AAEC/E221

AAEC/E221

AUSTRALIAN ATOMIC ENERGY COMMISSION

RESEARCH ESTABLISHMENT
LUCAS HEIGHTS

ACL-NOVA: A MULTI-USER CONVERSATIONAL INTERPRETER FOR THE NOVA COMPUTER

by

P.L. SANGER

April 1971

Re-issued July 1972

ISBN 0 642 99472 2

AUSTRALIAN ATOMIC ENERGY COMMISSION

RESEARCH ESTABLISHMENT

LUCAS HEIGHTS

ACL-NOVA : A MULTI-USER CONVERSATIONAL INTERPRETER FOR THE NOVA COMPUTER

by

P. L. Sanger

ABSTRACT

A multi-user conversational interpreter for the NOVA computer is described. A new conversational language ACL, on which the interpreter is based, is also described. The interpreter uses the first 6¾K of a 12K NOVA computer and provides a very flexible arrangement for supporting a number of teletype terminals.

The ACL-NOVA system provides a powerful conversational computing facility for many users in a time sharing environment.

National Library of Australia card number and ISBN 0 642 99472 2

The following descriptors have been selected from the INIS Thesaurus to describe the subject content of this report for information retrieval purposes. For further details please refer to IAEA–INIS–12 (INIS: Manual for Indexing) and IAEA–INIS–13 (INIS: Thesaurus) published in Vienna by the International Atomic Energy Agency.

COMPUTERS; PROGRAMMING; PROGRAMMING LANGUAGES

CONTENTS

		Page
1.	INTRODUCTION	1
2.	THE INTERPRETER PROGRAM	1
	2.1 General Discussion	1
	2.2 Arithmetic	2
	2.3 Variables	2
	2.4 Arithmetic Statements and Expressions	2
	2.5 Sequence Numbers and Statement Numbers	4
3.	USING THE ACL-NOVA SYSTEM	4
	3.1 Hardware Considerations	4
	3.2 Initialising the ACL-NOVA System	5
	3.3 Initialising a Terminal	5
	3.4 Operational State of a Terminal	6
	3.5 Input from a Terminal	7
	3.6 Error Correction	9
	3.7 Input from Paper Tape	9
	3.8 Allocation of Space Within a Work Area	10
	3.9 Expansion of a User's Work Area	10
4.	IMMEDIATE STATEMENTS	11
	4.1 RUN Statement	11
	4.2 GO TO Statement	12
	4.3 LIST Statement	12
	4.4 SYMBOLS Statement	13
	4.5 CLEAR Statement	13
	4.6 SPACE Statement	13
	4.7 FTRON and FTROFF Statements	13
	4.8 TRON and TROFF Statements	14
	4.9 TYPE Statement	14
	4.10 PA and PB Statements	16
	4.11 END Statement	16
	4.12 SUSPEND Statement	17
	4.13 EDIT Statement	17
	4.14 System Messages	18
5.	STORED STATEMENTS	18
	5.1 ACCEPT Statement	19
	5.2 CALL Statement	20
	5.3 RETURN Statement	20

CONTENTS (Continued)

		Page
5.4	CONTINUE Statement	20
5.5	STOP Statement	20
5.6	IF Statement	20
5.7	PAUSE Statement	21
6.	ERROR MESSAGES	21
7.	CONCLUSIONS	21
8.	ACKNOWLEDGEMENTS	22
9.	REFERENCES	22

TABLE 1: LIST OF ACL STATEMENTS

TABLE 2: SAMPLE PROGRAM USING THE ACL LANGUAGE

APPENDIX 1: LOADING AND RESTARTING THE ACL-NOVA SYSTEM

1. INTRODUCTION

The first conversational computing facility used at the A.A.E.C. Research Establishment was a one-terminal interpretive system called ACTIV-8 (N. W. Bennett, unpublished report, 1968) developed for a PDP8 computer. This system was designed to give individual scientists and engineers an easy, direct way of solving small numerical problems.

To expand this facility a multi-user conversational interpreter was written for a NOVA computer. This interpreter was based on a new conversational language ACL (Bennett and Sanger 1972) which is described in this report. The first 6¾K of a 12K NOVA computer contains the interpreter, called ACL-NOVA, and the system currently supports five teletypewriter terminals and one graphical display terminal.

2. THE INTERPRETER PROGRAM

2.1 General Discussion

In the ACL-NOVA system, statements can be entered at any of the teletypewriter terminals and either executed immediately or stored for later execution. An 'echo-checking' mechanism is used to ensure that only valid statements are entered.

The interpreter has two main parts. One part, the Interrupt Handler, syntax checks the input from the terminals and stores it essentially as a string of source characters in a buffer area located in the appropriate user work area. This part of the program also handles any output that must be sent to the terminals.

The second part, the Background Program, controls statement execution. Requests for statement execution from each terminal are treated at the same priority level. Each terminal is serviced in a round-robin approach with the computer processing one statement at a time for each user. Thus the Background Program; (i) executes an immediate statement by interpreting the source string stored in the user buffer area and performing the indicated operations, (ii) stores a statement for later execution by moving the character string from the buffer area to another part of the user work area, and (iii) executes a stored program by fetching one statement at a time from the user work area and moving this into the buffer area for processing as in (i).

The interpreter spends most of its time in the Background Program executing statements or checking whether there is a statement to be executed. However the Background Program can be interrupted at any time by input or output operations at a terminal and these interrupts are serviced completely

before control is returned to the Background Program.

2.2 Arithmetic

All arithmetic is performed on 32 bit floating point numbers. These numbers have a sign bit, a 7 bit characteristic in the well-known excess 64 exponent notation, and a 24 bit fraction. This form corresponds to the short floating point number used on the IBM System/360 computer and can represent numbers in the range:

$$16^{-65} \quad \text{number} \leqslant (1-16^{-6}).16^{63} \quad ,$$

or approximately $5.4 \times 10^{-79} \leqslant \text{number} \leqslant 7.2 \times 10^{75}$.

The times taken to perform the arithmetic operations are: addition and subtraction, 0.55 msec; multiplication, 1.72 msec; division, 2.95 msec. For input, the numbers may have free format; that is, they may be integers, may contain a decimal point or may contain an exponent.

Examples are: 327, 1.231, .0014, -1.45E+12, 3E2.

2.3 Variables

Three types of variables may be used; a simple variable, a singly subscripted variable or a doubly subscripted variable.

A simple variable name must begin with an alphabetic character and may be followed by up to three alphanumeric characters. Singly subscripted variables consist of an alphabetic character, which may be followed by an alphanumeric character, followed by an arithmetic statement or expression (see Section 2.4) enclosed in brackets. The arithmetic statement or expression is evaluated and truncated to form an integer in the range 0 to 65,535, the appropriate subscript. Doubly subscripted variables consist of an alphabetic character, which may be followed by an alphanumeric character, followed by two arithmetic statements or expressions which are separated by a comma and are enclosed in brackets. Each of these arithmetic statements or expressions is evaluated and truncated to form an integer in the range 0 to 255, the appropriate subscript. Examples are:

A,A1B2,ZA,RETN,..... simple variables

AC(1),D1(I+X-3/2E1),X(I←3+N←2),... singly subscripted variables

B3(7,2),Y(I←I+1,J←J+INT(BX(A←3)←4+Z←3)),... doubly subscripted variables.

2.4 Arithmetic Statements and Expressions

The basic operands in an arithmetic statement or expression are variables and numbers. The relevant operators are +, -, *,/(divide), ↑ (power), ← (assignment) and the built-in mathematical functions ABS, ATN (arctan), COS,

DPT‡, EXP, INT, LOG, SIN and SQR.

The mathematical operators have the usual hierarchy with the order of execution as ↑ first, then * or / at the same level and finally + or - at the same level. Brackets may be used to override the mathematical hierarchy.

If a variable is followed by an assignment arrow (←), then during statement execution the expression to the right of the assignment arrow is evaluated and its value given to that variable. The variable name and its value in floating point form are stored in a symbol table in the user work area. Multiple assignments may occur in the one arithmetic statement or expression, and in this case, assuming first of all that the expression is bracket free, then the rightmost assignment arrow is located, the expression to the right of this is evaluated and the resulting value given to the variable. This process is continued until all the assignments have been carried out.

Examples: A←3; A is assigned the value 3

C←6+B←X1←2; X1 is first assigned the value 2, B is next assigned the value 2 and finally C is assigned the value 8.

In more complicated expressions that contain a number of levels of brackets plus multiple assignments, the expression is evaluated by searching first for the rightmost opening bracket. The expression enclosed between the corresponding pair of opening and closing brackets (an expression at zero bracket level) is then evaluated by searching for the rightmost assignment arrow and proceeding as described in the last paragraph. The bracket processing and assignment arrow processing is continued until the whole expression is reduced to zero bracket level and evaluated.

Example: Y←(A←2*B)↑(B←SQR(Z←9))+X←4.

Z is first assigned the value 9, B is then assigned the value 3, A is assigned the value 6, X is assigned the value 4 and Y is finally assigned the value $6^3 + 4 = 220$.

An expression is termed an arithmetic statement if the first term in the expression is a variable followed by an assignment arrow; otherwise it is an arithmetic expression.

The result of executing an arithmetic expression is printed at a terminal in one of two possible formats depending on its magnitude. The number is given in exponent form if it is in the range $|\text{number}| \leq 10^{-4}$ or $|\text{number}| \geq 10^{3}$. If the number lies in the range $10^{-4} < |\text{number}| < 10^{3}$ it is printed in non-

---

‡ This function is used in conjunction with the TYPE statement and is described in Section 4.9.

exponent form with seven significant figures, if necessary. Integers are printed without a decimal point.

Examples: (i) 3↑2+AB1←2.453 is an arithmetic expression and during execution AB1 would be assigned the value 2.453 and the result 11.453 would be printed at the terminal.

(ii) AX←SQR(C←4*B(1)←1)+3-B←-2 is an arithmetic statement and during execution B(1) would first be assigned the value 1, C would then be assigned the value 4, B would be assigned the value -2 and AX would finally be assigned the value 7, but nothing would be printed at the terminal.

(iii) (Z2←8.332055E-3) is an arithmetic expression and during execution Z2 would be assigned the value 8.332055E-3 and the result .008332056 would be printed at the terminal.

(iv) C6 is an arithmetic expression and during execution the value of the variable C6 would be printed at the terminal.

2.5 Sequence Numbers and Statement Numbers

Each stored statement must have a sequence number in the range 100 to 999. The statements are ordered according to their sequence number and consequently statements in a stored program may be typed in any order and inserted or deleted quite freely.

A one or two digit statement number in the range 0 to 99 may also be associated with a stored statement. Thus a stored statement can be referred to either by a three digit sequence number or a one or two digit statement number. It is possibly better to use statement numbers for branching within a stored program since the branch statements do not have to be altered if the sequence numbers of the stored statements are changed.

3. USING THE ACL-NOVA SYSTEM

3.1 Hardware Considerations

The terminals used in the ACL-NOVA system were five ASR model 33 teletypes with standard character sets, and one Tektronix T4002 Graphic Computer Terminal with a standard character set. The character set used for the ACL language consists of the standard alphanumeric characters plus the special characters " ' ( ) * + , - . / : ; < > ? ↑ ← ♁† DEL ESC CR‡ BEL CAN. Note that the DEL character is often referred to as RUB OUT, BEL as CNTRL/G and CAN as CNTRL/X.

---

‡ Carriage return is represented by CR or ↵ in this report.

† Space or blank is represented by ♁ in this report.

These terminals are operated on-line in full-duplex mode. This means that a character pressed at the teletype keyboard is not printed at the teletype until it is sent back or 'echoed' by the NOVA computer. By making use of this feature, only characters that are valid in syntax are 'echoed' at a terminal and this ensures that only valid ACL statements are accepted by the system. This 'echo-checking' mechanism is a valuable feature of the ACL-NOVA system.

3.2 Initialising the ACL-NOVA System

When the ACL-NOVA system is first loaded into the NOVA computer a number of parameters must be specified. These parameters are the number of simultaneous users to be supported by the system, the number of words in each unit of work area (INCR) to be allocated to a user, and the number of words of work area available to the users of the ACL-NOVA system (see Appendix 1 for more details).

Once these parameters have been specified, the SET command (see Appendix 1) is used to define the device code of terminals for which work space should be reserved, and also the amount of work space that should be reserved.

At this stage, control is passed to the resident ACL-NOVA monitor program and the ACL-NOVA message (see Section 3.3) is sent to each of the reserved terminals. Users can now begin work at the terminals.

At the Research Establishment, the ACL-NOVA system is initialised in the following way

```
ACL-NOVA PARAMETERS
NUMBER OF TERMINALS  =  6)
WORK AREA INCREMENT  =  512)
AVAILABLE USER AREA  =  5120)
SET 10,20,30,50,70)
```

indicating that there can be six simultaneous users, each increment of work area is 512 words (½K), the available user area is 5120 words (5K) and five of the six simultaneous users are to be reserved ½K each of work space.

3.3 Initialising a Terminal

To begin work at a terminal at any time, the user types the CNTRL/G combination of keys at the teletype keyboard. The three possible responses to CNTRL/G are:

(i) If this terminal was specified as a reserved terminal, then the terminal gives CR and three line feeds, prints ACL-NOVA followed by another CR and three line feeds. The work space requested in the SET statement is

initialised regardless of the previous terminal state (see Section 3.4) and the terminal is now ready to receive program statements. These statements and the procedure to be followed in entering statements are described in the following sections.

(ii) If this terminal was not specified as a reserved terminal but one INCR of work space is available, then this area is initialised and the ACL-NOVA message is printed at the terminal to indicate that it is ready to receive program statements.

(iii) If this terminal was not specified as a reserved terminal and one INCR of work space is not currently available, then the CNTRL/G combination (BEL) is 'echoed' at the terminal to warn the user that he cannot use the system at this stage.

3.4 Operational State of a Terminal

Depending on the operations that are being carried out at a terminal, each terminal can be considered to be in one of the following three states:

State 1(a): Statements may be stored or immediate statements executed; no suspended program.

State 1(b): Statements may be stored or immediate statements executed; suspended program.

State 2: Execution of a stored program.

State 3: Suspended program state as the result of executing an ACCEPT statement.

When a work area is initialised by pressing CNTRL/G, the corresponding terminal is in state 1(a). Statements may be stored for later execution or immediate statements executed.

State 2 is entered as the result of executing a RUN statement (see Section 4.1) or an immediate GO TO statement (see Section 4.2). The stored program is executed one statement at a time as part of the Background Program.

The terminal may go from state 2 to state 1(b) as the result of executing a PAUSE statement (see Section 5.7), by taking note of certain PAUSE BEFORE or PAUSE AFTER conditions (see Section 4.10), by the character ? being typed at the terminal or as the result of an error condition in the stored program. Stored statements may now be inserted, modified or deleted, or immediate statements executed. The terminal returns to state 2 if the first input character on a line is carriage return and execution continues from the point where the program was suspended. Execution of the stored program may recommence at a different point by executing an immediate GO TO statement or it may be restarted by executing a RUN statement.

State 3 is entered as the result of executing an ACCEPT statement (see Section 5.1). In this state an arithmetic statement or expression can be entered to indicate the value of a variable, and when this is done the terminal returns to state 2.

The terminal may go from state 3 to state 1(b) by typing the character ? at the terminal. An initial carriage return will now cause execution of the ACCEPT statement to be restarted. Execution may recommence at some other point by use of the immediate GO TO statement or be restarted by executing the RUN statement.

An immediate STOP statement can be used to go from state 1(b) to state 1(a), while the stored STOP statement (see Section 5.5) causes the terminal to go from state 2 to state 1(a).

An END statement (see Section 4.11) is used to indicate that a user has completed work at a terminal. In the case where the terminal was not specified as a reserved terminal, this causes the user's work space to become available to other users. Execution of the END statement on a reserved terminal restores the work space for this terminal to the amount specified in the SET statement and makes any additional space (see Section 3.9) available to other users.

3.5 Input from a Terminal

Input begins from column 1, which is taken to be the leftmost position of the teletype carriage that results from a CR, and may consist of up to 72 characters. If input is continued past column 72, it is cancelled by a line of minus signs and must be entered again.

The discussion of keyboard input which follows is based on the possibilities that may occur at certain column positions. It is presented here to show how the syntax of the keyboard input defines the statement type or operation to be carried out.

Column 1(a): If the first character is the letter C followed by a blank, then the characters that follow are taken to be a comment field and are 'echoed' without syntax checking.

Column 1(b): If the first three characters are numbers in the range 100-999 followed by a blank, then this three digit number is taken to be the sequence number of a stored statement.

Column 1(c): If the first character is a blank, this indicates that a sequence number is to be generated automatically for the user and this is typed in columns 1, 2, 3 followed by a space in column 4. The automatically generated sequence number is ten greater than the sequence number last entered.

The sequence number 110 is given if no previous sequence number has been entered.

Column 1(d): If the first character is CR in state 1(a), then carriage return, line feed is echoed at the terminal. In state 1(b), this causes the terminal to return to state 2 and continue execution from the point where the stored program was suspended.

Column 1(e): If the characters do not fall into the above categories, they are syntax checked as though they are part of an immediate statement. On receipt of a 'valid' CR, this immediate statement is executed.

Column 5(a): If the carriage is positioned at column 5 after column 1(b) or 1(c) and the next character is CR, then the stored statement with the sequence number given in columns 1, 2, 3 is deleted.

Column 5(b): If the carriage is positioned at column 5 after column 1(b) or 1(c), then a one or two digit statement number in the range 0-99 may be entered at the keyboard. When a two digit statement number is supplied, the carriage is automatically positioned to column 8. If a one digit statement number is given, this should be followed by a blank and the carriage is then automatically positioned to column 8. If no statement number is required, a blank must be given in column 5 and the carriage is then automatically positioned to column 8.

Column 8(a): If the carriage is positioned at column 8 after column 5(b) and the next character is CR, then the statement number given in columns 5 and 6 is assigned to the stored statement with the sequence number given in columns 1, 2, 3. When no statement number appears in columns 5 and 6, this means that no statement number would now be assigned to the relevant stored statement.

Column 8(b): If the carriage is positioned at column 8 after column 5(b) and the next two characters are C followed by a blank, then any characters that follow are taken to be part of a stored comment and they are echoed without being syntax checked. In state 2 a stored comment is treated as a CONTINUE statement (see Section 5.4). When a request is made to LIST the stored program (see Section 4.3), the comment is typed at the terminal.

Column 8(c): If the carriage is positioned at column 8 after column 5(b) and neither of column 8(a) or 8(b) applies, then the input is syntax checked to allow any statement that may be used as a stored statement (see Section 5). When this statement is terminated by a 'valid' CR, it is saved in the user work area.

Examples:

(i) 6*3-4↵ is an immediate arithmetic expression that would be evaluated immediately and the result printed at the terminal.

(ii) A1←6↵ is an immediate arithmetic statement that would be processed immediately and results in A1 being assigned the value 6.

(iii) 108␣␣␣␣A1←6↵ is an arithmetic statement to be saved. The sequence number 108 is associated with this statement. (␣ indicates that a blank is generated automatically.)

(iv) 614␣72␣D213←B+C←9↵ is an arithmetic statement to be saved. The sequence number 614 and the statement number 72 are associated with this statement.

(v) 108␣↵ would cause the stored statement with the sequence number 108 to be deleted.

(vi) 614␣3␣␣↵ would now cause the statement number 3 to be associated with the stored statement with the sequence number 614.

(vii) 614␣␣␣␣↵ would now cause any statement number associated with the stored statement with the sequence number 614 to be deleted.

3.6 Error Correction

Errors which occur while a statement is being typed may be corrected quite simply. For example, to delete the last two characters that were accepted as input, type <2↵. This would cause the original line of input minus the last two characters to be typed on a new line and to be syntax checked as though they were the original keyboard input. Thus.

A2←B+SQR(A£1+C←3)-X3(4,2)-6<2↵

would result in the new line

A2←B+SQR(A£1+C←3)-X3(4,2)

being typed. A one or two digit number may follow the < symbol.

A statement being entered at the keyboard may also be corrected in edit mode (see Section 4.13) by typing <<↵ after the last input character.

Finally, if the whole input line is to be deleted, type <<< after the last input character.

3.7 Input from Paper Tape

Standard programs should be held on paper tape and may be entered from the paper tape reader once a user has commenced work at a terminal. The value of variables to satisfy a series of ACCEPT statements in a program may also be read from paper tape.

3.8 Allocation of Space Within a Work Area

The first 135 words of each user's work area is used for various pointers and buffers for the ACL-NOVA system. The remaining area is used for storing statements and for the symbol table. To provide the most efficient use of core storage, statements are stored starting from the end of the buffer areas and continuing towards the end of the work area, while symbol table entries are stored starting at the end of the work area and continuing backwards towards the start of the work area. In this way a program with many statements but few variables, or a program with few statements but many variables can be handled.

Each statement to be stored is packed two characters per NOVA word and the total number of words for a given statement is defined by the following formula:

$$W = (N-P+S+C)/2$$

where W = number of words required (including end of statement indicators to make the statement finish on a word boundary).

N = number of input characters typed before CR.

P = 0, if N is even or 1, if N is odd.

S = 2, if a statement number is present; zero otherwise.

C = 2, if statement is a CALL statement (see Section 5.2) or is an IF statement that includes a CALL statement (see Section 5.6); zero otherwise.

Thus the statement 427␣␣CALL␣217 would be stored in nine words.

A symbol table entry requires four words; two words to contain the variable name and two words to contain its floating point value. The symbol table entries are not ordered and a given entry is located by a sequential search of the symbol table. The choice of this search method simplified the structure of each work area and is adequate for this application where the symbol table is quite small for most users.

3.9 Expansion of a User's Work Area

When an arithmetic statement or expression is executed by the Background Program, an initial scan through the expression counts the number, n, of assignment arrows. During execution of this expression, a maximum of 4n words may be added to the symbol table, and if this space is not available within the user's own area, the Background Program allocates the user an additional INCR of work space if this is possible. Similarly if a statement to be stored cannot fit into the user's work area, the Background Program attempts to increase this work area.

If an extra INCR of work space is available, the message WORK AREA EXPANDED is printed at the terminal (preceded by a sequence number if it was in state 2) and processing of statements continues normally.

If no extra work space is available, the message WORK AREA FULL is printed at the terminal (preceded by a sequence number if it was in state 2) and the terminal either stays in state 1(a) or 1(b) or goes from state 2 to state 1(b). The statement is not stored away or the indicated statement is not executed, whichever is appropriate. At this stage the user can determine how much room is left in his work area by executing the SPACE statement (see Section 4.6), and he then has three courses of action open to him:

(i) he can try to obtain more space from his own work area by CLEARing (see Section 4.5) unnecessary variables from the symbol table or by deleting unnecessary statements, or

(ii) he can SUSPEND his program (see Section 4.12) and terminate his activity at the terminal until some other time (remembering to execute an END statement so that his work space may become available to another user), or

(iii) he can continue trying to obtain extra work space by repeating the operation that led to the message WORK AREA FULL being printed at his terminal in the hope that another user frees some work space.

Once a user is given additional work space, he retains it until he completes his work by executing an END statement or until he re-initialises his area by pressing CNTRL/G.

4. IMMEDIATE STATEMENTS

Statements which do not have a sequence number associated with them are classed as immediate statements and are executed by the Background Program as soon as a 'valid' CR is typed. These statements are used to perform one-time or 'desk calculator' calculations, to control the execution of a stored program and to perform various editing and debugging functions. However, the insertion, modification or deletion of stored statements and the insertion or deletion of statement numbers associated with stored statements must also be classed as immediate statements. Arithmetic statements, arithmetic expressions and the STOP statement (see Section 5.5) can be executed as immediate statements in addition to those mentioned specifically below. The basic statement forms used for immediate statements are summarised in Table 1.

4.1 RUN Statement

The RUN statement is used to begin execution of a stored program starting at the statement with the lowest sequence number.

4.2 GO TO Statement

A GO TO statement takes the form

GO␣TO␣ {arith stmt or exprn}†

and is used to pass control to the stored statement with the sequence or statement number obtained by evaluating the arithmetic statement or expression. A GO TO statement may be part of a stored program, but if it is an immediate statement it can begin execution of a stored program at any statement.

Examples:

(i) GO␣TO␣211↵ begins execution of a stored program at the statement with sequence number 211.

(ii) GO␣TO␣I←2*2E+2-309↵ begins execution of a stored program at the statement with the statement number 91, and also causes the variable I to be assigned the value 91.

4.3 LIST Statement

Stored statements may be listed at a terminal by executing the LIST statement which takes the form

LIST[:][␣ arith stmt or exprn[,arith stmt or exprn]]††

A single statement, a group of statements or the entire stored program may be listed, as shown in the examples below.

If the colon is present, it indicates that the statements are to be punched onto paper tape. In this case, to give the user time to turn the punch ON, the listing is delayed until a CR is given as the first character on the next line. When the CR is entered at the keyboard, a series of null characters is sent to the terminal so that five inches of feeder holes are punched at the start of the tape. Five inches of feeder holes are also punched at the end of the listing.

Listing may be terminated at any time by entering ? from the keyboard, and if the statements are being punched out, five inches of feeder holes are punched after the last statement.

Examples:

(i) LIST↵ causes the entire stored program to be listed.

(ii) LIST␣67↵ causes the stored statement with the statement number 67 to be listed.

(iii) LIST␣117,421↵ causes all the stored statements from sequence number 117 to sequence number 421 to be listed.

---

† Curly brackets are used throughout text to indicate a field that must be specified.

†† Square brackets are used throughout text to represent optional fields.

4.4 SYMBOLS Statement

Execution of the SYMBOLS statement, which takes the form SYMBOLS [:] causes the contents of the symbol table to be printed out in the form

```
A1 ← 1.320000E-20
Z13 ← 21
A2(7) ← .01
NA(3,7) ← .0823
  :
```

If the colon is typed, the contents of the symbol table are punched onto paper tape, and again listing may be terminated at any time by typing ?.

The above form of the symbol table listing was chosen so that each line is an immediate arithmetic statement. If the output is punched onto paper tape, it can be read in to re-initialise the symbol table.

4.5 CLEAR Statement

Variables may be removed from the symbol table by using the CLEAR statement which takes the form

CLEAR[↓ variable[,variable] ...]

If the character * is used as the subscript for a singly subscripted variable or for both subscripts for a doubly subscripted variable then all of the elements of the appropriate array are cleared.

Examples:

(i) CLEAR↓B6,M(1),X(3,1)↵ causes the variables B6,M(1),X(3,1) to be removed from the symbol table.

(ii) CLEAR↓A3(*),B9(*,*)↵ causes all of the elements of the singly subscripted variable A3 and all of the elements of the doubly subscripted variable B9 to be removed from the symbol table.

(iii) CLEAR↵ causes the entire symbol table to be cleared.

4.6 SPACE Statement

The number of unused words in a work area may be printed out by using the SPACE statement.

4.7 FTRON and FTROFF Statements

The FTRON statement is used to turn a global full-trace indicator ON to allow any subsequent stored statements that are executed to be traced (Section 4.8). When a work area is first initialised, this indicator is automatically turned ON. The global full-trace indicator is turned OFF when the FTROFF statement is executed. In this case no subsequent stored statements can be traced until an FTRON statement is executed.

4.8 TRON and TROFF Statements

Statements are said to be traced if any symbol table assignments which occur during execution are typed at the terminal. These assignments are typed in the form

{sequence number}b̸{variable} ← {value},

for example, 210b̸C10 ← 3.187.

Each stored statement has a local trace bit associated with it, which is turned OFF when the statement is stored away. If this bit is turned ON for a particular stored statement then this statement is traced every time it is executed.

A global trace indicator, which is initially turned OFF, is also associated with each user's work area. If this indicator is turned ON, then <u>every</u> stored statement which is executed is listed at the terminal and traced. This gives a complete trace of the execution of a stored program.

When the global trace indicator is ON it overrides the local trace bits, while the global full-trace indicator always overrides the global trace indicator and the local trace bits.

The TRON and TROFF statements can be used to turn the global trace indicator ON or OFF, <u>or</u> to turn the local trace bit ON or OFF in a stored statement and they take the form

$$\left\{\begin{matrix}\text{TRON}\\ \text{TROFF}\end{matrix}\right\}\ [\,b\!\!\!/\ \text{arith stmt or exprn}]$$

The arithmetic statement or expression, if present, is evaluated to indicate the sequence or statement number of a stored statement to be traced.

<u>Examples</u>:

(i) TRON⤶ turns the global trace indicator ON.

(ii) TRON b̸ 27-1*3⤶ turns the local trace bit ON in the stored statement having the statement number 24.

4.9 TYPE Statement

Values of variables or literal data can be printed at a terminal by using the TYPE statement. It can be executed immediately or as part of a stored program, and takes the general form

$$\text{TYPE}\left[\begin{matrix} b\!\!\!/\left\{\left[\begin{matrix}:\\;\end{matrix}\left[\begin{matrix}:\\;\end{matrix}\right]\ldots\right]\text{operand}\left[\left\{\begin{matrix}\begin{matrix}:\\;\end{matrix}\left[\begin{matrix}:\\;\end{matrix}\right]\ldots ,\\ \begin{matrix}:\\;\end{matrix}\left[\begin{matrix}:\\;\end{matrix}\right]\ldots\end{matrix}\right\}\text{operand}\right]\ldots\right\} \\ \begin{matrix}:\\;\end{matrix}\left[\begin{matrix}:\\;\end{matrix}\right]\ldots\end{matrix}\right]$$

where the operands takes the form

$$[\ <\text{arith stmt or exprn}>,]\ \left\{\begin{array}{c}\text{'characters'}\\ [\text{"}]\ \text{arith stmt or exprn}\end{array}\right\}.$$

Literal data should be enclosed within apostrophes, and if an apostrophe is to be printed, then a second apostrophe should follow the one required. For example, TYPE↓' X SHOULDN''T' BE ZERO' causes the message X SHOULDN'T BE ZERO to be printed.

The value of variables may be typed out in a format determined by its magnitude, as discussed in Section 2.4, or entirely in exponent form. For example, TYPE↓A3 causes the value of A3 to be typed in a form determined by its magnitude, while TYPE↓"A3 causes the value of A3 to be typed in exponent form, regardless of its magnitude.

When variables are separated by commas, they are printed on the same line with a space between each number. Thus if A1 has the value 271 and A(2,1) has the value 6731 in all of the examples that follow, then TYPE↓A1,A(2,1) causes the line 271↓6.731000E+03 to be printed.

To continue output on a new line, a semi-colon is used as a delimiter, so that TYPE↓A1;A(2,1) causes the lines

```
271
6.731000E+03
```

to be printed. The semi-colon delimiter can also be used to space a number of lines, and TYPE↓;;; causes the teletypewriter to be spaced three lines, although one line could be spaced by using TYPE.

Output may be placed at a particular carriage position by indicating the required position by the value of an arithmetic statement or expression enclosed between the <and> characters. This positional parameter must appear before the required variable or literal data and be separated from it by a comma. For example, TYPE↓<30>,"A1,<27>,'A1=' causes the line

```
col. 1                              col. 27 col. 30
  |                                      |A1=|2.710000E+02
```

to be printed.

After printing, the carriage can be left positioned at column 1 by using the delimiter colon. This allows output to be overprinted by successive TYPE statements; thus TYPE↓<30>,"A1: followed by TYPE↓<27>,'A1=' would also result in the above line being printed.

To print numerical output so that the decimal points of numbers from successive TYPE statements line up, the DPT function can be used in conjunction

with the above positional parameters. The argument of the DPT function is evaluated and the resulting value converted into a form ready to be printed. The function then takes a value equal to the position of the decimal point relative to the start of the number. For example, DPT(A1) has the value 4, DPT("-A1) has the value 3 and DPT(A(2,1)) has the value 2. When this function is used with the positional parameter, numerical output can be lined up in the columns required. For example, the statement TYPE␣<10-DPT(A1)>,A1 followed by TYPE␣<10-DPT("-A1)>, "-A1 followed by TYPE␣<10-DPT(A(2,1))>,A(2,1) cause the lines

to be printed.

4.10 PA and PB Statements

Two PAUSE statements, PAUSE BEFORE (PB) and PAUSE AFTER (PA), can be executed as immediate statements and these affect the subsequent execution of a stored program.

Each stored statement has two PAUSE bits associated with it, a PAUSE BEFORE bit and a PAUSE AFTER bit. Before each stored statement is executed a check is made to see whether the PAUSE BEFORE bit is ON. If it is, the terminal enters state 1(b) before the statement is executed and the message {sequence number}␣PB is printed. In the same way, if the PAUSE AFTER bit is ON, the terminal enters state 1(b) after the stored statement is executed and the message {sequence number}␣PA is printed.

When the stored statements are saved, the PAUSE bits are turned OFF, but they may be turned ON by executing the statements

$$\left\{\begin{matrix}\text{PA}\\ \text{PB}\end{matrix}\right\}\ \text{␣}\{\text{arith stmt or exprn}\}$$

where the arithmetic statement or expression specifies the appropriate sequence or statement number. Once the PA or PB condition has been noted during stored program execution, the appropriate PAUSE bit is turned off.

These statements allow a user to take checkpoints when a stored program is being tested and when combined with the TRON and TROFF statements provide a powerful aid to program debugging.

4.11 END Statement

The END statement is used to indicate that a user has completed his

calculations, and it may be executed as an immediate statement or as part of a stored program.

In the case where the terminal was not specified as a reserved terminal, this causes the user's work space to become available to other users. If the terminal was a reserved terminal, then the user's work space is restored to the size specified when the system was initialised with any additional space becoming available to other users and the ACL-NOVA message is again printed.

The END statement should always be the last statement executed when a user has completed his work at a terminal.

4.12 SUSPEND Statement

If a user is executing a stored program but finds that he must cease work at a terminal, then he can continue execution of his program at some other time by making use of the SUSPEND statement.

The terminal should first be put into state 1(b) by typing the character ?, and the LIST statement executed to obtain a listing of the program if this is required.

Execution of the SUSPEND statement then causes the contents of the symbol table to be printed out, in the same format as that produced by the SYMBOLS statement, followed by an immediate GO TO statement. When the program followed by the SUSPENDed output is read in at a terminal, preferably from paper tape, at some later time, then it continues executing at the point from which it was suspended.

The statement takes the form SUSPEND[:]. If the colon is present, it indicates that the output generated by the SUSPEND statement is to be punched onto paper tape (see Section 4.3). The output, however, can be terminated at any time by entering ? from the keyboard.

Once execution of the SUSPEND statement has completed (and the paper tape punch turned OFF if it was used), the user should complete his work at the terminal by executing an immediate END statement.

4.13 EDIT Statement

The EDIT statement is used to modify statements which are part of a stored program in what is termed 'edit mode'. The statement takes the form

EDIT␣{arith stmt or exprn}

and when executed the stored statement corresponding to the sequence or statement number specified by the arithmetic statement or expression is first printed and the carriage returned to the next line at column 1. At this point the statement is ready to be edited, and to follow the 'edit mode' procedure consider a pointer to each character in the original statement, the OS pointer,

where initially OS is one.

To copy characters from the original statement, the SPACE (␣) key is pressed once for each character and the copied character becomes virtual input from the terminal. If this character is syntactically correct, it is echoed at the terminal and the OS pointer increased by one.

To insert a new character, the required character should be typed and it is echoed if correct in syntax. To insert a space the special character ESC is typed. The OS pointer is not altered in these cases.

To jump over a character in the original statement, the DEL or RUB OUT character is typed and the OS pointer is increased by one, without motion of the carriage.

Once the OS pointer has reached the end of the original statement, further attempts to press SPACE or DEL have no effect. For example, the stored statement 211␣72␣A3 ← 6 can be modified by executing the statement EDIT␣211 or EDIT␣72. If the characters,

col. 1

|␣␣␣␣␣␣␣(2) (DEL) ␣2␣

are typed, after the statement has been listed, the resulting stored statement will be 211␣72␣A(2) ← 26.

4.14 System Messages

To allow system messages to be sent to users a special command which takes the form

⍀ ␣ [message]

can be typed at the monitor terminal†. Once this statement has been executed the message is printed after the ACL-NOVA message when a user initialises a terminal using CNTRL/G. The system message can also be sent to users currently working at terminals by pressing the combination CNTRL/X (CAN) at the keyboard of the monitor terminal. This message facility is normally used to inform users of the availability of the ACL-NOVA system.

If this command is executed on any terminal other than the monitor terminal, it is treated as an immediate comment statement.

5. STORED STATEMENTS

Each stored statement has a sequence number in the range 100 to 999 and may also have a statement number in the range 0 to 99. Stored statements are

---

†The terminal with device code $(10)_8$ is taken to be the monitor terminal.

used to build up a stored program. They are ordered according to their sequence number and may be inserted, modified, or deleted freely by the user. Stored statements may be typed in any order, and any newly entered statement will replace a previously saved statement with the same sequence number.

Stored statements are executed as part of a stored program once a RUN statement or an immediate GO TO statement has been executed and the terminal goes into state 2. If an error condition occurs during stored statement execution, then an appropriate message is printed and the terminal returns to state 1(b). This allows the user to take corrective action before continuing the execution of his stored program. The terminal can also go from state 2 to state 1(b) as the result of execution of the various PAUSE statements or if the character ? is typed at the terminal. In the latter case the message {sequence number}␣? is printed to indicate the sequence number of the next stored statement to be executed.

In addition to the statements discussed below, arithmetic statements and arithmetic expressions and the TYPE, END and GO TO statements discussed in Section 4 can form part of a stored program. The basic forms of stored statements are summarised in Table 1 while a sample stored program is listed in Table 2.

5.1 ACCEPT Statement

Data may be read by a program by using the ACCEPT statement which takes the form

ACCEPT␣{variable[,variable] ...}.

When an ACCEPT statement is executed, its sequence number is typed on a new line followed by the first variable name and an assignment arrow. Execution of the stored program is temporarily suspended, with the terminal going into state 3, until the value of the variable is specified. This procedure is repeated until all of the variables listed in the ACCEPT statement have been read. The terminal then goes into state 2 and program execution continues normally. For example, execution of the stored statement 317␣10␣ACCEPT␣B10,A(1,1),X causes the line 317␣B10 ← to be typed at the terminal. When the value of B10 has been given, the line 317␣A(1,1) ← is printed. After A(1,1) is given a value, the line 317␣X ← is printed. After reading X, the terminal returns to state 2 and the next stored statement is executed.

ACCEPT statement processing may be interrupted by typing the character ? and the terminal goes from state 3 to state 1(b). If an initial CR is used to

restart stored program execution then the ACCEPT statement is reprocessed beginning with the first operand.

If an error condition occurs as the result of processing ACCEPT statement input, then an error message is printed and the user must specify new input for the variable concerned. In the above example, if the user typed the value SQR(-3) for the variable B1O, the message SQR RANGE ERROR (see Section 6) would be printed on a new line, followed by 317␣B1O ← on the next line requesting that the value of B1O be re-specified.

5.2 CALL Statement

The CALL statement is used to pass control to a group of stored statements which forms a subroutine. The statement takes the form

CALL␣ {arith stmt or exprn}

and control is passed to the statement with a sequence or statement number corresponding to the value of the arithmetic statement or expression.

Calls to any depth are allowed. Note that all variables are global and arguments are accessed from the symbol table.

5.3 RETURN Statement

The RETURN statement is used to pass control from a set of statements which have been used as a subroutine to the statement after the last CALL statement that was executed.

5.4 CONTINUE Statement

The CONTINUE statement causes control to be passed to the next stored statement.

5.5 STOP Statement

Execution of a STOP statement completes execution of a stored program and causes the terminal to return to state 1(a) with the message {sequence number}␣STOP being printed.

5.6 IF Statement

Conditional branching may be performed by using the IF statement which takes the form

IF ({arith stmt or exprn} { .EQ. / .NE. / .GT. / .GE. / .LT. / .LE. } {arith stmt or exprn}) ␣ { arith stmt or exprn / TYPE stmt / ACCEPT stmt / GO TO stmt / CALL stmt / RETURN stmt / CONTINUE stmt / STOP stmt }

If the logical relation is obeyed then the statement on the right is executed; otherwise control is passed to the next stored statement.

5.7 PAUSE Statement

Execution of the simple PAUSE statement causes the terminal to enter state 1(b) with the message {sequence number}ÞP being printed.

6. ERROR MESSAGES

When an error condition occurs, an appropriate message is typed at the terminal. If the terminal was in state 2 when the error occurred, it returns to state 1(b) and the error message is typed out with the appropriate sequence number. The messages are self-explanatory and are as follows:

WORK AREA EXPANDED
WORK AREA FULL
NN MULTIPLY DEFINED
NN UNDEFINED
SSS UNDEFINED
VBLE UNDEFINED
* OVERFLOW
+ OVERFLOW
/ OVERFLOW
EXP OVERFLOW
SQR RANGE ERROR
LOG RANGE ERROR
SEQ NO RANGE ERROR
SUBSCRIPT ERROR
PRINT POSN ERROR
STOP ERROR
RETURN ERROR
ZERO DIVISOR

where NN is a statement number, SSS is a sequence number and VBLE is a variable name.

7. CONCLUSIONS

The multi-user conversational interpreter described provides a very flexible arrangement for supporting a number of teletype terminals attached to a NOVA computer.

The number of terminals to be supported by the system, the size of work area increments and the available user area do not have to be assembled into the monitor program but may be specified when the system is first loaded into the computer. Work space may be reserved for each terminal or left 'floating' to provide the greatest flexibility for the system. Dynamic allocation of work space allows the 'floating' space to be assigned to users who need extra work area, and then returns this work space to the system when the user

completes work at a terminal.

The interpreter is based on a new conversational language ACL which is a higher-level language designed to be well suited to scientific problems, and it provides a variety of statements for the user. Statements can be used to perform one-time or 'desk calculator' calculations or can be saved in a work area to form a stored program.

The language includes many special features such as syntax checking of input, ways of suspending stored program execution and powerful editing and tracing statements designed to provide interaction with the programmers.

The flexibility of the IF statement and the generality of the arithmetic statements and expressions, with the freedom of multiple assignments, adds a great deal of power to the ACL language.

Consideration of the full-duplex mode of operation of teletypewriter terminals played an extremely important part in the design of the ACL language and the ACL-NOVA system.

The ACL-NOVA system thus provides a powerful conversational computing facility for many users in a time-sharing environment.

8. ACKNOWLEDGEMENTS

Work on this project was started by Mr. N.W. Bennett who was the originator and major developer of the ACL language. It was taken over by the author who was responsible for the final development of the language and for its implementation on the NOVA computer.

The author thanks Dr. D.J. Richardson and Mr. R.P. Backstrom for valuable discussions throughout this work.

9. REFERENCES

BENNETT, N.W. and SANGER, P.L. (1972) "The Development of the ACL Language and its Implementation ACL-NOVA". To be published.

TABLE 1

LIST OF ACL STATEMENTS

A. IMMEDIATE STATEMENTS

Comments

Arithmetic statements

Arithmetic expressions

RUN

GOßTOß{arith stmt or exprn}

LIST [:][ß arith stmt or exprn[,arith stmt or exprn]]

SYMBOLS[:]

CLEAR [ß variable[,variable]...]

SPACE

FTRON

FTROFF

TRON[ß arith stmt or exprn]

TROFF[ß arith stmt or exprn]

TYPE [ß { [: [:] ... ; [;] ...] operand [{ : [:] ... ; [;] ... } operand]... }]
[: [:] ... ; [;] ...]

where the operands take the form,

[⟨ arith stmt or exprn ⟩ ,] { 'characters' / ["] arith stmt or exprn }

PB ß {arith stmt or exprn}

PA ß {arith stmt or exprn}

STOP

END

SUSPEND [:]

EDIT ß {arith stmt or exprn}

System Messages

TABLE 1 (Continued)

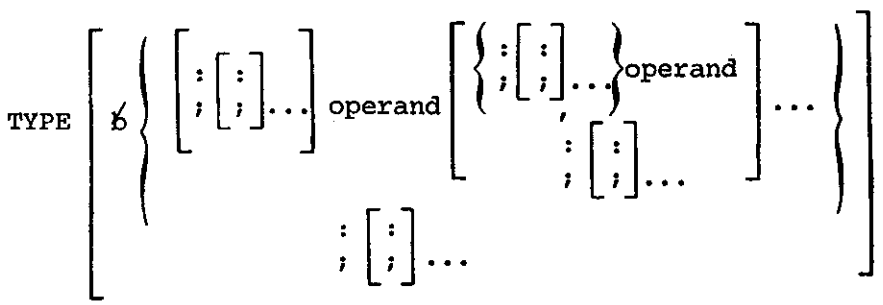
B. STORED STATEMENTS

Comments

Arithmetic statements

Arithmetic expressions

GO␣TO␣{arith stmt or exprn}



where the operands take the form

[⟨arith stmt or exprn⟩,] { 'characters' / ["]arith stmt or exprn }

ACCEPT ␣ {variable[,variable] ...}

CALL ␣ {arith stmt or exprn}

RETURN

CONTINUE

STOP

IF ({arith stmt or exprn} { .EQ. / .NE. / .GT. / .GE. / .LT. / .LE. } {arith stmt or exprn}) ␣ { arith stmt or exprn / TYPE stmt / ACCEPT stmt / GO TO stmt / CALL stmt / RETURN stmt / CONTINUE stmt / STOP stmt }

PAUSE

END

TABLE 2

SAMPLE PROGRAM USING THE ACL LANGUAGE

```
C
C     LEAST SQUARES FIT OF LINE Y=AX+B TO EXPERIMENTAL DATA
C
110   TYPE ;;;'LEAST SQUARES FIT OF LINE Y=AX+B TO EXPERIMENTAL DATA'
120   TYPE ;'SPECIFY NUMBER OF OBSERVATIONS'
130   ACCEPT N
140   TYPE 'ENTER X,Y VALUES FOR EACH OBSERVATION'
150   I←SMX2←SMX←SMXY←SMY←0
160 1  ACCEPT X(I←I+1),Y(I)
170   SMX←SMX+X(I)
180   SMX2←SMX2+X(I)*X(I)
190   SMXY←SMXY+X(I)*Y(I)
200   SMY←SMY+Y(I)
210   IF(I.LT.N) GO TO 1
220   DET←N*SMX2-SMX*SMX
230   A←(N*SMXY-SMX*SMY)/DET
240   B←(SMX2*SMY-SMX*SMXY)/DET
250   TYPE ;<5>,'A(SLOPE)=',A,<29>,'B(INTERCEPT)=',B
260   TYPE ;<10>,'X',<23>,'Y(OBS)',<34>,'Y(FIT)',<46>,'Y(OBS)-Y(FIT)'
270   J←0
280 2  C←13-DPT(X(J←J+1))
290   D←24-DPT(Y(J))
300   E←35-DPT(YF←(A*X(J)+B))
310   F←49-DPT(DF←(Y(J)-YF))
320   TYPE <C>,X(J),<D>,Y(J),<E>,YF,<F>,DF
330   IF(J.LT.N) GO TO 2
340   TYPE ;;
350   STOP
```

APPENDIX 1

LOADING AND RESTARTING THE ACL-NOVA SYSTEM

When the ACL-NOVA system is first loaded into the NOVA computer a number of parameters must be specified by the operator.

The system begins automatically with execution of the Initial Background Control Program (IBGCP) which prints the message ACL-NOVA PARAMETERS, spaces two lines and prints NUMBER OF TERMINALS = leaving the monitor teletype carriage positioned after the equals sign. The operator should type the number of simultaneous users to be supported by the system (MNUMB), followed by carriage return (↵).

A check is made to see that MNUMB $\leq$ 31 (otherwise IBGCP is restarted) and the message WORK AREA INCREMENT = is printed on the next line. The operator should specify the number of words in each unit of work area (INCR) that is to be allocated to a user, followed by carriage return.

INCR is checked to see that it is greater than 230, otherwise a new value of INCR must be specified. The value of MNUMB is then used in conjunction with a pointer to the end of the ACL-NOVA program PGEND to form a pointer to the device code list IDEVC = PGEND + MNUMB + 1 and it is also used to form a pointer to the start of the user work areas SWKAD = IDEVC + MNUMB. A check is then made to see that SWKAD + INCR * MNUMB is an address inside the core storage of the NOVA computer (otherwise IBGCP is restarted) and the message AVAILABLE USER AREA = is printed on the next line. The operator should specify the number of words of work area (WAREA) available to the users of the ACL-NOVA system, followed by carriage return.

A check is made to see that WAREA is a multiple of INCR, otherwise a new value of WAREA must be specified. Checks are also made to see that WAREA $\geq$ INCR * MNUMB and that a pointer to the end of the user work area EWKAD = SWKAD + WAREA is inside the core storage of the NOVA computer, otherwise IBGCP is restarted. This completes the first part of IBGCP.

The remaining part of IBGCP processes the SET command which allows work space to be reserved for a number of users. Certain information must be stored in the user work area during this processing and the first part of IBGCP is overwritten. IBGCP must <u>never</u> be restarted once SET command processing begins.

SET command† processing begins when the message SET is printed on a new line. If work space is not to be reserved for any terminals, then the operator should simply type carriage return. Otherwise, the operator must type a space followed by the input device codes of terminals for which space should be reserved, and also the amount of space that should be reserved. For example, if INCR = 1024 (1K) then the statement

SET 10.2, 20, 30, 50.1, 70.10 ↵

would indicate that the terminals with device codes 10_8, 20_8, 30_8, 50_8 and 70_8 were to be reserved 2K, 1K, 1K, 1K and 10K work areas respectively. If an incorrect device code is specified or the amount of space specified exceeded the available space, then SET command processing is restarted.

At this stage control is passed to the Background Control Program (BGCP) and the ACL-NOVA message is sent to each of the reserved terminals. Users can now begin work at the terminals and IBGCP is now completely overwritten.

At the AAEC Research Establishment the following responses should be given for the IBGCP parameters:

ACL-NOVA PARAMETERS

NUMBER OF TERMINALS = 6 ↵

WORK AREA INCREMENT = 512 ↵

AVAILABLE USER AREA = 5120 ↵

SET 10, 20, 30, 50, 70 ↵

RESTARTING ACL-NOVA AFTER POWER FAILURE OR COMPUTER TURNED OFF

If ACL-NOVA is to be restarted after a power failure or after the NOVA computer has been turned off, the data switches must be set to 400_8 and the program restarted manually by pressing the START switch. The Restart Background Control Program (RBGCP) begins executing and this simulates the SET command given at IBGCP time based on the parameters defined at IBGCP time. RBGCP uses the fact that the user work areas are not altered by a power failure or by turning the computer off.

---

†The SET statement takes the form

SET [↵ device[.number] [,device[.number]] ...]

where device is the input device code of a terminal in octal, and number multiplied by INCR defines the amount of work space to be reserved for that terminal. If number is not specified it is assumed to be one.

RESTARTING ACL-NOVA AFTER RUNNING PROGRAMS IN HIGH CORE

ACL-NOVA can be restarted after programs have been run in high core using the user work areas. These programs can use page zero of memory but must leave the rest of the ACL-NOVA program intact. The ACL-NOVA system is restarted in this case by loading an object tape that restores page zero of memory and contains IBGCP, and by re-specifying the ACL-NOVA parameters.