p DO and ENDWHILE will be inserted when a program list is printed.

In RC COMAL it is possible to have up to seven REPEAT .. UNTIL loops and up to seven WHILE .. ENDWHILE loops inside and independent of one another. For example, there may be a total of fourteen loops of the two types inside one another independent of sequence, as long as seven of them are REPEAT loops and the rest are WHILE loops.

The FOR .. NEXT loop inherited from BASIC has no influence on the number of REPEAT and WHILE loops that there may be inside one another (in RC COMAL there may be up to seven FOR .. NEXT loops inside one another). The number of IF .. (ELSE) .. ENDIF branches is also independent of any possibly inserted loops. In other words, it is actually possible to have a total of 21 loops + 7 branches built into one another in an RC COMAL program. However, the author does not remember having seen this in practical application.

Examples

```
INPUT "THE TWO NUMBERS:", A, B
LET X = A; Y = B
REPEAT (* EUCLID*)
  LET REST = X TOWARD Y
  LET X = Y; Y = REST
UNTIL REST = 0
PRINT "SFD FOR"; A; "AND"; B; "IS"; X
```

```
INPUT "NAME" (SURNAME CHRISTIAN NAME): ", NAME$
WHILE NAME$ <> "END" DO
  INPUT "ADDRESS:", ADR$
  INPUT "TOWN (POSTNUMBER NAME OF TOWN): ", TOWN$
  INPUT "NEXT NAME:", NAME$
ENDWHILE
```

<u>Note</u>

The use of GOTO sentences in connection with the control structures (1) - (4) may be problematic. Generally GOTOs should not be used in the program sections A and B and certainly not if these GOTO sentences refer to sentences which are outside the sentences that start or finish the said program section. On the whole, the use of GOTO should be restricted to unusual situations (for example in the case of fault in input, etc.).

3.5 CASE expr OF .. WHEN .. ENDCASE

The structure is built up as follows:

CASE expr OF

$\boxed{\quad A \quad}$

WHEN $\langle\text{list}\rangle_1$

$\boxed{\quad A_1 \quad}$

WHEN $\langle\text{list}\rangle_2$

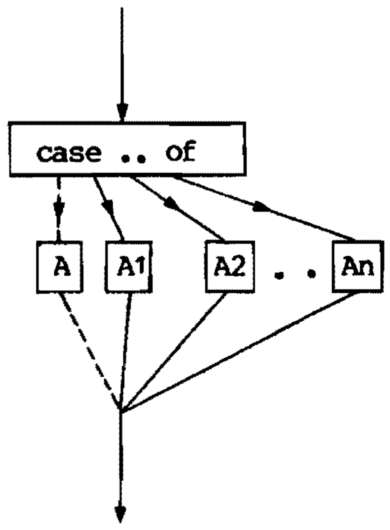
$\boxed{\quad A_2 \quad}$

.
.
.

WHEN $\langle\text{list}\rangle_n$

$\boxed{\quad A_n \quad}$

ENDCASE



expr is an arithmetical expression (a constant, a variable or a formula), one of the Boolean constants TRUE or FALSE, or a variable string, and $\langle\text{list}\rangle_i$ is a list of arithmetical expressions, a list of Boolean expressions, or a list of character sequences.

When the value of expr has been calculated, the interpreter searches in the lists after the various WHEN for a value which is equal to that of expr. If such a value is found in <list>$_i$, the following program section A$_i$ will be carried out and then the program will continue with the sentence after ENDCASE. If the relevant value is not found, the alternative section A, which is placed immediately after the CASE sentence will be carried out, and then the program will continue with the next sentence after ENDCASE.

The number of WHEN sentences following a produced CASE sentence is unlimited, and you may have as many CASE .. ENDCASE inside one another as you like.

The texts for A, A$_1$, A$_2$, ... A$_n$ are inserted in relation to the CASE, WHEN and ENDCASE sentences when a program list is printed out.

Examples

```
INPUT "1 = IN, 2 = LIST, 3 = SEARCH, 4 = DELETE, 5 = STOP:",
JOBCODE
CASE JOBCODE OF
  PRINT "THE CODE DOES NOT EXIST."
WHEN 1
  REM (*NEW CODES INPUT*)
  EXEC ENCODER
WHEN 2
  REM (*PRINTOUT OF CODE LIST*)
  EXEC PRINTOUT
WHEN 3
  REM (*SEARCH A GIVEN CODE*)
  EXEC SEARCH
WHEN 4
  REM (*DELETE A GIVEN CODE*)
  EXEC DELETE CODE
WHEN 5
  REM (*END OF DAY*)
  STOP
ENDCASE
```

```
INPUT SVAR$
CASE SVAR$ OF
  PRINT "READ INSTRUCTIONS PROPERLY"
WHEN "YES"
  EXEC PANE1
WHEN "NO"
  EXEC PANE2
WHEN "NONE", "NOTHING", "NOT ANY"
  EXEC PANE3
ENDCASE
```

```
INPUT "THE COEFFICIENTS A, B AND C", A,B,C
IF A <> 0 THEN
  LET DETM = B* B-4* A*C
  CASE TRUE OF
  WHEN DETM > 0
    LET X1 = (-B+SQR (DETM))/2/A; X2 = (-B-SQR (DETM))/2/A
    PRINT"X1 ="; X1, "X2 ="; X2
  WHEN DETM = 0
    LET X =-B/2/A
    PRINT "DOUBLEROOT: X ="; X
  WHEN DETM <0
    PRINT "NO REAL ROOTS".
  ENDCASE
ELSE
  PRINT "I THOUGHT WE WERE TO SOLVE QUADRATIC EQUATIONS?"
ENDIF
```

3.6 Procedures

It is a great advantage to use procedures to make the programs clear and logical in their construction. This is why RC COMAL also has an extension in this area.

If a program starts with the sentence

```
PROC <name>
```

where <name> is a character sequence with the same format as the name of a variable (see section 2), and ends with the sentence

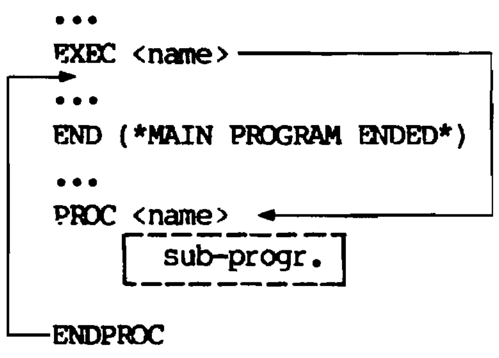
```
ENDPROC
```

this program may be called as a sub-program of another program by using the sentence

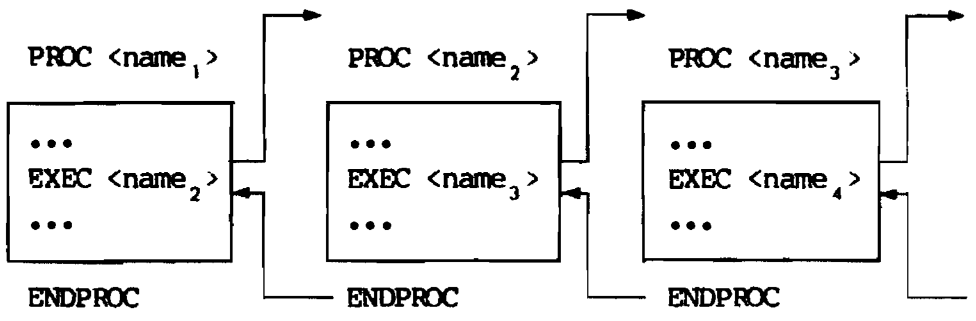
```
EXEC <name>
```

When the sub-program has been run, the interpreter continues with the sentence following immediately after the EXEC sentence from which the call to the sub-program was made.

We may illustrate this in the following way:



From a sub-program a new sub-program may be called, and so on to a total of seven. This phenomenon provides the following picture:



4 Operators

In addition to the usual arithmetical and logical operators, RC COMAL contains the following facilities:

DIV	:	Whole number division, e.g. 11 DIV 4 (=2)
MOD	:	Modulus, e.g. 11 MOD 4 (=3)
AND	:	Logic "AND", e.g. (A > 10) AND (A < 20)
OR	:	Logic "OR", e.g. (A <20) OR (A > 30)
NOT	:	Logic negation, e.g. NOT (A <10)

While the two first, DIV and MOD are a convenient and logical extension of the arithmetical operators, the last three (AND, OR and NOT) provide a number of special applications in connection with Boolean expressions, which will be explained in more detail.

4.1 Boolean expressions

In RC COMAL, Boolean expressions may be formed in the usual Boolean algebra by using the operators AND, OR and NOT.

Examples

```
IF MAXNR = 0 OR LAST = 10 THEN
UNTIL NAME$ = "NONE" OR END = TRUE
WHILE SQR (OBS) <> 25 AND OBS <> 0 DO
UNTIL NODE = TRUE AND H(NODE) = 0 OR H(NODE) = LAST
```

In RC COMAL numerical variables may be used as (pseudo) Boolean variables, because a numerical variable in the right context will be understood as a Boolean variable with the value "false" if the numerical variable in question has the numerical value 0 and the value "true" in all other cases. In other words, "true" corresponds to "different from 0", while "false" corresponds to "equal to 0". To make it easier to remember this equivalent, we have introduced in RC COMAL the constants TRUE and FALSE, which have the numerical values 1 and 0 respectively.

Examples

When the sentence

```
LET WORKOUT = TRUE
```

has been carried out, WORKOUT has the value 1 and is therefore in this sentence

```
IF WORKOUT THEN PRINT "THE DIVISION WORKS OUT"
```

interpreted as a Boolean variable of the value "true".

Due to the stated equivalence between numerical values and trueness values, Boolean expressions may be used in arithmetical expressions. A Boolean expression with the value "true" is in this connection interpreted as an arithmetical expression of the value 1, while a Boolean expression which has the value "false" is assigned the numerical value 0.

Examples

In the sentence

```
DEF FNF (X) = (X<0)* (X+1) + (X=0) * X+(X>0) * (2*X+3)
```

the function FNF is given after a "division of the definition volume".

If CLOSED and DOGLOOSE are variables of the values 1 (TRUE) and 0 (FALSE) respectively, the execution of the sentence

```
LET IND = CLOSED AND NOT DOGLOOSE
```

will result in the variable IND being assigned the value 1 (TRUE).

5 Terminal commands

A number of new terminal commands in addition to the usual (RUN, LIST etc.,) have been added in RC COMAL.

The following should be mentioned:

AUTO :	Used when keying in programs. The command starts the automatic generation of line number: 10, 20, 30, ... etc.
RUNL :	Starts the run of a program the same way as RUN, but all printouts will be on the line printer.
CON :	Starts a program from the point where it was last stopped. The program may be stopped by means of a STOP statement, pressing ESC key, etc..
CONL :	As CON, but continues printout on the line printer.
BATCH:	Starts input of programs from the card

6 Systems with a card reader

An RC 7000 system with a card reader provides facilities for BATCH runs, i.e., automatic run of, for example, the programs of a whole class.

The cards are packed in a JOB (JOB = program + control cards) and the various JOBs are packed together in a BATCH (BATCH = stack).

Each JOB is provided, for example, with the following control cards:

1. SCRATCH <text>, where <text> is printed out before output from the program.
2. LIST, prints out the program.
3. RUN, starts the execution of the program.
4. EOJ, (End-Of-Job) completes the individual job.

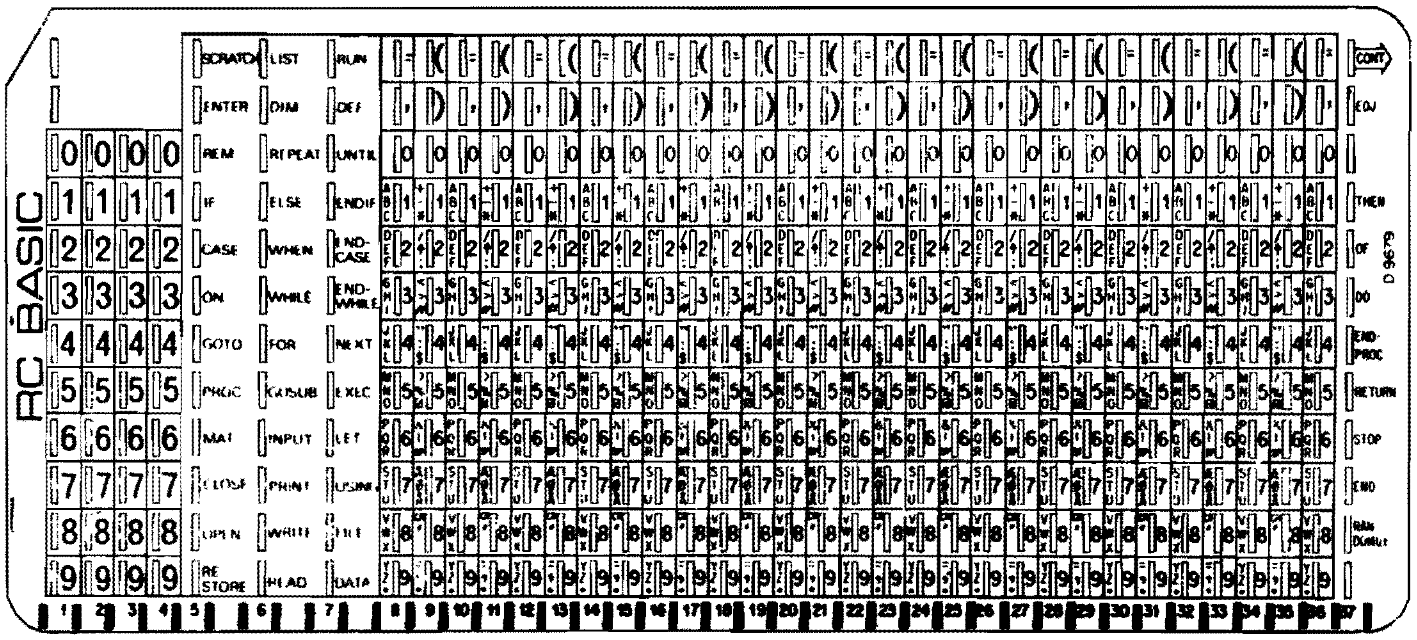
When the individual JOBs are stacked together in a BATCH, the BATCH is placed in the card reader.

There are two terminal commands which may start running the cards:

BATCH :	All printouts will appear at the terminal from which the command has been given.
BATCH "$LPT":	All printouts appear on the line printer.

The card run is now fully automatic. If the programs contain faults and cannot be run within a pre-determined time limit, e.g., 60 seconds, the next job will be input automatically. It should be noted that all other terminals and peripheral equipment are not affected by the BATCH run, but continue undisturbed.

Two types of line marking cards may be used in RC COMAL, the BATCH-BASIC card (HP-standard) and a newly designed RC COMAL card. The new card contains all the types of RC COMAL statements. Both these cards are shown on the next page.



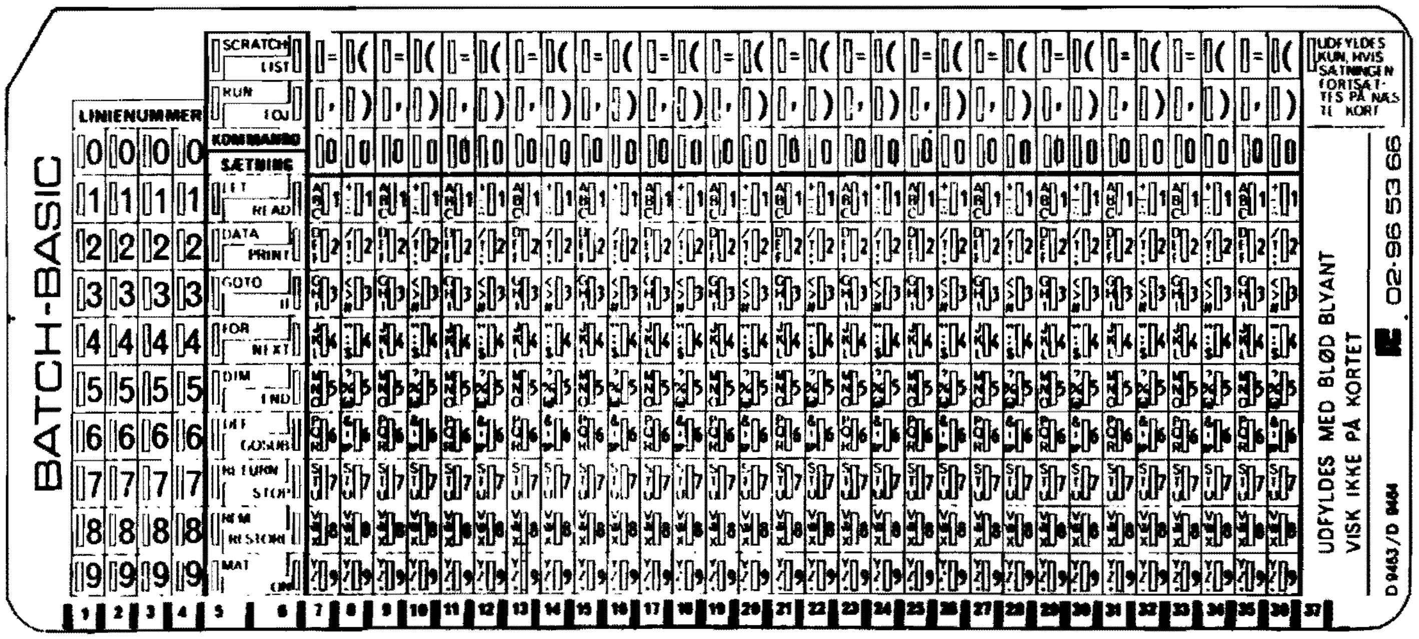
RC BASIC

SCRATCH	LIST	RUN
ENTER	DIM	DEF
REM	REPEAT	UNTIL
IF	ELSE	ENDIF
CASE	WHEN	END-CASE
ON	WHILE	END-WHILE
GOTO	FOR	NEXT
PROC	GOSUB	EXEC
MAT	INPUT	LET
CLOSE	PRINT	USING
OPEN	WRITE	FILE
RESTORE	READ	DATA

CONT, EOJ, THEN, OF, DO, END-PROC, RETURN, STOP, END, RANDOMIZE

D 9679

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37



BATCH-BASIC

LINIENUMMER

SCRATCH	LIST
RUN	EOJ
KOMMANDO	
SÆTNING	
LET	READ
DATA	PRINT
GOTO	IF
FOR	NEXT
DIM	END
DEF	GOSUB
RETURN	STOP
REM	RESTORE
MAT	ON

UDFYLDES KUN, HVIS SÆTNINGEN FORTSÆTTES PÅ NÆSTE KORT

UDFYLDES MED BLØD BLYANT
VISK IKKE PÅ KORTET

02·96 53 66

D 9463/D 9464

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37

7 Systems with floppy discs

RC COMAL contains software for handling FLOPPY DISCS.

The floppy disc has three primary fields of application:

1. System storage (e.g. RC COMAL)
2. Program storage
3. Data files.

When an RC 7000 SYSTEM is provided with a floppy disc, it would be natural to use this for system start. This gives an easy and convenient system start, which takes about 30 seconds. The main application of the disc would probably be for program storage. Programs are stored and called by means of a name of up to eight characters. The number of programs which can be stored depends on diskette's capacity.

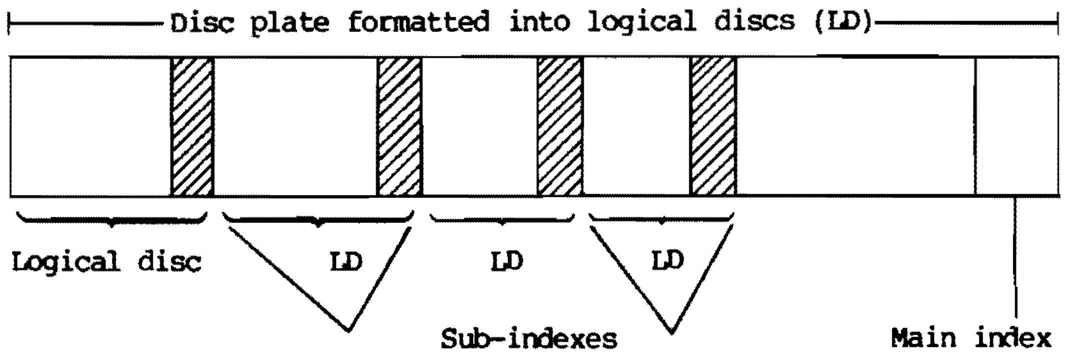
Finally, the disc may contain data files. Reading and printing of data files may be carried out directly from the individual program.

A new concept in connection with the floppy disc is the LOGICAL DISC.

Before a disc plate is used for the first time, it must be formatted, i.e., divided into a number of sub-sections called logical discs.

The number of logical discs is dependent only on the size of the individual logical discs that make up the whole disc. In practical applications, the number may vary from 1 to about 70 logical discs.

The formatted disc plate looks as follows:



The main index describes the logical discs. This index contains information on names of the logical discs, protection keys, the length of the logical disc and the number of users.

The contents of the main index will not be readily accessible to "lay" users, but can be written out by means of a special program.

The sub-indexes contain information on the content of the logical discs. The primary information in the index is the names of the stored files (data files and programs) and their length. Any user may be connected to a logical disc. In order to have access to a logical disc the user must know the name of the logical disc and this gives the user the right to read the files stored on it. If the user wishes to write on the logical disc, then in addition to the name he must give the protection key, which is a number in the interval 0 to 65535. There are a number of terminal commands connected with the use of the floppy disc.

CONNECT	: Used for connecting the user to a logical disc. After CONNECT follows the name of the logical disc and protection key, if any. E.g., CONNECT "DISC 1", 637.
COPY	: Used for copying files from one logical disc to another.
LOCK	: Used from the master terminal to bar the connection of users to the floppy disc.
LOOKUP	: Prints out information from the sub-index in the connected logical disc. This includes the printout of the names of all the files.
RELEASE	: Disconnects the terminal from the logical disc.
USERS	: Prints out the number of users connected to the floppy disc.

There are a number of RC COMAL statements connected with the use of data files by programs. These are as follows:

CREATE, DELETE, RENAME, OPEN FILE, CLOSE FILE, INPUT FILE, PRINT FILE, READ FILE, WRITE FILE, EOF (END OF FILE), MAT INPUT FILE, MAT PRINT FILE, MAT READ FILE, MAT WRITE FILE.

These statements are not explained in this publication, but further information may be obtained about them from the "RC COMAL PROGRAMMING GUIDE".

8 Summary of RC COMAL

This summary provides a brief though total picture of RC COMAL. Naturally the summary cannot contain all features, and therefore, further references should be made to the "RC COMAL PROGRAMMING GUIDE". In the summary, the following abbreviations and symbols are used:

<var>	:	The name of a numeric variable or the name of a procedure.
<svar>	:	The name of a text string (alphanumeric variable).
<expr>	:	Numeric, Boolean or alphanumeric expression.
<slit>	:	Alphanumeric constant, e.g., "PETER".
<val>	:	Numeric constant.
<lno>	:	Line numbers.
<statements>	:	One or more RC COMAL statements.
<ldnames>	:	The name of a logical disc.
<filename>	:	The name of a disc file or of a peripheral unit.
<device>	:	The name of a peripheral unit.
<file>	:	The number of a user file.
<array>	:	Indicates a dimensioned variable.
<comment>	:	Comments.
<mvar>	:	The name of a matrix.
<recl>	:	Record length.
<recno>	:	Record number.
	{	Indicates that one of the possibilities shown may be selected.
	[	Indicates that the contents may be omitted.

8.1 RC COMAL statement types

Format/Description

```
CASE <expr> OF
  [<statements-0>]
WHEN <expr> [,<expr>] ...
   <statements-1>
┌                           ┐
│     .                     │
│     .                     │
│     .                     │
│WHEN <expr> [,<expr>] ...  │
└   <statements-n>          ┘
ENDCASE [<comment>]
```

The expression following CASE is evaluated and compared with the expressions following WHEN. If there is a match in the ith WHEN statement, statements-i is executed. If no match is found, statements-0 is executed. Control is then transferred to the first statement following ENDCASE.

```
CHAIN <filename> [THEN GOTO <lineno.>]
```

Runs the SAVEd program referred to by a filename when the statement is encountered in the user's program. When used as a command, CHAIN will LOAD, but not execute, the SAVEd program.

```
     {<val> } [[,<val> ]]
DATA {<slit>} [[,<slit>]]  ...
```

Provides values to be read into variables appearing in READ or MAT READ statements.

```
DEF FN<a>(<d>) = <expr>
```

Used with the function FNa(d) to define a user function.

```
DELAY = <expr>
```

Interrupts program execution for a specified number of seconds.

```
    |<svar>(<m>)             |   [  |<svar>(<m>)             | ]
DIM {<array>(<m>)            }   [ ,{<array>(<m>)            } ]...
    |<array>(<row>,<col>)    |   [  |<array>(<row>,<col>)    | ]
```

Defines the size of string variables or numeric variable arrays.

```
END [<comment>]
```

Terminates execution of the program.

```
EXEC <name>
```

Executes a procedure defined by PROC-ENDPROC.

```
FOR <control var> = <expr1> TO <expr2> [STEP <expr3>]
  <statements>
NEXT <control var>
```

FOR begins a FOR-NEXT loop and defines the number of times a block of statements is to be executed. NEXT is the last statement in the loop and changes the value of the control variable.

```
GOSUB <lno>
  .
  .
  .
<statements>
RETURN [<comment>]
```

GOSUB transfers control to the first statement of a sub-routine. RETURN is the last statement in a subroutine and returns control to the first statement following the GOSUB statement that called the subroutine.

```
GOTO <lno>
```

Transfers control unconditionally to a statement not in normal sequential order.

```
IF <expr> [THEN] <statement>
```

Executes a single statement depending on whether an expression is true or false.

```
IF <expr> [THEN] [DO]
  <statements>
ENDIF [<comment>]
```

Executes a block of statements depending on whether an - expression is true or false.

```
IF <expr> [THEN] [DO]
  <statements-1>
ELSE [<comment>]
  <statements-2>
ENDIF [<comment>]
```

Executes statements-1 if an expression is true, otherwise statements-2.

```
                  {<var> } [               {,<var> }]
INPUT [<slit-0>,] {<svar>} [ [,<slit-n>]   {,<svar>}] ...
```

Assigns values entered from the user's terminal to numeric or string variables.

```
      {<var> }          [  {<var> }          ]
[LET] {<svar>} = <expr> [; {<svar>} = <expr> ] ...
```

Assigns the value of an expression to a variable.

```
ON ERR THEN <statement>
```

Enables the programmer to take special action, if an error occurs during program execution.

```
ON ESC THEN <statement>
```

Enables the programmer to take special action, if the ESCape key is pressed during program execution.

```
                     {GOTO }
ON <expr> [THEN]     {GOSUB} <lineno.> [,<lno>] ...
```

Transfers control to one of several lines in a program depending on the computed value of an expression when the statement is executed.

$$\left\{\begin{matrix}; \\ \text{PRINT}\end{matrix}\right\} \left[\left\{\begin{matrix}\text{<expr>} \\ \text{<slit>} \\ \text{<svar>}\end{matrix}\right\} \left[\left\{\begin{matrix}, \\ ;\end{matrix}\right\} \left\{\begin{matrix}\text{<expr>} \\ \text{<slit>} \\ \text{<svar>}\end{matrix}\right\}\right] \ldots \right] \left[\left\{\begin{matrix}, \\ ;\end{matrix}\right\}\right]$$

Prints specified items on the user's terminal.

$$\text{PRINT USING <format>,} \left\{\begin{matrix}\text{<expr>} \\ \text{<slit>} \\ \text{<svar>}\end{matrix}\right\} \left[\left\{\begin{matrix}, \\ ;\end{matrix}\right\} \left\{\begin{matrix}\text{<expr>} \\ \text{<slit>} \\ \text{<svar>}\end{matrix}\right\}\right] \ldots \left[\left\{\begin{matrix}, \\ ;\end{matrix}\right\}\right]$$

Outputs the values of items in the argument list using a specified format.

```
PROC <name>
  <statements>
ENDPROC [<comment>]
```

Defines a procedure. When the procedure is called by EXEC, control is transferred to the first statement following PROC. ENDPROC is the last statement in a procedure and returns control to the first statement following the EXEC statement that called the procedure.

RANDOMIZE

Causes the random number generator to start at a different point in the sequence of random numbers generated by the function RND(X).

$$\text{READ} \left\{\begin{matrix}\text{<var>} \\ \text{<svar>}\end{matrix}\right\} \left[\left\{\begin{matrix}\text{,<var>} \\ \text{,<svar>}\end{matrix}\right\}\right] \ldots$$

Reads in values from DATA statements and assigns the values to the variables listed in the statement.

REM [<comment>]

Inserts explanatory comments within a program.

```
REPEAT [<comment>]
  <statements>
UNTIL <expr>
```

Executes a block of statements repetitively until an expression is true. The block of statements is always executed at least once.

```
RESTORE [<lno>]
```

Resets the data element pointer to the beginning of the data list or to a particular DATA statement.

```
STOP [<comment>]
```

Terminates execution of the current program.

```
TAB(<expr>)
```

Used in PRINT statements to tabulate the printing position to the column number evaluated from an expression.

```
WHILE <expr> [THEN] DO
  <statements>
ENDWHILE [<comment>]
```

Executes a block of statements repetitively while an expression is true. If the expression is false the first time WHILE is encountered, the block of statements is not executed even once.

8.2 RC COMAL standard functions

```
ABS(<expr>)
```

Returns the absolute (positive) value of an expression.

```
ATN(<expr>)
```

Calculates the angle, in radians, whose tangent is an expression.

```
COS(<expr>)
```

Calculates the cosine of an angle which is expressed in radians.

```
EXP(<expr>)
```

Calculates the value of e (2.71828) to the power of an expression.

FN<a>(<d>)

A user function which is defined by DEF and returns a numeric value.

INT(<expr>)

Returns the value of the nearest integer not greater than an expression.

LOG(<expr>)

Calculates the natural logarithm of an expression.

RND(<expr>)

Produces a pseudo random number between 0 and 1.

SGN(<expr>)

Returns the algebraic sign of an expression.

SIN(<expr>)

Calculates the sine of an angle which is expressed in radians.

SQR(<expr>)

Computes the square root of an expression.

SYS(<expr>)

Returns system information, based on an expression which is evaluated to an integer, as follows:

0	Time of day.
1	Day.
2	Month.
3	Year.
4	Terminal port number.
5	Time used since terminal was logged on.
6	Number of file I/O statements executed.
7	Error code of last run-time error.
8	File number of last file referenced.

9	Page size.
10	Tab size.
11	Hour.
12	Minutes past last hour.
13	Seconds past last minute.
14	Constant π (3.14159)
15	Constant e (2.71828).

TAN(<expr>)

Calculates the tangent of an angle which is expressed in radians.

8.2.1 Functions for processing text strings

CHR(<expr>)

Returns the character corresponding to the number found as an expression modulo 128.

$$\text{LEN(}\begin{Bmatrix}\text{<svar>}\\\text{<slit>}\end{Bmatrix}\text{)}$$

Returns the current number of characters in a string.

$$\text{ORD(}\begin{Bmatrix}\text{<svar>}\\\text{<slit>}\end{Bmatrix}\text{)}$$

Returns the decimal number of the first character of a string.

8.3 Matrix operations in RC COMAL

MAT <mvar1> = <mvar2>

Copies the elements of one matrix to another matrix.

$$\text{MAT <mvar1> = <mvar2>}\begin{Bmatrix}+\\-\end{Bmatrix}\text{<mvar 3>}$$

Performs the scalar addition or subtraction of two matrices.

MAT <mvar1> = $\begin{Bmatrix} \text{<mvar2>} \\ \text{(<expr>)} \end{Bmatrix}$ * <mvar3>

Performs the multiplication of one matrix by another matrix or by a scalar.

<var> = DET(<expr>)

Returns the determinant of the last matrix inverted by a MAT INV statement.

MAT <mvar> = CON

Initializes a matrix such that all elements are set to one.

MAT <mvar> = IDN

Initializes a matrix such that all elements (i,i) are set to one and the remaining elements are set to zero.

MAT INPUT <mvar1> [,<mvar2>, ... ,<mvar-n>]

Assigns numeric values entered from the user's terminal to the elements of one or more matrices.

MAT <mvar1> = INV(<mvar2>)

Inverts a matrix and assigns the resultant element values to another matrix.

MAT PRINT <mvar> $\left[\begin{Bmatrix} ; \\ , \end{Bmatrix} \text{mvar>}\right]$ ... [;]

Outputs the values of the elements of one or more matrices on the user's terminal.

MAT READ <mvar> [,<mvar>] ...

Reads in numeric values from DATA statements and assigns the values to the elements of one or more matrices.

MAT <mvar1> = TRN(<mvar2>)

Transposes a matrix and assigns the resultant element values to another matrix.

MAT <mvar> = ZER

Initializes a matrix such that all elements are set to zero.

8.4 Commands to logical discs

CONNECT <ldname> [,<expr>]

Connects the user's terminal to a logical disc for reading or, if the protection key is correctly specified, for both reading and writing.

COPY "<ldname>:<filename1>","<filename2>"

Copies a file (<filename1>) from any logical disc (<ldname>) to a file (<filename2>) in the logical disc to which the terminal is connected.

INIT <device>

Initializes the main catalog in a device containing logical discs.

LOCK <device>

Locks a device, when changing discs or closing down the system, so that no user can connect his terminal to a logical disc in that device.

LOOKUP ["$LPT"]

Returns a listing of the files in the logical disc to which the terminal is connected.

RELEASE

Disconnects the user's terminal from the logical disc to which it is connected.

USERS <device>

Returns the number of users whose terminals are connected to any logical disc in a device.

8.5 Statements for file processing

CLOSE [FILE(<file>)]

Dissociates a filename and a user file number (see OPEN FILE) so that the file no longer can be referenced. The CLOSE form of the statement closes all open files.

CREATE <filename>,<size>[,<recl>]

Creates a file in the logical disc to which the user's terminal is connected.

DELETE <filename>

Deletes a file in the logical disc to which the user's terminal is connected.

EOF(<file>)

Returns a value of +1, if an end of file condition was detected in the last INPUT FILE or READ FILE statement; otherwise, a value of 0 is returned.

```
                         |<var> |[  |<var> |]
INPUT FILE(<file>) [,]   |<svar>|[, |<svar>|]  ...
```

Reads data in ASCII format from a sequential access file for the variables in the argument list.

MAT INPUT FILE(<file>) [,] <mvar> [,<mvar>] ...

Reads data in ASCII format from a sequential access file for the matrix variables in the argument list.

MAT PRINT FILE(<file>) [,] <mvar> [,<mvar>] ...

Writes matrix data in ASCII format to a sequential access file.

```
MAT READ FILE(<file>[,<recno>]) [,] <mvar> [,<mvar>] ...
```

Reads data in binary format from a sequential access file or record of a random access file for the matrix variables in the argument list.

```
MAT WRITE FILE(<file>[,<recno>]) [,] <mvar> [,<mvar>] ...
```

Writes matrix data in binary format to a sequential access file or record of a random access file.

```
OPEN FILE(<file>,<mode>) [,] <filename>
```

Associates a filename, i.e. a disc file or a device, with a user file number so that the file can be referenced in other file I/O statements; also specifies how the file is to be used.

```
                            |<expr>| [|,| |<expr>|]       [|,|]
PRINT FILE(<file>) [,]      {<slit>} [{ } {<slit>}] ...   [{ }]
                            |<svar>| [|;| |<svar>|]       [|;|]
```

Writes data in ASCII format to a sequential access file.

```
                                          |<expr>| [     |<expr>|]       [|,|]
PRINT FILE(<file>) [,] USING <format>,    {<slit>} [|,|  {<slit>}] ...   [{ }]
                                          |<svar>| [|;|  |<svar>|]       [|;|]
```

Writes data in ASCII format to a sequential access file, using a specified output format.

```
                                  |<var> | [   |<var> |]
READ FILE(<file>[,<recno>]) [,]   {      } [,  {      }] ...
                                  |<svar>| [   |<svar>|]
```

Reads data in binary format from a sequential access file or record of a random access file for the variables in the argument list.

```
RENAME <filename1>,<filename2>
```

Renames a file in the logical disc to which the user's terminal is connected.

```
                                   |<expr>| [   |<expr>| ]
WRITE FILE(<file>[,<recno>]) [,]  {<slit>} [ , {<slit>} ] ...
                                   |<svar>| [   |<svar>| ]
```

Writes data in binary format to a sequential access file or record of a random access file.

8.6 System commands in RC COMAL

```
{<lno n1>,<lno n2> }
{<lno n1>          }
{<lno n1>,         }
{,<lno n2>         }
```

Deletes one or more statements in a program.

```
      [ {<lno n1>                          } ]
      [ {|STEP|                            } ]
AUTO  [ {|,   | <lno n2>                   } ]
      [ {           |STEP|                 } ]
      [ {<lno n1>   |,   | <lno n2>        } ]
```

Provides automatic line numbers in a program, thereby making it easier to enter programs from a terminal.

```
BATCH ["$LPT"]
```

Places the terminal in batch mode and causes the system to start reading cards from the mark-sense card reader. Job output will appear on the terminal or, if the BATCH "$LPT" form of the command is used, on the line printer.

```
BYE
```

Logs the terminal off the system.

```
|CON |
|CONL|
```

Continues execution of the current program after the execution of a STOP statement in the program, after the ESCape key has been pressed, or after an error has occurred. Output from PRINT statements will appear on the terminal or, if the CONL form of the command is used, on the line printer.

ENTER <filename>

Merges the statement lines from the disc file or the device specified by a filename into the current program storage area.

```
     ┌ ┌<lno n1>                        ┐ ┐
     │ │ ┌TO┐                           │ │
LIST │ { │, │<lno n2>                  } │ [<filename>]
     │ │              ┌TO┐              │ │
     └ └<lno n1>      │, │<lno n2>      ┘ ┘
```

Outputs part or all of the currently loaded program in ASCII format to the disc file or the device specified by a filename or, if no filename is specified, to the terminal.

LOAD <filename>

Loads a previously SAVEd program in binary format from the disc file or the device specified by a filename into the user's program storage area.

NEW

Clears all currently stored program statements and variables from core memory and closes any open files.

PAGE=<expr>

Sets the right-hand margin of the terminal.

```
       ┌ ┌<lno n1>                        ┐ ┐
       │ │ ┌TO┐                           │ │
PUNCH  │ { │, │<lno n2>                  } │
       │ │              ┌TO┐              │ │
       └ └<lno n1>      │, │<lno n2>      ┘ ┘
```

Outputs part or all of the currently loaded program in ASCII format to the terminal punch (when present).

```
         ┌┌ <lno n1>                        ┐┐
         ││ │STEP│                          ││
RENUMBER ││ │,   │<lno n2>                  ││
         ││                                 ││
         ││               │STEP│            ││
         └└ <lno n1>      │,   │<lno n2>    ┘┘
```

Renumbers the statements in the current program.

```
│RUN │  ┌│<lno>      │┐
│RUNL│  └│<filename> │┘
```

Executes the current program, either from the lowest numbered statement or from a specified line number, or loads and executes a previously SAVEd program as the current program. Output from PRINT statements will appear on the terminal or, if the RUNL form of the command is used, on the line printer.

SAVE <filename>

Writes the currently loaded program, including the current values of all variables and parameters, in binary format to the disc file or the device specified by a filename.

SIZE

Returns the number of bytes used by the current program and the numbers of bytes left.

TAB=<expr>

Sets the zone spacing between the print elements output by PRINT statements.

8.7 Commands in connection with batch runs

EOJ

Terminates a job.

SCRATCH [<text>]

Initiates a job.

TIME=<val>

Specifies how may seconds a job may run. If nothing is specified, TIME = 60 seconds.

```
          ┌┌ <lno n1>                          ┐┐
          ││ |STEP|                             ││
RENUMBER  ││ |,   |<lno n2>                     ││
          ││                                   ││
          ││               |STEP|              ││
          └└ <lno n1>      |,   |<lno n2>      ┘┘
```

Renumbers the statements in the current program.

```
|RUN |  ┌|<lno>     |┐
|RUNL|  └|<filename>|┘
```

Executes the current program, either from the lowest numbered statement or from a specified line number, or loads and executes a previously SAVEd program as the current program. Output from PRINT statements will appear on the terminal or, if the RUNL form of the command is used, on the line printer.

SAVE <filename>

Writes the currently loaded program, including the current values of all variables and parameters, in binary format to the disc file or the device specified by a filename.

SIZE

Returns the number of bytes used by the current program and the numbers of bytes left.

TAB=<expr>

Sets the zone spacing between the print elements output by PRINT statements.

8.7 Commands in connection with batch runs

EOJ

Terminates a job.

SCRATCH [<text>]

Initiates a job.

TIME=<val>

Specifies how may seconds a job may run. If nothing is specified, TIME = 60 seconds.