AAEC/E409

AAEC/E409

AUSTRALIAN ATOMIC ENERGY COMMISSION
RESEARCH ESTABLISHMENT
LUCAS HEIGHTS

TERMINAL FACILITIES PROVIDED BY THE DATERCOM SYSTEM

by

P.L. SANGER

April 1977

ISBN 0 642 99771 3

AUSTRALIAN ATOMIC ENERGY COMMISSION
RESEARCH ESTABLISHMENT
LUCAS HEIGHTS

TERMINAL FACILITIES PROVIDED BY THE DATERCOM SYSTEM

by

P.L. SANGER

ABSTRACT

The computer network at the Australian Atomic Energy Commission Research Establishment is currently based on an IBM360 model 65 central computer, a DEC PDP9L computer linked to the IBM360 computer *via* a selector channel, and nine minicomputer systems linked to the PDP9L computer *via* the AAEC Dataway. The Dataway Terminal Communication System, DATERCOM, developed for DGC NOVA computers makes use of the computer network facilities to allow terminals to have access either to the multi-user conversational interpreter ACL-NOVA or to the resources of the IBM360 central computer system. The features of DATERCOM in terms of the facilities made available to the terminal user are described here; they greatly expand the computing power that can be made available to AAEC scientists.

National Library of Australia card number and ISBN 0 642 99771 3

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.

DATA TRANSMISSION; DIGITAL COMPUTERS; IBM COMPUTERS; PROGRAMMING LANGUAGES

CONTENTS

CONTENTS (Continued)

1. INTRODUCTION

To cater for small-to-medium scale scientific problems, a conversational language ACL [Bennett & Sanger 1973] was developed at the AAEC Research Establishment. It was implemented as the multi-user conversational interpreter ACL-NOVA on a 12K NOVA computer supporting five teletypewriter terminals [Sanger 1971]. The performance of ACL-NOVA was later improved by the use of a directed-graph syntax analyser [Sanger & Hayes 1974], and the system was also set up to run on a second NOVA computer, an 8K NOVA 1220 computer, at the AAEC's Mascot Office.

At that stage, the AAEC computer network [Richardson 1971] was being developed and the NOVA computer was the first computer to be connected to the AAEC Dataway [Sanger, Jones & Ellis 1973] and to have access to the IBM360 computer *via* the computer network [Sanger & Backstrom 1973, Sanger 1974]. A special version of ACL-NOVA was then developed to allow ACL-NOVA to be used at five teletypewriter terminals, while a Tektronix T4002 graphical display terminal was used to interact with programs running in the IBM360 computer. The computer network communication required for the Tektronix access to the central computer used a fixed Dataway address, and conventions were established for terminal interaction with the IBM360 programs.

In March 1973, a 24K NOVA 820 computer was installed to run the ACL-NOVA system on a more dedicated basis. The five teletypewriter terminals were supported by the NOVA 820 computer, and the 12K NOVA computer supporting a teletypewriter terminal and Tektronix display terminal became fully available for system development work.

The NOVA computer was subsequently used to develop the Dataway Terminal Communication System, DATERCOM, to allow any terminal to have access to either ACL-NOVA or the resources of the IBM360 computer. This time, a range of Dataway addresses was made available for computer network communication, and new conventions were defined for the terminal interaction with programs running in the IBM360 computer. A number of new commands were added to ACL-NOVA, and access to the IBM360 computer was used to provide for the saving and loading of ACL programs and symbol tables using IBM360 disk storage. The memory of the NOVA 820 computer was later increased to 32K words, and the DATERCOM system now supports twenty terminals connected to the NOVA 820 computer in a variety of ways.

The features of the DATERCOM system in terms of facilities available to the terminal user are described in the sections that follow. Terminal access to the IBM360 computer is discussed in general terms, including a description of localised error correction facilities, since the details of the terminal interactions depend upon the particular IBM360 program invoked by the user. ACL-NOVA, on the other hand, is a complete subsystem within the DATERCOM system and, although the original version has already been extensively discussed [Sanger 1971], the features currently available to DATERCOM users will be described in detail. This will allow this document to present a complete description of the terminal facilities provided by the DATERCOM system.

2. THE DATERCOM SYSTEM

2.1 General Discussion

The DATERCOM system uses the first 8K words of the 32K NOVA 820 computer, leaving 24K words for the user work areas. To provide for the maximum flexibility, no space is reserved for any terminal and the work area is allocated as required. No response is received at a terminal until either the CNTRL/G character (BEL) or the $ character is entered at the terminal keyboard. When one of these characters has been pressed, a 138-word

control block is allocated to the terminal and either the ACL-NOVA message is printed out or the $ character is echoed.

ACL Mode When CNTRL/G is entered at the terminal, it is referred to as being in ACL mode and it allows ACL-NOVA to be used in the normal way. One increment of 512 words (½K) is allocated to a terminal when it is first initialised, in addition to the 138-word control block, and further increments of ½K are allowed up to a maximum of 4K words. Details of the current ACL-NOVA facilities are presented in Section 3.

2.2 Terminal Access to the IBM360 Computer

Non-ACL Mode A terminal is said to be in non-ACL mode when a user enters an appropriate $ command at a terminal. Entering the $ character causes a 138-word control block to be allocated to the terminal, and this is the only work space needed for interacting with programs in the IBM360 computer. Entering $LOGON for example, causes the IBM360 program, LOGON, to be executed. The interactions that can occur between the terminal user and this program are built into the LOGON program, and the DATERCOM system takes a 'passive role' by printing out anything that is sent from the IBM360 program, and by sending to the IBM360 computer any information that is entered at a terminal. Up to 132 characters of information can be sent at a time by the IBM360 program and up to 80 characters of information may be entered at a terminal and then sent to the IBM360 computer in response to a Read request.

No IBM360 Access If the IBM360 computer is not available when the $command is entered (or there is not sufficient space in the IBM360 computer to run the program specified, or the command name has been misspelt), then the message NO ACCESS is printed at the terminal; *but* in this case, the user control block is automatically returned to the system and there will be no further response at the terminal until CNTRL/G or a new $command is entered.

Using Question Mark Character to Interrupt Output The question mark character is normally used to interrupt the sending of output to a terminal, with the IBM360 program issuing some appropriate read request to determine the action to be taken by the user.

2.2.1 Local aids to non-ACL mode terminal input

Deleting Single Characters or Cancelling All Input In non-ACL mode, DATERCOM provides for input error correction by allowing the DEL character (RUBOUT) to be used to delete the last character entered (backslash is echoed), or the CAN character (CNTRL/X) to be used to cancel the whole line (exclamation mark is echoed).

Tabbing Facility A tabbing facility is also provided to assist the terminal user, and up to nine TAB settings can be set either by the user or automatically when an IBM360 program is invoked. Once these have been set, then the CNTRL/I character (TAB) can be used to space out automatically to the different tab settings. For example, assuming firstly that the terminal has been taken to its leftmost position as the result of echoing a CR character, and secondly that tabs have been set to 10 and 20, then the input characters

ABCTAB12TABXCR

will cause the characters

ABCbbbbbb12bbbbbbbbX

to be echoed and sent to the IBM360 computer (where the symbol TAB represents the CNTRL/I character, CR represents the CR character and b represents the space character).

Setting TAB Positions Users may set their own TAB positions by entering the 'at' (@) character followed by the tab settings, separated by commas, whenever input is required; the correct input response can then be entered on the next line. For example, the input characters

@10,20,30CR

would set the tab positions to 10, 20 and 30 with the CR, LF characters being echoed to make the terminal go to a new line; the IBM360 program would still be waiting for the input that can now be entered on the new line.

Paper Tape Input To cope with users wishing to enter data from paper tape, the null and LF characters are ignored as input data in non-ACL mode.

2.2.2 Dataway addresses for IBM360 communication

DATERCOM running in the 32K NOVA 820 computer uses any one of the sixteen Dataway addresses X'70' to X'7F' for loading and saving ACL information. These requests for IBM360 computer activity are of short duration; for this reason, if there are more requests than available Dataway addresses, the requests are stacked until an address does become available. Any one of the fifteen addresses X'71' to X'7F' is used for terminal communication with the IBM360 computer,and the corresponding IBM360 programs are written so that they do not depend on a particular Dataway address. Terminal interactions with the IBM360 computer can proceed for fairly long periods; for this reason,these requests are not stacked and the user is told that there is NO ACCESS if a Dataway address is not available. For the same reason, Dataway address X'70' is not made available for terminal interaction to ensure that there is at least one Dataway address available for ACL load and save requests.

In the original 12K NOVA computer, any of the four addresses X'58' to X'5B' are used for ACL loading and saving, while the addresses X'59' to X'5B' are also used for terminal interaction with the IBM360 computer.

2.2.3 Timing out non-ACL mode activity

To ensure that the Dataway addresses used for terminal communication with the IBM360 computer are used efficiently, terminals in non-ACL mode are timed-out after nine minutes of inactivity. The message TIME-OUT is printed at the terminal and the user control block is automatically returned to the system; there will be no further response at the terminal until CNTRL/G or a new $command is entered.

2.2.4 IBM360 software for terminal interaction

IBM360 programs were written to support ACL loading and saving [Backstrom 1975], and a number of programs developed to interact with DATERCOM terminals. These programs provide extensive terminal support for terminal users in the form of conversational remote job entry facilities ($LOGON), IBM360 plotter output viewing facilities ($PLOT), and terminal printer output facilities ($PRINT and $PRINTER). IBM360 Assembler routines have also been set up to provide FORTRAN access to ACL symbol tables [Backstrom 1975]. The Attention signalling capabilities of DATERCOM [Sanger 1976] have also been used in conjunction with modifications made to the HASP system [Johnstone 1974] to allow users at DATERCOM terminals to display the status of jobs running in the IBM360 computer ($#CONSOLE).

3. THE ACL-NOVA SUBSYSTEM

3.1 The ACL Interpreter

The ACL interpreter has now been set up as part of the DATERCOM system to provide users with dedicated access to ACL-NOVA. In ACL mode, ACL statements can be entered at any of the DATERCOM 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 ACL interpreter can be considered as having three main parts. One part, the Interrupt Handler, accepts input characters from a terminal, stores them in a look-ahead buffer located in the user control block, and schedules them for analysis by the second part, the Syntax Analyser. The Syntax Analyser checks the validity of the terminal input, stores it essentially as a string of source characters in a buffer area located in the appropriate user control block and schedules the Interrupt Handler to echo the appropriate output characters at the terminal. All terminal output is, in fact, scheduled through the Interrupt Handler. The third part, the Background Program, controls ACL 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 control block and performing the indicated operations, (ii) stores a statement for later execution by moving the character string from the control block to the user work area, and (iii) executes a stored program by fetching one statement at a time from the user work area and moving it into the control block for processing as in (i).

DATERCOM normally executes the Background Program processing ACL statements or checks 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; these interrupts are serviced completely and all required syntax analysis is performed before control is returned to the Background Program.

3.1.1 Arithmetic

Floating Point Numbers 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 IBM360 computer and can represent numbers in the range:

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

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

Input Values in Free Format 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

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

3.1.2 Variables

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

Simple Variables A simple variable name must begin with an alphabetic character and may be followed by up to three alphanumeric characters.

Singly Subscripted Variables Singly subscripted variables consist of an alphabetic character, which may be followed by an alphanumeric character, followed by an arithmetic expression (see Section 3.1.3) enclosed in brackets. The arithmetic expression is evaluated and when converted to integer form should specify a subscript value in the range 0 to 65 535.

Doubly Subscripted Variables Doubly subscripted variables consist of an alphabetic character, which may be followed by an alphanumeric character, followed by two arithmetic expressions which are separated by a comma and are enclosed in brackets. Each of these arithmetic expressions is evaluated and, when converted to integer form, should specify a subscript value in the range 0 to 255.

Examples

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

AC(1),D1(L+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+X←3)),... doubly subscripted variables.

3.1.3 Arithmetic expressions

Operands The basic operands in an arithmetic expression are variables and numbers. The relevant operators are +, -, *,/(divide), ↑ (power), ← (assignment) and the built-in mathematical functions ABS,SIN,COS,TAN,ASN(arcsin), ACS(arccos),ATN(arctan),EXP,LOG,INT, SQR,RND† and DPT‡ (see Table 1).

Mathematical Operators 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.

Assigning Values to Variables 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 Multiple assignments may occur in the one arithmetic expression and, in this case, assuming first of all that the expression is bracket free, the rightmost assignment arrow is located, the expression to the right of this is evaluated and the resulting value is given to the variable. This process is continued until all the assignments have been carried out.

Examples

(i) A←3; A is assigned the value 3,

(ii) 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.

Processing Expressions Containing Brackets 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

---

†The operand of the Pseudo-Random Number Generator function, RND, is restricted to being a simple variable name as discussed in Table 1 and, unlike the other functions, the value of this function is stored in the operand variable as well as being used in the expression evaluation.

‡This function is used in conjunction with the TYPE statement and is described in Section 3.3.11.

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 $6^3 + 4 = 220$.

Output Produced by Evaluating an Arithmetic Expression If the first term in an arithmetic expression is a variable followed by an assignment arrow, then no printed output is produced as the result of its execution; however, the result of executing all other forms of 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-exponent form with seven significant figures, if necessary. Integers are printed without a decimal point.

Examples

(i) `3↑2+AB1←2.453)`[†] causes AB1 to be assigned the value 2.453 and the result 11.453 is printed at the terminal.

(ii) `AX←SQR(C←4*B(1)←1)+3-B←-2)` first results in B(1) being assigned the value 1, C then being assigned the value 4, B being assigned the value -2 and, finally AX being assigned the value 7, but nothing is printed at the terminal.

(iii) `(Z2←8.332055E-3)` causes Z2 to be assigned the value 8.332055E-3 and the result .008332056 is printed at the terminal.

(iv) `C6)` results in the value of the variable C6 being printed at the terminal.

Defining Arrays One- and two-dimensioned arrays of data items can be set up by assigning values to the required singly and doubly subscripted variables. Elements of an array are thus created one at a time, often *via* the ACCEPT statement (see Section 3.4.1), and they only require symbol table space when they are defined - there is no need to reserve symbol table space for ACL arrays as there is in FORTRAN with the DIMENSION statement.

3.1.4 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; 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 or if the stored program is RENUMBERED (see Section 3.3.8).

[†] Carriage Return is represented by CR or) in this report.

3.2 Using ACL-NOVA

3.2.1 Hardware considerations

ACL Character Set The terminals used in the original ACL-NOVA system were five ASR model 33 teletypes with standard character sets. As a result, the character set used for the ACL language was based on the teletype subset of the standard ASCII character set at that time, and consists of the standard upper case alphanumeric characters plus the special characters " ' ( ) * + , - . / : ; < > ? ↑ ← b† 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. On some of the newer terminals, and even on some teletypes, some of the ASCII codes are represented differently; for example, assignment arrow (←) is replaced by the underline character (_) and upwards arrow (↑) is replaced by the circumflex character (^). The user must be aware of these differences in the so-called standard teletype subset of the ASCII character set.

Lower to Upper Case Translation In ACL mode, lower case alphabetic characters are translated to upper case alphabetic characters except

(i) between quotes in a TYPE statement (see Section 3.3.11),

(ii) for Comment Statement input (see Section 3.2.4), and

(iii) for System Message input (see Section 3.3.15).

In non-ACL mode, lower case alphabetic characters are only translated to upper case for input after the $ character; that is, for $command input, all input entered in response to IBM360 read requests is passed on untranslated to the IBM360 program invoked by the user.

Echo-checking Using Full-duplex Mode The DATERCOM 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 ACL-NOVA.

3.2.2 Entering ACL mode

A terminal is said to be in ACL mode when a user enters CNTRL/G at a terminal. In this case a 138-word control block and a 512-word work area are allocated to the user, and a message consisting of CR and three line feeds, ACL-NOVA, and then another CR and three line feeds is printed at the terminal.

3.2.3 Operational state of an ACL mode terminal

Depending on the operations that are being carried out at a terminal, each ACL mode terminal can be considered to be in one of the following 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.

---

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

Initial Terminal State When ACL mode is entered by pressing CNTRL/G, the corresponding terminal is in state 1(a). Statements may be stored for later execution or immediate statements executed.

Stored Program State State 2 is entered as the result of executing a RUN statement (see Section 3.3.1) or an immediate GO TO statement (see Section 3.3.2). The stored program is executed one statement at a time as part of the Background Program.

Interrupting Stored Program State The terminal may go from state 2 to state 1(b) as the result of executing a PAUSE statement (see Section 3.4.7), by taking note of certain Pause Before or Pause After conditions (see Section 3.3.12), by the question mark 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.

Restarting Stored Program State 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.

ACCEPT Statement Input Mode State 3 is entered as the result of executing an ACCEPT statement (see Section 3.4.1). In this state an arithmetic expression can be entered to indicate the value of a variable and, when this is done, the terminal returns to state 2.

Interrupting ACCEPT Statement Input Mode The terminal may go from state 3 to state 1(b) by typing the question mark 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.

Returning to Initial Terminal State An immediate STOP statement can be used to go from state 1(b) to state 1(a), while the stored STOP statement (see Section 3.4.5) and the stored END statement (see Section 3.3.13) causes the terminal to go from state 2 to state 1(a).

Completing Work at a Terminal An immediate END statement (see Section 3.3.13) is used to indicate that a user has completed work in ACL mode and the corresponding control block and work area are automatically returned to the system.

3.2.4 Input from ACL mode terminals

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.

Immediate Comment Statement 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 'echo' without syntax checking.

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

Automatic Generation of Sequence Numbers 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.

Immediate Carriage Return 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.

Immediate Statements 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 (see Section 3.3, Table 2A). On receipt of a 'valid' CR, this immediate statement is executed.

Deleting a Stored Statement 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.

Entering a Statement Number after a Sequence Number 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 to 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.

No Statement Number after a Sequence Number Column 5(c): If the carriage is positioned at column 5 after column 1(b) or 1(c), then if no statement number is required, a blank must be entered and the carriage is automatically positioned at column 8.

Assigning a Statement Number to a Stored Statement 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.

Deleting the Statement Number Associated with a Stored Statement Column 8(b): If the carriage is positioned at column 8 after column 5(c) and the next character is CR, then no statement number would now be assigned to the relevant stored statement.

Stored Comment Statement Column 8(c): If the carriage is positioned at column 8 after column 5(b) or 5(c) 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 3.4.4).

Stored Statements Column 8(d): If the carriage is positioned at column 8 after column 5(b) or 5(c) and neither of column 8(a), 8(b) or 8(c) applies, then the input is syntax checked to allow any statement that may be used as a stored statement (see Section 3.4, Table 2B). 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 expression that would be processed immediately and results in A1 being assigned the value 6; nothing is printed at the terminal.

(iii) 108bbbbA1←6↵ is an arithmetic expression to be saved. The sequence number 108 is associated with this statement. (b indicates that a blank is generated automatically.)

(iv) 614b72bD213←B+C←9↵ is an arithmetic expression to be saved. The sequence number 614 and the statement number 72 are associated with this statement.

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

(vi) 614b3bb↵ would now cause the statement number 3 to be associated with the stored statement with the sequence number 614.

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

3.2.5 Error correction in ACL mode

Deleting Characters 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(AS1+C←3)-X3)4,2)-6<2↵
```

would result in the new line

```
A2 ← B+SQR(AS1+C ← 3)-X3(4,2)
```

being typed. A one- or two-digit number may follow the < symbol[†].

Edit Mode A statement being entered at the keyboard may also be corrected in edit mode (see Section 3.3.14) by typing <<↵ after the last input character[†].

Cancelling All Input Finally, if the whole input line is to be deleted, type <<< after the last input character (the appropriate <n combination can also be used to cancel all the input on a line)[†].

DEL and CNTRL/X Characters To maintain some compatibility between ACL mode and non-ACL mode, the DEL character can also be used for single character deletions in ACL mode (backslash is echoed) and CNTRL/X (CAN) can also be used in ACL mode to cancel all the input on a line (exclamation mark is echoed)[‡].

3.2.6 Allocation of space within a work area

Structure of User Work Area The 512-word work area allocated to an ACL mode terminal is used for storing ACL statements and for the ACL symbol table. To provide for the most

---

[†]This form of error correction cannot be used to correct input between quotes in a TYPE statement - the DEL character can be used.

[‡]Only the DEL character can be used to correct input between quotes in a TYPE statement.

efficient use of core storage, statements are stored starting from the beginning of the work area and continuing towards the end of the work area, whereas symbol table entries are stored starting at the end of the work area and continuing backwards to 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.

Space Required for a Stored Statement The number of NOVA words required for each statement to be stored is defined by the following formula:

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

where W = number of words required (including internal 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, and
C = 2, if statement is a CALL statement (see Section 3.4.2) or an IF statement that includes a CALL statement (see Section 3.4.6); zero otherwise.

For example, the statement

```
427 9  CALL 217
```

would be stored in eight words.

Space Required for Symbol Table Entries 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.

3.2.7 Expansion of a user's work area

Automatic Work Area Expansion When an arithmetic 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 ½K words 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 - a maximum of 4K words of work space is allocated to each user. If an extra ½K of work space is available, the message AREA EXPANDED is printed at the terminal (preceded by a sequence number if it was in state 2) and processing of statements continues normally.

Work Area Full If no extra work space is available, the message 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 work area by executing the SPACE statement (see Section 3.3.7), 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 3.3.5) unnecessary variables from the symbol table or by deleting unnecessary statements (see Section 3.2.4 or Section 3.3.6) - this is, in fact, the only course open to him if he has reached the 4K limit;

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

(iii) he can terminate his activity until some other time (remembering to execute an immediate END statement so that his work space may become available to another user).

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

Segmenting ACL Programs and Symbol Tables With the DATERCOM system now making it possible to save and load ACL programs and/or symbol tables using IBM360 disk storage (see Section 3.6), it is more feasible to segment ACL programs into smaller functional units and also to form data libraries that may be processed by a number of different ACL programs.

3.3 Immediate Statements in ACL Mode

Statements which do not have a sequence number associated with them are classed as immediate statements and executed by the Background Program as soon as a 'valid' CR is typed. These statements are used to perform single expression evaluation, to control the execution of a stored program or 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 expressions, the STOP statement (see Section 3.4.5), and the #LOAD and #SAVE statements (see Section 3.6) 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 2A.

3.3.1 RUN statement

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

3.3.2 GO TO statement

A GO TO statement takes the form

GObTOb{arith exprn}†

and is used to pass control to the stored statement with the sequence or statement number obtained by evaluating the arithmetic 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.

---

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

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.

3.3.3 LIST statement

Stored statements may be listed (printed out) at a terminal (in sequence number order) by executing the LIST statement which takes the form

LIST[:][barith exprn[,arith 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 the question mark character 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.

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

---

††Square brackets are used throughout this report to represent optional fields.

3.3.5 CLEAR statement

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

```
CLEAR[bvariable[,variable] ... ]
```

Clearing Arrays 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.

3.3.6 DELETE statement

The DELETE statement takes the form

```
DELETE[barith exprn[,arith exprn]]
```

and allows the whole stored program, an individual statement or a range of stored statements to be deleted. This statement is used in conjunction with the #SAVES statement (see Section 3.6.1) to allow data libraries to be saved with no stored program - the ACL program used to create the symbol table data library can simply be DELETEd just before saving the results on IBM360 disk storage.

Examples

(i) DELETE 130,170↵ deletes all stored statements with sequence numbers in the range 130 to 170.

(ii) DELETE 210↵ deletes the stored statement with the sequence number 210 - of course, 210b↵ will do the same thing (see Section 3.2.4).

(iii) DELETE↵ deletes the whole stored program

3.3.7 SPACE statement

The number of words that are still free in the user's work area out of the number of words allocated to the user may be printed out by executing the SPACE statement. Executing this statement immediately after entering CNTRL/G will result in the values 508/512 being printed out (four words in the user area are used by the system for a dummy symbol table entry). The SPACE statement can be used to indicate the size of a stored program and/or symbol table library and also to indicate the space saved by deleting stored statements or CLEARing symbol table entries.

3.3.8 RENUMBER statement

Once a stored program has been developed and thoroughly tested, it is often convenient to renumber the sequence numbers uniformly to allow for future changes. This can be done by executing the RENUMBER statement that takes the form,

```
RENUMBER[barith exprn[,arith exprn]]
```

Default Starting Value and Step Size If there are no operands, the whole stored program is renumbered starting with the sequence number 110 and increasing the sequence number in steps of ten.

Specifying Starting Value If only one operand is present, it is used as the starting sequence number and the program is renumbered using steps of ten.

Specifying Starting Value and Step Size Both the starting value and the step size may be specified by supplying two operands.

Range Checking If a sequence number greater than or equal to 1000 would be generated by renumbering a stored program, then the message 'RANGE ERROR' is printed out *leaving the original program unaltered.*

WARNING

Before using the RENUMBER statement, the user should check that any GO TO or CALL statements (including those that follow an IF statement) that occur in the stored program make use of statement numbering rather than sequence numbering since use of the RENUMBER statement *will not* alter the GO TO statements in the user's program.

The RENUMBER statement has proved useful in renumbering ACL subroutine packages so that these can be loaded from IBM360 disk storage without overwriting the user's main stored program.

3.3.9 TRON and TROFF statements

Special tracing facilities are built into the ACL language to assist the user in debugging a stored program. The result of tracing a stored statement is that any symbol table assignments that occur while the statement is being executed are printed at the terminal in the form

{sequence number}b{variable}←{value}

or, for example,

215 A72←6.13

Tracing Individual Statements Individual statements may be traced by executing an immediate statement of the form

TRONb{arith exprn}

before beginning the execution of a stored program. For example, if the immediate statements

TRON 190)

TRON 220)

TRON 230)

were executed, followed by the execution of the RUN statement (see Section 3.3.1), then any symbol table assignments that occurred as a result of executing statements 190,220 and 230 would be printed out every time they were executed. When the user has finished tracing individual statements, the trace should be turned off by using statements of the form

TROFFb{arith exprn}

or, in the above example,

TROFF 190)

TROFF 220)

TROFF 230)

Complete Program Trace If a user is really having trouble with a stored program, then a complete trace of program execution can be obtained by executing the immediate statement

TRON)

before executing the RUN statement. Every stored statement that is subsequently executed is listed at the terminal and traced. When the full trace is finished the immediate statement

TROFF)

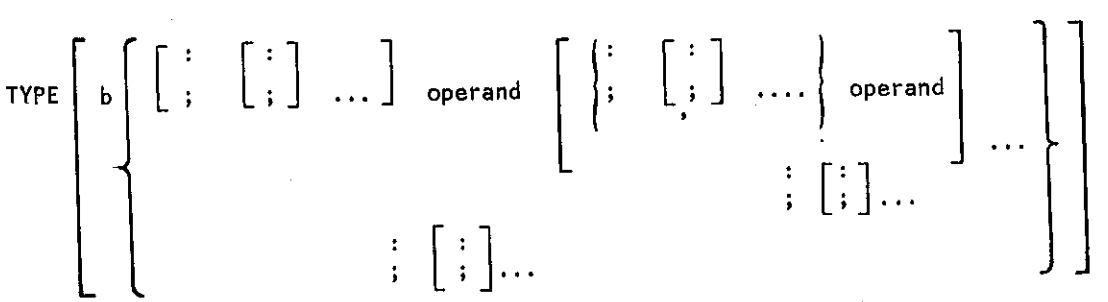
should be executed.

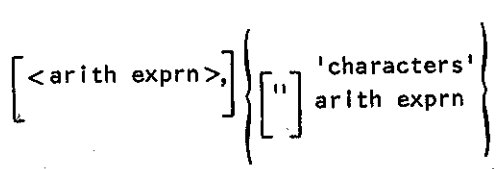
3.3.10 FTRON and FTROFF statements

Stored statement tracing can be controlled still further by the use of FTRON and FTROFF statements. Execution of these statements indicate whether any tracing is to be allowed - the initial control block state allows tracing to occur (the FTRON state) - so that the FTROFF statement can be used to temporarily override the effects of any TRON statements that may have been executed (subsequent execution of the FTRON statement would return the user to the former state).

3.3.11 TYPE statement

The results of calculations 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





Printing the Value of Expressions The value of an expression may be typed out in a format determined by its magnitude, as discussed in Section 3.1.3, or entirely in exponent format. For example

```
TYPE A3)
```

causes the value of A3 to be typed in a form determined by its magnitude, whereas TYPE "A3) causes the value of A3 to be typed in exponent form, regardless of its magnitude.

When expressions are separated by commas, they are printed on the same line with a space between each value. Thus, if A1 has the value 271 and A(2,1) has the value 6731 in all of the following examples, then

```
TYPE A1,A(2,1))
```

causes the line

```
271 6.73000E+03
```

to be printed.

Printing Headings or Text Strings Literal data (character output, headings or text strings) can be printed by enclosing them within apostrophes. For example, the statement

```
TYPE 'ANSWER=',A1)
```

causes the line

```
ANSWER=271
```

to be printed. If an apostrophe is to be printed, then a second apostrophe should follow the one required; for example

```
TYPE 'LOAN AMOUNT SHOULDN'T BE NEGATIVE')
```

causes the line

```
LOAN AMOUNT SHOULDN'T BE NEGATIVE
```

to be printed.

Multiple Line Output To continue output on a new line, a semi-colon should be used as the 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 terminal to space 3 lines; although one line could be spaced by using

```
TYPE)
```

Positioning Output on a Line Output may be placed at a particular carriage position by indicating the required position by means of an arithmetic expression enclosed between the symbols less than (<) and greater than (>). This positional parameter specifies the number of leading spaces required and it must appear before the required variable or literal data and be followed by a comma. For example,

```
TYPE <30>,"A1,<27>,'A1='
```

causes the line

```
Col.1                Col.27
  |                    |A1=2.710000E+02
  |                    |
```

to be printed.

Overprinting 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 execution of the stored statements

```
812  TYPE <30>,"A1:
814  TYPE <27>,'A1='
```

would also result in the line

```
Col.1                Col.27
  |                    |A1=2.710000E+02
  |                    |
```

being printed.

Lining Up Numerical Output (DPT Function) 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, execution of the stored statements

```
600  TYPE <10-DPT(A1)>,A1

610  TYPE <10-DPT("-A1)>,"-A1

620  TYPE <10-DPT(A(2,1))>,A(2,1)
```

cause the lines

```
Col.1              Col.9
  |                  |
  |                271
  |                 -2.710000E+02
  |                  6.731000E+03
```

to be printed with the decimal point being positioned at column ten.

3.3.12 PA and PB statements

To allow a user to take checkpoints when a stored program is being tested, two Pause statements, Pause Before (PB) and Pause After (PA), can be executed as immediate statements. Execution of these statements allows for a once-only interruption of the subsequent execution of nominated stored statements (of course, the same Pause conditions can be reset when the Pause interruption has occurred).

Pause Switches Associated with Stored Statements Each stored statement has two Pause switches associated with it, a Pause Before switch and a Pause After switch. Before each stored statement is executed, a check is made to see whether the Pause Before switch is ON. If it is, the terminal enters state 1(b) *before* the stored statement is executed and the message

{seq.no.} PB

is printed out.
In the same way, if the Pause After switch is ON, the terminal enters state 1(b) *after* the stored statement is executed and the message

{seq.no.} PA

is printed. Both PA and PB may therefore refer to the same statement.

Setting Checkpoints (Turning Pause Switches On) When stored statements are saved, the Pause switches are turned OFF, but they may be turned ON by executing the statements

$$\left\{\begin{matrix}\text{PA}\\\text{PB}\end{matrix}\right\}\ \text{b}\{\text{arith exprn}\}$$

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

Checkpoints with Program Tracing When the above statements are used in conjunction with the trace statements TRON and TROFF (see Section 3.3.9), they allow small sections of a stored program to be thoroughly traced and analysed by the user. These facilities thus provide powerful aids to program debugging.

3.3.13 END statement

The immediate END statement is used to indicate that a user has completed work at a terminal, and the user control block and work areas are automatically returned to the system. A stored END statement has the same function as the stored STOP statement (see Section 3.4.5) and completes stored program execution, returning the terminal to state 1(a) with the message

{seq. no.} STOP

being printed out.

NOTE

The immediate END statement should always be the last statement executed when a user has completed work in ACL mode.

3.3.14 EDIT statement

The EDIT statement allows users to modify statements that have already been saved as part of a stored program. The statement takes the form

EDITb{arith exprn}

and, when executed, the stored statement corresponding to the sequence or statement number specified by the arithmetic 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. To follow the 'edit mode' procedure, consider a pointer to each character in the original statement, namely the OS pointer, where initially OS is one.

Copying Characters in Edit Mode To copy characters from the original statement, the SPACE (b) 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.

Inserting New Characters in Edit Mode To insert a new character, the required character should be typed and it is echoed if correct in syntax - the OS pointer is not altered in this case. The only exception to this is if the space character itself is to be inserted and, in this case, the special character ESC should be entered; in 'edit mode' the syntax analyser translates ESC to the space character, checks the statement syntax and echoes space if it is valid.

Skipping Over Existing Characters in Edit Mode 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
 |
bbbbbbbb(2)DELb2b↵
```

are typed, after the statement has been listed, the resulting stored statement will be

```
211 72 A(2)←26
```

3.3.15 System messages

Sending Messages to New Users To allow system messages to be sent to users a special command which takes the form

```
#:[system 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 enters ACL mode, or after the $command when a user enters non-ACL mode; that is, when a new user logs on to the system.

Broadcasting Messages to Existing Users To send the system message to users currently working at DATERCOM terminals, the combination CNTRL/O must be pressed at the monitor terminal and the message will be printed at each terminal at a time when it is valid to send a line of output to the terminal.

3.3.16 Interrupting stored program execution

Execution of a stored program may be interrupted by entering the question mark character, as the result of an error condition in the stored program or as the result of certain Pause conditions. At this point, the terminal enters state 1(b), and stored statements may be inserted, modified or deleted, or immediate statements executed. If the first input character on a line is CR, then stored program execution resumes 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.

Hitting the Question Mark Character If the stored program execution is interrupted by the question mark character, then the message

```
{seq.no.}?
```

is printed to indicate the sequence number of the *next* stored statement to be executed. This is the stored statement that would be executed if CR is entered as the first character of the next line.

3.4 Stored Statements in ACL Mode

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 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 question mark character is typed at the terminal.

---

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

In addition to the statements discussed below, arithmetic expressions and the TYPE, END and GO TO statements discussed in Section 3.3 can form part of a stored program. The basic forms of stored statements are summarised in Table 2B.

3.4.1 ACCEPT statement

Reading Data Using an ACL Program Data may be read by a program by using the ACCEPT statement which takes the form

```
ACCEPTb{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 by an appropriate arithmetic expression terminated by CR. 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 printed out. When the value of B10 has been given, the line

```
317 A(1,1)←
```

is typed out. After A(1,1) is given a value, the line

```
317 X←
```

is printed. After reading a value for X, the terminal returns to state 2 and the next stored statement is executed.

Interrupting ACCEPT Statement Input ACCEPT statement processing may be interrupted by typing the question mark character and the terminal goes from state 3 to state 1(b). At this point, stored statements may be inserted, modified or deleted, or immediate statements executed. If an initial CR is used to restart stored program execution, then the ACCEPT statement is reprocessed beginning with the first operand.

Error Conditions With ACCEPT Statement Input 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 expression SQR(-3) as the value of the variable A(1,1), then the messages

```
SQR RANGE ERROR
317 A(1,1)←
```

would be printed requesting that the value of A(1,1) be respecified.

Correcting ACCEPT Statement Input Errors Errors made while entering ACCEPT statement input may be corrected in the usual way by using the normal character deletion (DEL and <n), edit mode (<<) and line cancelling (CAN and <<<) facilities. If an error has been made with one of the earlier variables, then the user could restart the ACCEPT statement execution and re-enter all of the variable values. By using the fact that general arithmetic expressions can be used as ACCEPT statement input, the earlier error can be corrected by adding zero times the required variable assignment to the current ACCEPT statement input. In the above example, assume that the user noticed that B10 had incorrectly been given the value 4 instead of 11 at the time that the line

317 X←

printed out. Then by entering

7+0*B10←11↵

the user can assign X the required value of 7 as well as resetting B10 to 11.

Paper Tape Input Terminals with paper tape reading facilities can be used to enter ACCEPT statement input from paper tape. Values entered in this way are usually terminated by a valid CR, but, to cater for a number of AAEC experimental rigs, ACCEPT statement input may be terminated by the colon character (:) or the space character (b). For ACCEPT statement input, the syntax analyser translates colon and space to CR before checking them for validity.

3.4.2 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

CALLb{arith exprn}

and control is passed to the statement with a sequence or statement number corresponding to the value of the arithmetic expression. Calls to any depth are allowed.

Subroutine Arguments All ACL variables are global and subroutine arguments are accessed *via* the symbol table.

3.4.3 RETURN statement

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

3.4.4 CONTINUE statement

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

3.4.5 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

{seq.no.} STOP

being printed.

3.4.6 IF statement

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

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

If the logical relation between the two arithmetic expressions enclosed in brackets is obeyed when the statement is executed, then the third statement outside the brackets is executed; otherwise control is passed to the next stored statement. For example, executing the stored statement

```
273     IF(I←I+1.LE.10) GO TO 20
```

causes the value of the variable I to be increased by one, and if the new value of I is less than or equal to ten, then control is passed to the stored statement having the statement number twenty; otherwise, control is passed to the next stored statement.

Loop Control There is no direct equivalent to the FORTRAN DO statement in the ACL language, so the above form of the IF statement can be used for program looping, that is, if the same set of statements is to be executed a number of times.

3.4.7 PAUSE statement

Execution of the simple PAUSE statement causes the terminal to enter state 1(b) with the message

```
{seq.no.} P
```

being printed.

3.5 Error Messages in ACL Mode

When an error condition occurs in ACL mode, an appropriate message is printed at the terminal. If a stored program was being executed at the time the error occurred (state 2), it is interrupted, the terminal goes into state 1(b) and the error message with appropriate sequence number is printed out; this now allows relevant user action to be taken. The messages, which are self-explanatory, are as follows:

```
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.

Warning Messages The message AREA EXPANDED and the System Message (see Section 3.3.15) will also print out at a terminal as a warning to the user, but this does not interrupt stored program execution.

3.6 Saving and Loading ACL Programs and Symbol Tables Using IBM360 Disk Storage

Creating User Program Libraries DATERCOM access to the resources of the IBM360 computer is also used to provide for the storage and retrieval of ACL programs and symbol tables using IBM360 disk storage. These facilities allow each user to create what is essentially a private library of ACL programs and data referenced by the user's three initials. Programs can be shared by users, since the system allows programs to be loaded from any of the ACL user libraries provided the correct initials are supplied. However, programs can be saved in a user library only if the user's initials are validated by the user's account number.

Information Saved in Internal Code Form The ACL programs and symbol tables are saved in their internal code form and this provides for efficient program loading as the information does not have to be processed by the syntax analyser. The IBM360 program [Backstrom 1975] transfers the ACL information to and from the NOVA computers, 224 bytes at a time, using the DATERCOM communication conventions, and then sends an appropriate message to be printed at the user's terminal to indicate that the user's program has been saved or loaded.

3.6.1 Saving ACL programs and symbol tables

Saving an ACL Program An ACL program can be saved on IBM360 disk storage by executing the immediate statement

```
#SAVEb{progname,int/acctnmbr}
```

The program name (progname) can be up to eight characters long - the first letter must be alphabetic and the rest must be alphanumeric. The initials (int), as contained on the user's IBM360 job card, and account number (acctnmbr), also contained on the same job card, are required to validate the SAVE request - the asterisk character (*) is actually echoed for each of the account number characters entered to preserve system security. When the program has been saved, the message

```
-progname-SAVED AT 10.20 AM ON 76.274
```

is printed, indicating the time of day at which the program was saved if the program was being saved for the first time, or

```
-progname-REPLACED AT 10.20 AM ON 76.274.
```

if it was replacing an earlier version.

Saving ACL Program Plus Symbol Table The ACL program plus the contents of the symbol table can be saved by using the immediate statement

```
#SAVESb{progname,int/acctnmbr}
```

This can be useful for saving a test program and its data for later use.

Saving Only the ACL Symbol Table If only the ACL symbol table is to be saved, then the stored program should be DELETEd (see Section 3.3.6) and the #SAVES statement executed.

ACL data libraries can be saved in this way for later processing by a number of different ACL programs.

No IBM360 Access When the IBM360 computer is not available for ACL save (or load) requests, then the message

NO ACCESS

is printed out in response to the #SAVE(or #LOAD) requests. This message can also mean that there is no space in the IBM360 computer to execute the IBM360 program that satisfies the ACL save (or load) requests at the instant that the request was made, and it is worth trying the request a second time before completing work at a terminal if the user is dependent on IBM360 storage or retrieval (the LIST: option described in Section 3.3.3 could be used in the case of ACL save problems *if* a paper tape punch is available at the terminal).

No IBM360 Response If there is no response at all to the save (or load) request, then control can be returned to the user if the question mark character or the CNTRL/G character is pressed. The message

NO ACCESS

will be printed and the terminal returns to state 1(a). This situation will occur if the IBM360 computer develops a hardware or software problem.

3.6.2 Loading ACL programs and symbol tables

Information previously saved on IBM360 disk storage can be loaded into the user's work area by executing the immediate statement

#LOADb{progname,int}

and, in this case, only the program name and user's initials are required. The message

-progname-LOADED AT 10.30 AM ON 76.275

is printed to indicate that the program has been successfully loaded, or the message

-progname-NOT LOCATED IN ACL LIBRARY

is printed if the program does not exist or is misspelt.

Work Area Expansion Any work area expansions required for the information being loaded are carried out automatically by the system without comment but, if there is not sufficient work space available, the messages

AREA FULL
LOADING TERMINATED

are printed out.

Executing Loaded ACL Programs Information loaded from IBM360 disk storage is stored in the user work area in the same way as if it has been entered from the terminal in state 1(a). A number of load requests may be used, for example, to load an ACL main program, an ACL subroutine package and the ACL symbol table library containing the data to be processed; once these three load requests have been satisfied, execution of the RUN statement or an

immediate GO TO statement will cause the calculation to be performed.

3.6.3 Deleting ACL programs and symbol tables

To delete ACL information no longer required on IBM360 disk storage, the user work area should first be cleared, either by initialising it by entering the CNTRL/G character or by executing the immediate statements

```
CLEAR
DELETE
```

and then a normal SAVE command is executed

```
#SAVEb{progname,int/accnmbr}
```

This 'null' program SAVE request is interpreted as a Delete request, and the response will be

```
-progname-DELETED AT 11.15 AM ON 76.257
```

if the information was removed from the user's library, or

```
-progname-NOT LOCATED IN ACL LIBRARY
```

if the program does not exist.

Efficient Use of IBM360 Disk Storage It is important that users control the number of ACL programs and symbol tables saved on IBM360 disk storage as disk space is a valuable resource that must be shared by all. So far, users have cooperated by tidying up from time to time and removing unwanted information from their libraries, so there has been no need to place limitations on the number of programs that can be saved by each user.

3.6.4 The ACL program library

The initials 'ACL' have been reserved for the ACL program library. Programs in this library are available for general use and they have been documented, with copies of the write-ups being available from Applied Mathematics and Computing (A.M. & C.) Division. Any user may contribute to this library by documenting the program that can then be copied into the ACL library by A.M. & C. Section staff - responsibility for the performance of contributed programs rests with the program authors. Programs currently available from the ACL library are:

(i) EIGEN and EIGMAIN (author N.B. Datyner*): an eigenvalue and engenvector package using the Special Cyclic Jacobi Method.

(ii) LRP and LRPW (author P.L. Sanger): a general linear regression package using full matrix least-squares techniques.

(iii) NLRP and NLRPW (author P.L. Sanger): a general non-linear regression package using full matrix least-squares techniques.

3.6.5 The $LISTACL facility

The $LISTACL program [Backstrom 1975] allows the user to list the names of ACL programs currently held on IBM360 disk storage. To execute the program, the user enters non-ACL mode by entering the command

---

*Vacation Student 1975

```
$LISTACL
```

and the response is

```
INITIALS:
```

The user's three initials should then be entered and the names of the user's ACL programs are then listed in alphabetical order at the terminal followed by

```
-END:n-
```

where 'n' is the total number in the list. All ACL programs currently stored on disk are listed if the space character is entered instead of the three initials. The list may be suspended at any time by entering the question mark character and the response is

```
CONT...
```

Carriage return will allow the list to continue, but anything else will terminate the list (the number n is the same as if the entire list had been printed).

The LISTACL Catalogued Procedure Listings of ACL programs and symbol tables saved on IBM360 disk storage can also be printed out on the IBM360 line printer by using the LISTACL catalogued procedure [Backstrom 1975]. The information to be printed is specified in two places on the user's EXEC card; the user's initials are taken from the stepname and the range of programs required is specified in the PARM field. For example, assuming that the user's initials replace the letters 'INT' in the stepnames in the following, then

```
//INT     EXEC     LISTACL,PARM=PROGNAME
```

lists the program and/or symbol table data named 'progname',

```
//INT     EXEC     LISTACL,PARM='AB*'
```

lists all programs starting with the letters AB, and

```
//INT     EXEC     LISTACL,PARM='*'
```

lists all programs.

3.6.6 FORTRAN access to ACL symbol tables

The FORTRAN callable subroutine ACL [Backstrom 1975] was developed to provide access to ACL symbol table values saved on IBM360 disk storage. The calling sequence is written up in the AAE.FORTLIB User's Manual (together with a second subroutine ACLD that gives access to the ACL Program Library Directory) and is as follows:

```
CALL ACL('PROGNAME,INT',RES,'VARb',&E1,&E2)
```

to access simple variables,

```
CALL ACL('PROGNAME,INT',RES,'Vb',I,&E1,&E2)
```

to access singly subscripted variables, or

```
CALL ACL('PROGNAME,INT',RES,'Vb',I,J,&E1,&E2)
```

to access doubly subscripted variables.

The first argument (PROGNAME,INT) describes the ACL symbol table name and user's initials; RES is a FORTRAN REAL*4 variable to be given the value of the requested symbol table variable; the third argument (VARb, or Vb) is the name of the required symbol table entry in EBCDIC (four characters padded with trailing blanks if necessary for simple variables, and two characters padded with a trailing blank if necessary for subscripted variables); I is the single subscript of a singly subscripted variable or the first subscript of a doubly subscripted variable; J is the second subscript of a doubly subscripted variable; E1 is the FORTRAN statement number to be given control if the requested symbol table element is not located; and E2 is the FORTRAN statement number to be given control if the requested ACL symbol table itself is not located. The following FORTRAN DD card is also required in the FORTRAN GO step:

```
//GO.ACLLIB DD DSN=AAE.ACLLIB,DISP=SHR
```

FORTRAN ACL Subroutine Applications The FORTRAN ACL subroutine has already been used to develop an IBM360 program to print summary reports from results saved as ACL symbol Table values [FPREPORT; McLaughlin & Wong 1975], and also to develop an IBM360 program to plot ACL symbol table values [PLOTACL; Pollard 1975].

4. CONCLUSIONS

The DATERCOM system allows terminals that can communicate with the NOVA computers to have access either to ACL-NOVA or to the resources of the IBM360 computer. The system takes a 'passive role' in the central computer access, and any interactive features must be built into the particular IBM360 programs invoked by the user. Local error correction facilities are provided to assist with the entry of terminal input: these include character deletion, line deletion and tabbing features. IBM360 access has proved to be very popular, with the $LOGON program, for example, resulting in eighteen hundred and sixty seven LOGON programs being saved on IBM360 disk storage for one hundred and six users. DATERCOM allows dedicated access to ACL-NOVA and also uses the IBM360 computer access to provide for the storage and retrieval of ACL programs and symbol tables on IBM360 disk storage. New ACL statements have been included to supplement these load and save features, and eight hundred and seventy ACL programs and/or symbol tables are currently saved on IBM360 disk storage for one hundred and ten users.

The generality of the IBM360 computer communication that has been built into the DATERCOM system and the extensions made to the ACL-NOVA facilities have thus greatly expanded the computing power that can be made available to AAEC scientists.

5. ACKNOWLEDGEMENTS

The software development required to provide terminal access to the IBM360 computer is the result of work done in several areas: The Computer Network software in the PDP9L computer and the Attention Handling software in the IBM360 computer were developed by Dr. D.J. Richardson; the Dataway Terminal Communication software in the NOVA computers was developed by the author; and the IBM360 software written by Mr. R.P. Backstrom; Mr P.J. Ellis designed and built the Serial Multi-User Terminal System for the NOVA computers and contributed to the development of the software to support this system.

6. REFERENCES

Backstrom, R.P. [1975] - IBM360 Software for the AAEC Computer Network. AAEC unpublished work.

Bennett, N.W. & Sanger, P.L. [1973] - The Development of the ACL Language and its Implementation ACL-NOVA. *Aust. Comput. J.*, 5 (3) 105-113.

Johnstone, I.L. [1974] - The Development of the HASP Internal Console. AAEC Unpublished work.

McLaughlin, R.J. & Wong, S.C.K. [1975] - FPREPORT - A FORTRAN Program to produce summary Reports from ACL Symbol Table Values. AAEC Unpublished work.

Pollard, J.P. [1975] - An IBM360 Program to plot ACL Symbol Table Values. AAEC Unpublished work.

Richardson, D.J. [1971] - The AAEC Computer Network Design. *Aust. Comput. J.* 3 (2) 55-59.

Sanger, P.L. [1971] - ACL-NOVA: A Multi-User Conversational Interpreter for the NOVA Computer. AAEC/E221 (revised July 1972).

Sanger, P.L. [1974] - Computing Facilities at the AAEC Research Establishment. *At. Energy Aust.*, 17(2)2-8.

Sanger, P.L. [1976] - The AAEC Dataway Terminal Communication System. *Proc. 7th Aust. Computer Conference*, Perth 30 August - 3 September, 2:610-622.

Sanger, P.L. & Backstrom, R.P. [1973] - IBM360 and NOVA Software Developed to Give the NOVA Computer Access to the Resources of the IBM360 Computer. AAEC/E262.

Sanger, P.L. & Hayes, I.J. [1974] - A Directed-Graph Syntax Analyser for the ACL-NOVA System, AAEC/E312.

Sanger, P.L., Jones, C.G. & Ellis, P.J. [1973] - Programming the NOVA Computer for Dataway Communication. AAEC/E268.

GLOSSARY OF TERMS

Term	Explanation
AAEC Computer Network	Consists of an IBM360 model 65 central computer, a DEC PDP9L computer linked to the IBM360 computer *via* a selector channel, and nine minicomputer systems linked to the PDP9L computer *via* the AAEC Dataway..
PDP9L Computer Link	The first stage in setting up the AAEC Computer Network consisted of linking an 8K PDP9L computer to the IBM360 computer to allow up to 128 different devices or destinations to be handled through a normal IBM selector channel as though they were all standard IBM devices. The PDP9L computer and link thus act as the 'telephone exchange' for Computer Network communication with the IBM360 computer using selector channel two.
AAEC Dataway	The second stage of the network consisted of the development of the AAEC Dataway. Communication over the Dataway occurs between identical Dataway control units on a party line basis, with any two computers being able to communicate with each other if the Dataway is not busy. The NOVA computer was the first computer on the AAEC Dataway and this was followed by the connection of the PDP9L computer. Ten minicomputers have now been connected to the AAEC Dataway.
Dataway Addresses	An 8-bit address code used to establish communication between Dataway control units. 64 of the 256 possible Dataway addresses, in the range X'40' to X'7F', can currently be used to establish communication with the IBM360 computer *via* the PDP9L computer,and the Dataway address becomes the last 8 bits of the corresponding UCB address in the IBM360 computer.
UCB Address	The Channel, Control Unit, Device address used by the IBM360 computer for input/ output operations.
DATERCOM	The Dataway Terminal Communication System developed for the NOVA computers makes use of the computer network facilities to allow terminals to have access either to ACL-NOVA or to the resources of the IBM360 computer.
ACL	A higher level language named ACL (A Conversational Language) developed at the AAEC Research Establishment.
ACL-NOVA	The implementation of the ACL language as a multi-user conversational Interpreter on the DGC NOVA computers.

TABLE 1

MATHEMATICAL FUNCTIONS AVAILABLE IN ACL MODE

ACL Function	Mathematical Purpose
ABS	Absolute value
ACS	Inverse cos ($\cos^{-1}$) (result in radians)
ASN	Inverse sin ($\sin^{-1}$) (result in radians)
ATN	Inverse tan ($\tan^{-1}$) (result in radians)
COS	Trigonometric function (argument in radians)
DPT	Position of decimal point in number (only used with the TYPE statement)
EXP	Exponential
INT	Mathematical integer part (INT(3.1) is 3, INT(3.9) is 3, INT(-1.2) is -2)
LOG	Natural log (base e)
RND†	Pseudo random number in the range 0 to 1
SIN	Trigonometric function (argument in radians)
SQR	Square root
TAN	Trigonometric function (argument in radians)

†The operand must be a simple variable name and, if the value of this variable is positive, then it is used to generate the next pseudo random number, which then replaces the original variable value; this allows the user to specify a starting value and then use successive evaluations with the same variable name argument to generate the pseudo random numbers in the correct sequence. If the variable value is negative, then the last pseudo random number generated by the RND function is used to generate a new pseudo random number which then replaces the original variable value; this allows the user to select a random starting value for the generation of a series of pseudo random numbers. For example, if X has the value 0.1, then execution of the arithmetic expression A←10*RND(X) will cause X to be given the value of a pseudo random number in the range 0 to 1 and A to be given the value of ten times the new value of X, *i.e.* a pseudo random number in the range 0 to 10.

TABLE 2

LIST OF ACL STATEMENTS

A. IMMEDIATE STATEMENTS

Arithmetic expressions

Cb[comments]

CLEAR[bvariable[,variable] ...]

DELETE[barith exprn[,arith exprn]]

EDITb{arith exprn}

END

FTROFF

FTRON

GObTOb{arith exprn}

LIST[:][barith exprn[,arith exprn]]

PAb{arith exprn}

PBb{arith exprn}

RENUMBER[barith exprn[,arith exprn]]

RUN

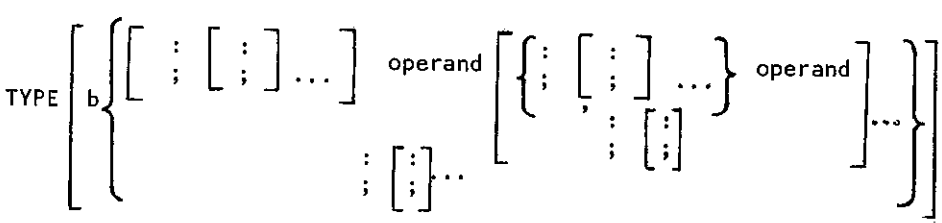
SPACE

STOP

SYMBOLS[:]

TROFF[barith exprn]

TRON[barith exprn]



where the operands take the form,[<arith exprn>,]{'characters' / ["] arith exprn}

#LOAD{bprogname,int}

#SAVE[S]{bprogname,int/acctnmbr}

#:[system message]

B. STORED STATEMENTS

ACCEPTb{variable[,variable] ...}

Arithmetic expressions

Cb[comments]

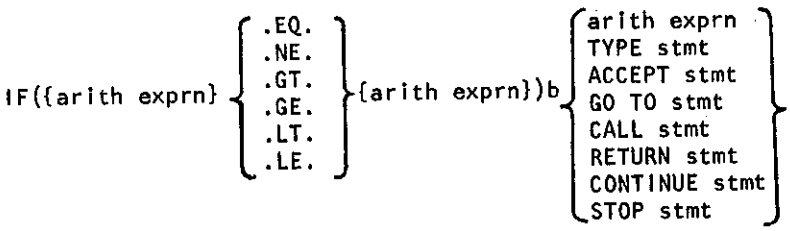
CALLb{arith exprn}

CONTINUE

END

GObTOb{arith exprn}

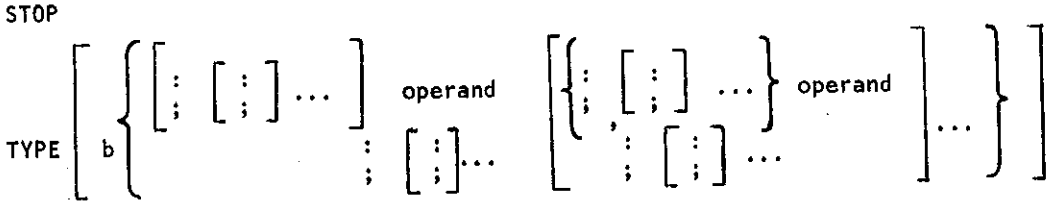
TABLE 2 (Continued)



PAUSE

RETURN

STOP



where the operands take the form, [<arith exprn>,]{'characters'} {['']arith exprn}