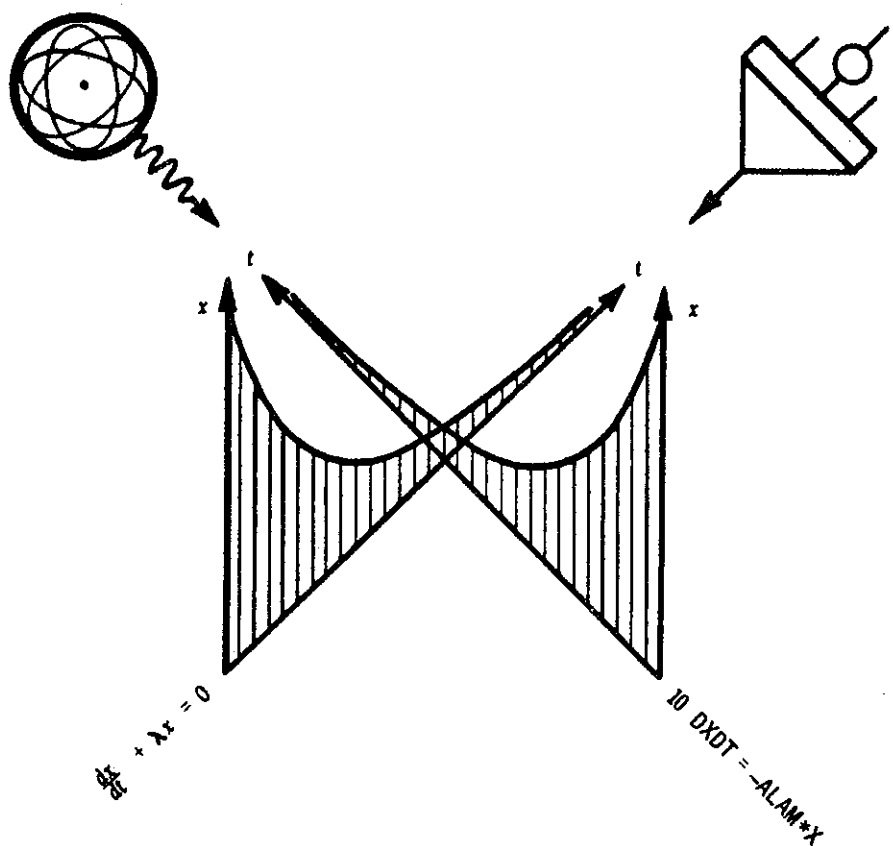
AUSTRALIAN ATOMIC ENERGY COMMISSION
RESEARCH ESTABLISHMENT
LUCAS HEIGHTS

INTERACTIVE COMPUTING

REACTOR PHYSICS, MATHEMATICS AND COMPUTERS

SUMMER SCHOOL, JANUARY 1972

Lecture by P.L. SANGER



January 1972

AUSTRALIAN ATOMIC ENERGY COMMISSION

RESEARCH ESTABLISHMENT

LUCAS HEIGHTS

INTERACTIVE COMPUTING

by

P. L. SANGER

ABSTRACT

The advantages of interactive computing are discussed by comparing the solution of a mathematical problem in Fortran with its solution using the interactive ACL–NOVA system. The ACL–NOVA system is based on the new language ACL, the most important features of which are presented here.

CONTENTS

1. INTRODUCTION

A computer user who has access to a powerful computer, and has a convenient method of getting information into it directly, and who can immediately see the results of a complex calculation in a simple, understandable form has a great advantage over another who submits a program to be run on the computer and gets back tables of numbers a few hours later. If the user can experiment with the input and see the effect of each change then he may quickly begin to understand a complicated problem.

This is the principle of man-machine interaction or on-line problem solving. The point is that both man and computer can each do some things much better than the other. A closed loop system allows the special abilities of each to be used to advantage and the combination may be far more powerful than the sum of the two components (it has been described as the 'two-plus-two equals five effect').

By comparison, the man who has to wait for his results loses 'thinking momentum' and has to collect his thoughts on the problem after every computer run.

The computer network being set up at the Research Establishment will allow powerful interactive computing facilities to be provided at various locations on site. In the meantime, the ACL–NOVA system allows users to solve small to medium-sized problems in an interactive manner.

To see some of the advantages of interactive computing, the solution of the power surge problem using Fortran on the IBM 360 computer (Pollard, 1972) will be compared with its solution using the ACL–NOVA system.

Assuming that sufficient Fortran has been mastered to solve the power surge problem, the conversational programming language, ACL, will now be presented as an alternative, more flexible method of solution. For more details of the ACL language and the ACL–NOVA system, refer to the publications (Sanger 1971, Bennett and Sanger 1971).

2. THE ACL–NOVA SYSTEM

2.1 General Discussion

The ACL language (ACL stands for A Conversational Language) was developed at the A.A.E.C. It was implemented on a NOVA computer supporting five teletypewriter terminals and this is referred to as the ACL–NOVA system.

The ACL language consists of immediate statements and stored statements. Stored statements are translated and saved in a user work area from which they may be recalled and executed under user control. 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.

An immediate statement, which does not have a sequence number, is translated and executed when typed and is then discarded. Immediate 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. The basic statement forms used for immediate and stored statements are shown in the ACL–NOVA summary in Appendix 1.

2.2 Input from a Terminal

The ACL-NOVA system currently supports five teletypewriter terminals. Input to the system is specified by pressing keys on the typewriter-like keyboard. To begin work at a terminal, you press the CTRL key and the G key at the same time, and the system responds by typing ACL-NOVA and spacing a few lines. Once the terminal has been initialised in this way, statements may be stored or immediate statements executed.

Input begins from column 1, which is taken to be the leftmost position of the teletype carriage that results from pressing the Carriage Return (CR) key, and may consist of up to 72 characters. If input is continued past column 72, the whole line is cancelled and must be entered again. In the ACL-NOVA system, then, you can think of input on a 72 column layout. For statements to be stored for later execution, this layout takes the form

1	4	5	7	8 ACL 72
230				A←4
235		1		B←6
240		23		A + B

Immediate statements begin from column 1 and have the layout:

1 ACL 72
26 + 4
A2←23.5
SQR(42.978)

The terminals attached to the NOVA computer are operated in what is termed full-duplex mode. This means that a character pressed at the teletype keyboard is sent to the computer, but is not printed at the teletype unless it is sent back to the terminal (echoed) by the NOVA computer. By making use of this feature, only characters that are valid in syntax are 'echoed' at a terminal; that is, only characters that make sense are typed at the terminal. For example, if you pressed the keys 3++3 at the keyboard, the system would only type 3+3 at your terminal. A statement is executed when it is completed by a 'valid' carriage return.

This 'echo-checking' mechanism is a very valuable feature of the ACL-NOVA system and provides interaction even while statements are being entered at the terminal.

2.3 Arithmetic

All arithmetic operations are carried out using single precision floating point numbers. 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.

2.4 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.

In ACL all variables are treated as real variables, so that there is no distinction between real and integer variables as there is in Fortran. In addition, there is no conflict between simple and subscripted variable names in ACL. For example, in ACL one can use A, A(1), A(2,3) as separate variables, while this is not allowed in Fortran. The subscript zero is also allowed in ACL.

2.5 Arithmetic Statements and Expressions

Arithmetic statements and expressions play an important role in the ACL language and their definitions and mode of evaluation should be understood.

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 such as COS, EXP, LOG, SIN and SQR.

Examples: addition A+B
subtraction A–B
multiplication A*B
division A/B
exponentation A↑B (in Fortran this is A**B)
use of functions EXP(A+B↑3) equivalent to e^{a+b^3}

The mathematical operators have the usual heirarchy 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. For example:

3+4*6↑2 would give the result 36*4+3=147

3+(4*6)↑2 would give the result 24*24+3=579

((3+4)*6)↑2 would give the result 42*42=1764

Brackets are also necessary to prevent two arithmetic operators from appearing next to each other, for example,

A*–B must be written as A*(–B).

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 this is

an important difference between ACL and Fortran. When multiple assignments occur in an expression, 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 pair of opening and closing brackets. The expression enclosed between these 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.

Examples: 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 $6^3+4=220$.

To do this in Fortran, the following five statements are required.

```
Z=9.
B=SQR(Z)
A=2.*B
X=4.
Y=A**B+X
```

Another important difference between ACL and Fortran is that no DIMENSION statement is required in ACL to indicate the size of singly or doubly subscripted arrays. Subscripted variables are added to the symbol table as they are defined, and a user can thus use the variables A(5), A(7), A(9) without needing to say DIMENSION A(9) as required in Fortran.

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-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.

2.6 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 will not have to be altered if the sequence numbers of the stored statements are changed.

3. STORED STATEMENTS

Statements that have a sequence number associated with them are classed as stored statements, and these are used to build up a stored program that may later be executed. Arithmetic statements and arithmetic expressions may form part of a stored program, and they may also be used as operands in the statements that are described below.

3.1 ACCEPT Statement

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

ACCEPT ↓ {variable[, variable [, variable.....]]}

The curly brackets are used to indicate a field that must be present, while the square brackets are used to indicate an optional field. In the case of the ACCEPT statement, then, at least one variable name must be specified to complete the statement.

When an ACCEPT statement is executed, its sequence number is typed on a new line followed by the variable name and assignment arrow. Execution of the stored program is temporarily suspended 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, and program execution then proceeds normally.

For example, execution of the stored statement

1	5	8 ACL 72
317	10	ACCEPT B51,A(1,1),X

causes the line

1	5
317	B51←

to be typed at the terminal. When the value of B51 has been given (completed by a valid CR), the line

1	5
317	A(1,1)←

is then printed. After A(1,1) is given a value, the line

1	5
317	X←

is printed. On reading X, the ACCEPT statement is fully processed and program execution continues normally with the next stored statement.

3.2 GO TO Statements

Stored statements are usually processed sequentially in the order of increasing sequence numbers. However, it is possible to transfer out of the current sequence of statement processing by executing a GO TO statement which takes the form

GO TO {arith stmt or exprn}

When this statement is executed, control is passed to the stored statement with the sequence number (if 3 digits) or statement number (if 1 or 2 digits) obtained by evaluating the arithmetic statement or expression.

Examples:

1	5	8
325		GO TO 211

causes control to be passed to the stored statement with the sequence number 211.

1	5	8
411		GO TO I←2*2E2-309

cause the variable I to be assigned the value 91, and control then to be passed to the stored statement with the statement number 91.

3.3 IF Statement

The most important statement in controlling the logical flow of a program is 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 between the two arithmetic statements or 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.

Example

1	5	8
273		IF(I←I+1.LE.10) F1←0

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 10 then the variable F1 is set to zero before executing the next stored statement.

There is no DO statement in the ACL language and the IF statement must be used for program looping, that is, if the same set of statements are to be executed a number of times.

3.4 TYPE Statement

The results of the calculations carried out by a stored program may be printed at the terminal by using the TYPE statement. The value of an expression may be typed out in a format determined by its magnitude, as discussed in Section 2.5, or entirely in exponent format. For example,

```
1   5   8
512     TYPE A3
```

causes the value of A3 to be typed in a form determined by its magnitude, while the statement

```
1   5   8
512     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. For example, if A1 has the value 271 and A(2,1) has the value 6731 in all of the following examples, then

```
1   5   8
473     TYPE A1,A(2,1)
```

causes the line,

```
1   5
271 6.731000E+03
```

to be typed at the terminal.

Literal data (character output or headings) can be printed by enclosing it within apostrophes. For example, the statement

```
1   5   8
671     TYPE 'ANSWER=' ,A1
```

causes the line

```
1
ANSWER=271
```

to be printed.

To continue output on a new line, a semi-colon should be used as the delimiter, so that

```
1   5   8
423     TYPE A1;A(2,1)
```

causes the lines

```
1
271
6.731000E+03
```

to be typed at the terminal. The semi-colon delimiter can also be used to space a number of lines, for example

```
1    5    8
483       TYPE ;;;
```

causes the teletypewriter to be spaced three lines.

Output may be placed at a particular carriage position by indicating the required position by means of an arithmetic statement or expression enclosed between the < and > symbols. This positional parameter must appear before the required variable or literal data and be separated from it by a comma. For example, the statement

```
1    5    8
821       TYPE <27>,'A1=',''A1
```

causes the line

```
1                    27 28
                        A1=2.710000E+02
```

to be printed.

3.5 STOP Statement

Execution of a STOP statement completes the execution of a stored program. The statement takes the form

```
1    5    8
418       STOP
```

and when it is executed the message

```
1    5
418  STOP
```

is printed at the terminal.

3.6 END Statement

The END statement is used to indicate that work has finished at a terminal. It should be the last statement executed before leaving the terminal.

4. POWER SURGE PROBLEM

You have now been introduced to enough of the ACL language to be able to set up a stored program to solve the power surge problem (Pollard, 1972). That is, given

$$\frac{dp}{dt} = -(p - 1 - be^{-t})p,\ p(0) = 1,\ (e = 2.718282),$$

then we require the solution of this differential equation so that we may calculate the maximum power surge

$$h(b) = \max_{0 \leqslant t \leqslant 10} p(t)$$

The particular values of b to be considered are b=0.3 and 0.5 since we are given an experimentally measured result

(B) $h(0.3) = 1.12 \pm 0.01$

as a check and we are required to determine whether

(A) $h(0.5) > 1.2$ or not.

To simplify the presentation, the approximate solution

$$p(t) = e^{(bte^{-t})}$$

will be used instead of solving the differential equation numerically. The peak of the power surge (maximum if b is +ve, minimum if b is -ve) will be found from successive evaluations of p(t) using the above formula. (We will ignore the fact that we could use the analytic result

$$h(b) = e^{0.3678794b}$$

for this approximate case).

The computational procedure for the approximate solution is similar to that for the numerical solution of the differential equation and it is shown below

$\delta t = 0.1$

$t_A = 0$

$p_A = 1$

b = variable (0.3, then 0.5 then any other value)

$t_B = t_A + \delta t$ ←

$p_B = e^{(bt_B e^{-t_B})}$

If $(b(p_B - p_A)) \leq 0$ then a peak is located and we type results

$t_A = t_B$

$p_A = p_B$

go to ——→

The first version of an ACL program using this computational procedure could be the following

```
1    5    8
110       DT←0.1
120       TYPE ;' POWER SURGE PROBLEM (APPROX.) JANUARY 72'
130       TYPE ;' INPUT B=INITIAL DISTURBANCE'
140       TYPE ;' OUTPUT T=TIME'; <8>,'P=POWER'
150       TYPE <8>,' CORRESPONDING TO MAX OR MIN POWER SURGE';;
160  1    ACCEPT B
170       TA←0
180       PA←1
190  2    TB←TA+DT
200       PB←EXP(B*TB*EXP(-TB))
210       IF(B*(PB-PA).LE.0) GO TO 3
220       PA←PB
230       TA←TB
240       GO TO 2
250  3    TYPE <5>,'T=',TA,' P=',PA;;
260       GO TO 1
270       STOP
280       END
```

However, this program can be improved in a number of ways. For example, statements 170 and 180 can be combined to give

```
1     5    8
170          PA←-1*TA←-0
```

Statements 190 and 200 can also be combined as indicated below.

```
1     5    8
190   2    PB←-EXP(B*TB*EXP(-TB←-TA*DT))
```

An improved version of the program could thus be the following.

```
1     5    8
110          DT←-0.1
120          TYPE ,' POWER SURGE PROBLEM (APPROX.) JANUARY 72'
130          TYPE ,' INPUT B=INITIAL DISTURBANCE'
140          TYPE ,' OUTPUT T=TIME';<8>,'P=POWER'
150          TYPE <8>,'CORRESPONDING TO MAX OR MIN POWER SURGE';;
160   1      ACCEPT B
170          PA←-1*TA←-0
190   2      PB←-EXP(B*TB*EXP(-TB←-TA*DT))
210          IF(B*(PB-PA).LE.0) GO TO 3
220          PA←-PB
230          TA←-TB
240          GO TO 2
250   3      TYPE <5>,'T=',TA,' P=',PA;;
260          GO TO 1
270          STOP
280          END
```

Results obtained from running the above program (approx. solution) and an extension of the program to solve the differential equation numerically ('exact' solution) are shown in Table 1. In each case, the results obtained for b=0.3 are in good agreement with the experimental result $h(0.3)=1.12\pm0.01$. However, the approx. solution gives $h(0.5)=1.202$, while the 'exact' solution which gives $h(0.5)=1.193$ is really needed to show that $h(0.5) < 1.2$.

The difference between the results for the two solutions increases as the magnitude of b increases and this is to be expected from the assumptions made in deriving the approx. solution.

Looking at the way the program has been set up to calculate the approx. solution, you can see how easy it is to change the value of b and how quickly one can get a feel for the effect of b on the maximum or minimum power surge. The ability of the user to interact with the problem by immediately being able to try various values of b depending on the outcome of the previous values tried is a considerable advantage over the FORTRAN user who must anticipate suitable values of b prior to submitting his run.

What change in the results, if any, would you expect if you changed DT? — let's run the program and see. Do the results of the approx. solution agree with the analytic result $h(b)=e^{0.35787946}$? — let's write a small ACL program and see. Any other aspects? — well let's try them too. With interactive computing you can observe the effect of various modifications to the basic program as ideas come to mind!

5. IMMEDIATE STATEMENTS

So far, the ACL statements that you have met are very similar to Fortran statements and it is now time we looked at the immediate statements that provide most of the interaction with the user.

Statements that do not have a sequence number associated with them are classed as immediate statements and they are executed as soon as a 'valid' CR is typed. Arithmetic statements, arithmetic expressions, TYPE statements and the END statement can be executed as immediate statements, in addition to those mentioned specifically below.

5.1 RUN Statement

Perhaps the first thing a user wants to know once he has typed in a stored program is how to execute it. This is done by using the RUN statement which begins execution of the stored program starting at the statement with the lowest sequence number. The statement takes the form

```
1
RUN
```

The results in Table 1 were obtained by using this statement to begin execution of stored programs designed to solve the power surge problem.

5.2 GO TO Statement

An immediate GO TO statement can be used to begin execution of a stored program with any stored statement and it takes the form

```
1
GO TO {arith stmt or exprn}
```

For example, the power surge problem could be started by executing the statement

```
1
GO TO 110
```

as shown in Table 2.

5.3 LIST Statement

Stored statements are listed (printed out) at a terminal (in sequence number order) when the LIST statement is executed. A single statement, a small group of statements or the entire stored program may be listed.

Examples: (i)

```
1
LIST
```

causes the whole program to be listed

(ii)

```
1
LIST 67
```

causes the stored statement with the statement number 67 to be listed

(iii)

```
1
LIST 117,421
```

causes all the stored statements starting from sequence number 117 and finishing with sequence number 421 to be listed. Some of the listing possibilities are shown in Table 3 for the case of the power surge program.

The teletypewriter terminals have a paper tape reader and paper tape punch attached to them. Input from a terminal can come from either the teletype keyboard or the paper tape reader. When output is being printed at a terminal, it will also be punched onto paper tape if the paper tape punch is ON. The paper tape punch is normally OFF.

If you have tested your program and wish to keep a copy of it on paper tape, then you should execute the statement

```
1
LIST:
```

In this case, to give you time to turn the punch ON, the listing is delayed until a CR is entered as the first character on the next line. When the CR is entered at the terminal, 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.

5.4 SYMBOLS Statement

Execution of the SYMBOLS statement, which takes the form

```
1
SYMBOLS [:]
```

causes the values of all the variables in the symbol table to be printed out in the form:

```
1
A1←-1.320000E-20
Z13←21
A2(7)←.01
NA(3,7)←0823
```

If the colon is present, it indicates that the output is to be punched onto paper tape. Entering? from the keyboard will again cause listing to be terminated.

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

5.5 Program Tracing

To assist in debugging a program, special tracing facilities are built into the ACL language.

A statement is said to be traced if any symbol table assignments that occur while the statement is being executed are typed at the terminal. These assignments are typed in the form

1	5
{sequence number}	{variable}←{value}

or for example,

1	5
317	A72←6 13

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

1	6
TRON	{arith stmt or exprn}

before beginning the execution of a stored program. For example, in the case of the power surge problem, if the immediate statements

1	6
TRON	190
TRON	220
TRON	230

were executed, followed by the execution of the RUN statement then the results shown in Table 4(a) would be obtained. When you have finished tracing individual statements you should turn the trace off using statements of the form

1	7
TROFF	{arith stmt or exprn}

as shown, for example in Table 4(b).

If you are really having trouble with your program then you can get a full trace of your program by executing the immediate statement

1	6
TRON	

before saying RUN.

Every stored statement that is subsequently executed is listed at the terminal and traced as shown for example in Table 5(a). This allows a complete trace of a stored program to be obtained. When you have finished your full trace you should execute the immediate statement

1	7
TROFF	

as shown in Table 5(b).

5.6 Interrupting Program Execution

Execution of a stored program may be interrupted by the character? being typed at the terminal as the result of an error condition in the stored program or as the result of certain PAUSE conditions. At this point, stored statements may be inserted, modified or deleted, or immediate statements executed. If the first input character on a line is CR, then 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.

For example, in Table 6 you can see the effect of typing ? while the program was calculating results for b=6. The message

```
1   5
210 ?
```

was printed to indicate that the next statement to be executed was the one with the sequence number 210. At this stage the SYMBOLS statement was executed to list the contents of the symbol table. The character CR was next typed as the first character on the line, and the stored program continued and printed the results for b=6.

The second calculation of the results for b=6 was also interrupted by the typing of ? The value of TA was determined, and statement 210 listed. Execution of an immediate GO TO statement caused a STOP statement to be executed and execution of the stored program was terminated with the message

```
1   5
270 STOP
```

being printed at the terminal.

5.7 Error Correction

Errors that 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(CR). 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,

```
1                    ACL
A2←B÷SQR(AZ1÷C←3)-X3(4,2)-6<2
```

would result in the new line

```
1                    ACL
A2←B÷SQR(AZ1÷C←3)-X3(4,2)
```

being typed out. 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 next Section) by typing <<(CR) after the last input character.

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

5.8 EDIT Statement

The EDIT statement is used to modify statements that are part of a stored program in what is described as '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 a character from the original statement, the SPACE (␣) key is pressed for each required 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 it is correct in syntax. If a space character itself is to be inserted, then the special character ESC should be typed. The OS pointer is not altered in this case.

To jump over a character in the original statement the DEL or RUB OUT character should by typed and the OS pointer is simply 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 RUB OUT have no effect. For example, the stored statement

1	5	8
211	72	A3←6

can be modified by executing the immediate statement

1	6
EDIT	211

or

1	6
EDIT	72

If the characters

1
␣␣␣␣␣␣␣(2) (Rub Out) ␣2␣

are typed, after the statement has first been listed, then the resulting stored statement would be

1	5	8
211	72	A(2)←26

5.9 Special Input Features

When you are typing a statement that is to be part of a stored program, you need a three digit sequence number followed by a space, followed by three spaces if you don't want a statement number and then the stored statement beginning from column 8 completed by a 'valid' CR. However to make this process a little easier, the system is designed so that the pressing of SPACE at the column 1 position will cause a sequence number to be automatically generated for you 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 last sequence number that was typed in. The sequence number 110 is given if there was no previous sequence number.

If no statement number is required, then the SPACE key should again be pressed at the column 5 position, and the teletype carriage is automatically positioned to column 8. The required stored statement can be entered at this point.

If you wish to delete a statement from your program then you must type in the relevant sequence number followed by a space and then type CR. For example

1
231 ␣ (CR)

causes statement 231 to be deleted.

6. CONCLUSIONS

The syntax checking of input and the powerful editing and error correction facilities provided by the ACL-NOVA system allow a user to set up a stored program very simply and easily.

In no time at all, the user can be producing results and with the interrupt and tracing facilities at his disposal he can also quickly debug his program. By having the opportunity to control the input to a program, the user can gain valuable insight into his calculations.

However, with such ready access to problem solution using interactive computing one must be careful not to be carried away by the computer. There are times when the only way to discover an error is to THINK – and that is solely our perogative.

7. REFERENCES

Bennett, N.W. and Sanger, P.L. (1971). – ACL – A New Language for Small Computers. To be published.

Pollard, J.P. (1972). – Numerical Mathematics and Fortran. AAEC/S1.

Sanger, P.L. (1971). – ACL-NOVA: A Multi-user Conversational Interpreter for the Nova Computer. AAEC/E221.

APPENDIX 1

ACL-NOVA SUMMARY

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

All arithmetic is performed on 32 bit floating point numbers representing the range $5.4 \times 10^{-79} \leq$ number $\leq 7.2 \times 10^{75}$ For input numbers may be in free format.

Three types of variables may be used – simple variables, singly subscripted variables or doubly subscripted variables. Singly subscripted variables may have subscripts in the range 0 to 65535. Subscripts for doubly subscripted variables may be in the range 0 to 255

The basic operands in an arithmetic statement or expression are variables and numbers. The relevant operators are +, -, * /(divide), ↑(power), ←(assignment) and the functions ABS,ATN,COS, DPT,EXP,INT,LOG,SIN and SQR. The mathematical operators have the usual heirarchy of ↑ first, * or / next, and then + or – ,

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. If a number of assignment arrows occur in an expression then they are processed from right to left.

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 typed at the terminal in two possible formats depending on its magnitude. The result is given in exponent form if $|\text{result}| \leq 10^{-4}$ or $|\text{result}| \geq 10^{3}$; otherwise it is typed in non-exponent form with seven significant figures (rounded) if necessary.

Every statement that is to be stored for later execution must have a sequence number in the range 100 to 999. These statements are stored in the user's work area in an order based upon increasing sequence numbers.

A one or two digit statement number in the range 0 to 99 may also be associated with each stored statement.

To begin work at a terminal at any time the user types the CONTROL G combination of keys at the teletype keyboard. If work space is available the terminal gives CR and three line feeds types ACL-NOVA followed by another CR and three line feeds and the terminal is now ready to receive statements. If work space is not available the CONTROL G (BELL) combination is 'echoed' at the terminal to warn the user that he cannot use the system at this stage.

Keyboard input begins from column 1 which is taken to be the leftmost position of the teletype carriage that results from a CR.

Statements to be stored for later execution take the form;

cols 1-4	cols 5-7	cols 8-72
{ sequence number ↵ }	{ blanks / statement number }	{ statement to / be stored }

Stored statements may be deleted by typing:

cols 1–4

{sequence number ⏎}↵

Statement numbers may be altered or removed by typing:

cols 1–4	cols 5–7
{sequence number ⏎}	{ blanks / statement number }↵

Statements that do not begin with {sequence number ⏎} are termed immediate statements.

Errors that occur while a statement is being entered at the keyboard may be corrected quite simply. To delete the last two characters that were accepted as input, for example, type <2↵. A statement may be corrected in edit mode by typing <<↵ after the last input character. Finally, the whole input line may be deleted by typing <<<↵ after the last input character.

Stored statements may be altered by using the EDIT statement.

IMMEDIATE STATEMENTS

Arithmetic statements or expressions can be executed as immediate statements, in addition to those summarised below.

col 1

SET [⏎{device[.number]}[,{device[.number]}[,{device[.number]}...]]]

RUN

GO⏎TO⏎{arith stmt or exprn}

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

SYMBOLS [:]

CLEAR [⏎ variable[,variable[,variable ...]]]

SPACE

FTRON

FTROFF

TRON [⏎ arith stmt or exprn]

TROFF [⏎ arith stmt or exprn]

TYPE [⏎{ operand[{[;[;[;...]]]}operand[{[;[;[;...]]]}operand ...]] / operand[{[;[;[;...]]]}operand[{[;[;[;...]]]}operand ...]] }]

Where the operands take the form, [< arith stmt or exprn >,] { 'characters' / [''] arith stmt or exprn }

col 1

PB ↵ {arith stmt or exprn}
PA ↵ {arith stmt or exprn}
PC ↵ {arith stmt or exprn}
END
SUSPEND [:]
EDIT ↵ {arith stmt or exprn}

STATEMENTS TO BE STORED

Arithmetic statements or expressions may form part of a stored program. The TYPE, END and GO TO statements can also form part of a stored program in addition to the statements summarised below.

col 8

ACCEPT ↵ {variable [,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

Execution of a stored program may be interrupted by typing the character ?

To ensure that the maximum amount of work space is available to the ACL–NOVA system, the END statement must be executed when a user completes work at a terminal.

RESULTS FOR POWER SURGE PROBLEM

```
C
C  APPROX. SOLUTION
C

RUN


 POWER SURGE PROBLEM (APPROX.) JANUARY 72

 INPUT B=INITIAL DISTURBANCE

 OUTPUT T=TIME
        P=POWER
        CORRESPONDING TO MAX OR MIN POWER SURGE

160 B=0.3
     T=1 P=1.116684

160 B=0.5
     T=1 P=1.201943

160 B=0.7
     T=1 P=1.293712

160 B=1
     T=1 P=1.444668

160 B=6
     T=1 P=9.090924

160 B=10
     T=1 P=39.59865

160 B=-5
     T=1 P=.1589132

160 B=-10
     T=1 P=.02525339

160 B=-20
     T=1 P=6.377344E-04
```

```
C
C  'EXACT' SOLUTION
C

RUN


 POWER SURGE PROBLEM 'EXACT' JANUARY 72

 INPUT  B=INITIAL DISTURBANCE
 OUTPUT T=TIME
        P=POWER
        CORRESPONDING TO MAX OR MIN POWER SURGE

160 B=0.3
     T=1 P=1.113474

160 B=0.5
     T=1 P=1.192636

160 B=0.7
     T=.9 P=1.274961

160 B=1
     T=.9 P=1.403749

160 B=6
     T=.6000005 P=4.236405

160 B=10
     T=.5000005 P=6.970009

160 B=-5
     T=1.700003 P=.06916544

160 B=-10
     T=2.300005 P=.001981707

160 B=-20
     T=3.000008 P=5.202669E-05
```

TABLE 2

EXECUTION OF STORED PROGRAM USING IMMEDIATE GO TO STATEMENT

```
GO TO 110

 POWER SURGE PROBLEM (APPROX.) JANUARY 72

 INPUT B=INITIAL DISTURBANCE

 OUTPUT T=TIME
        P=POWER
        CORRESPONDING TO MAX OR MIN POWER SURGE

160 B=6
     T=1  P=9.090924
```

TABLE 3

LIST STATEMENT EXAMPLES

```
LIST 190
190 2   PB←EXP(B*TB*EXP(-TB←TA+DT))

LIST 190,240
190 2   PB←EXP(B*TB*EXP(-TB←TA+DT))
210     IF(B*(PB-PA).LE.0) GO TO 3
220     PA←PB
230     TA←TB
240     GO TO 2

LIST
110     DT←0.1
120     TYPE ;' POWER SURGE PROBLEM (APPROX.) JANUARY 72'
130     TYPE ;' INPUT B=INITIAL DISTURBANCE'
140     TYPE ;' OUTPUT T=TIME';<8>,'P=POWER'
150     TYPE <8>,'CORRESPONDING TO MAX OR MIN POWER SURGE';!
160 1   ACCEPT B
170     PA←1+TA←0
190 2   PB←EXP(B*TB*EXP(-TB←TA+DT))
210     IF(B*(PB-PA).LE.0) GO TO 3
220     PA←PB
230     TA←TB
240     GO TO 2
250 3   TYPE <5>,'T=',TA,' P=',PA;!
260     GO TO 1
270     STOP
280     END
```

TABLE 4

TRACING INDIVIDUAL STATEMENTS

(a)
```
TRON 190
TRON 220
TRON 230
RUN

 POWER SURGE PROBLEM (APPROX.) JANUARY 72

 INPUT B=INITIAL DISTURBANCE

 OUTPUT T=TIME
        P=POWER
        CORRESPONDING TO MAX OR MIN POWER SURGE

160 B=6
190 TB=.1
190 PB=1.720995
220 PA=1.720995
230 TA=.1
190 TB=.2
190 PB=2.671065
220 PA=2.671065
230 TA=.2
190 TB=.3
190 PB=3.794195
220 PA=3.794195
230 TA=.3
190 TB=.4000003
190 PB=4.996654
220 PA=4.996654
230 TA=.4000003
190 TB=.5000005
190 PB=6.169342
220 PA=6.169342
230 TA=.5000005
190 TB=.6000005
190 PB=7.211826
220 PA=7.211826
230 TA=.6000005
190 TB=.7
190 PB=8.049886
220 PA=8.049886
230 TA=.7
190 TB=.8
190 PB=8.643258
220 PA=8.643258
230 TA=.8
190 TB=.9
190 PB=8.984277
220 PA=8.984277
230 TA=.9
190 TB=1
190 PB=9.090924
220 PA=9.090924
230 TA=1
190 TB=1.1
190 PB=8.997527
     T=1 P=9.090924
```

(b)
```
TROFF 190
TROFF 220
TROFF 230
RUN

 POWER SURGE PROBLEM (APPROX.) JANUARY 72

 INPUT B=INITIAL DISTURBANCE

 OUTPUT T=TIME
        P=POWER
        CORRESPONDING TO MAX OR MIN POWER SURGE

160 B=6
     T=1 P=9.090924

160 B=0
170 ?
GO TO 270
270 STOP
```

TABLE 5

COMPLETE TRACE OF PROGRAM

(a)
```
TRON
RUN

110     DT=0.1
110 DT=.1
120     TYPE !' POWER SURGE PROBLEM (APPROX.) JANUARY 72'

 POWER SURGE PROBLEM (APPROX.) JANUARY 72
130     TYPE !' INPUT B=INITIAL DISTURBANCE'

 INPUT B=INITIAL DISTURBANCE
140     TYPE !' OUTPUT T=TIME';<8>,'P=POWER'

 OUTPUT T=TIME
        P=POWER
150     TYPE <8>,'CORRESPONDING TO MAX OR MIN POWER SURGE'!!
        CORRESPONDING TO MAX OR MIN POWER SURGE

160 1   ACCEPT B
160 B=6
170     PA=1+TA=0
170 TA=0
170 PA=1
190 2   PB=EXP(B*TB*EXP(-TB-TA+DT))
190 TB=.1
190 PB=1.720995
210     IF(B*(PB-PA).LE.0) GO TO 3
220     PA=PB
220 PA=1.720995
230     TA=TB
230 TA=.1
240     GO TO 2
190 2   PB=EXP(B*TB*EXP(-TB-TA+DT))
190 TB=.2
190 PB=2.671065
210     IF(B*(PB-PA).LE.0) GO TO 3
220     PA=PB
220 PA=2.671065
230     TA=TB
230 TA=.2
240     GO TO 2
190 2   PB=EXP(B*TB*EXP(-TB-TA+DT))
190 TB=.3
190 PB=3.794195
210     IF(B*(PB-PA).LE.0) GO TO 3
220     PA=PB
220 PA=3.794195
230     TA=TB
230 TA=.3
240     GO TO 2
190 2   PB=EXP(B*TB*EXP(-TB-TA+DT))
190 TB=.4000003
190 PB=4.996654
210     IF(B*(PB-PA).LE.0) GO TO 3
220     PA=PB
220 PA=4.996654
230     TA=TB
230 TA=.4000003
240     GO TO 2
        .
        .
        .
```

(b)
```
TROFF
RUN

 POWER SURGE PROBLEM (APPROX.) JANUARY 72

 INPUT B=INITIAL DISTURBANCE

 OUTPUT T=TIME
        P=POWER
        CORRESPONDING TO MAX OR MIN POWER SURGE

160 B=6
     T=1 P=9.090924

160 B=0
170 ?
GO TO 270

270 STOP
```

TABLE 6

INTERRUPTING EXECUTION OF A STORED PROGRAM

```
RUN

 POWER SURGE PROBLEM (APPROX.) JANUARY 72

 INPUT B=INITIAL DISTURBANCE

 OUTPUT T=TIME
        P=POWER
        CORRESPONDING TO MAX OR MIN POWER SURGE

160 B=6
210 ?
SYMBOLS
DT=.1
B=6
TA=.2
PA=2.671065
TB=.3
PB=3.794195

      T=1  P=9.090924

160 B=6
240 ?
TA
  .3
LIST 210
210     IF(B*(PB-PA).LE.0) GO TO 3
GO TO 270

270 STOP
```

? typed here

CR typed here

? typed here

immediate GO TO statement executed to finish program