large amount of work involved in changing the Gestalt system program to correspond to the change of vocabulary required to include some new feature. It is hoped that a solution to this difficulty will be found by writing a program to generate translation programs which will translate from statements in arbitrary Gestalt languages into selections of computer behavior.

The goal of the experimental programming phase of this work is to allow the programmer to alter drastically his planned attack on a very large and complex problem, and try out the new solution within a matter of days, while the new approach is fresh in his mind. All too often a volatile thought pattern disappears in the months of arduous toil required to program a complex problem using ordinary techniques. It is unlikely that present and future problems being considered at the Servomechanisms Laboratory could be solved with limited manpower without the use of these techniques.

Concluding Remarks

It seems appropriate to close this paper by again acknowledging the very real debt which is owed to all of the various schools of computer programming for substantial contributions upon which this paper is based. The emergence and development of these various techniques in the past several years have established firmly the intellectual climate necessary for continued expansion in these directions. There are several groups in the United States which for some time have been developing systems for using computers which have many, if not all, of the attributes of Gestalt programming systems as defined here. The purpose of this paper has been to try to establish the outlines of the abstract structure of this type of system. It is hoped that this analysis will prove useful to all who are interested in connecting humans and computers by clarifying the problems and relationships involved.

In its full generality Gestalt programming is not just a computer technique, but is a problem-solving technique, i.e., a point is reached where it is difficult to tell which is more important, the human, the problem, or the computer. The extension of these techniques and concepts is sure to have a profound influence on the design and operation of future computers, so much so that it seems probable that the term "computer" for describing these mechanisms will become less and less appropriate. The day is fast approaching, if it is not already here, when the arithmetic capabilities of a machine will be its least valuable attributes. If the logical trend toward more and more elaborate systems of this type continues, the primary attribute of a computing machine will be its flexibility in the most general sense. Even if significant advances in the speed of computer elements can be achieved, these gains will be swiftly swallowed up if the logical design of these machines is not advanced to fit the peculiar requirements of these techniques, to obtain the same results with much fewer operations.

At the present state of the art, these future developments can only be sensed in a most intuitive way, although, for example, the growing concept of a microprogrammed computer appears to be a well-founded first step. Continued and rapid advance in these directions both in programming techniques and in computer design, can only be achieved by building on experience gained in studies using present-day facilities. It is hoped that the presentation of these ideas will encourage the participation of other groups in this fascinating line of endeavor.

A Truly Automatic Computing System

MANDALAY GREMS R. E. PORTER

LIKE many comparable groups, members of the computing facility at the Boeing Airplane Company feel that it takes too long to prepare a problem for a digital computing machine. The daily repetition of effort expended in outlining a problem for coding, the tedious task of coding the instructions, and the time consumed in checking-out or "debugging" the instructions all emphasize this fact. In this jet age, it is vital to shorten the time from the definition of a problem to its solution.

A new plan of attack for problem setup is necessary to shorten the elapsed time by shifting more of the monotonous burden of coding to the machine. It is a generally accepted belief that whenever rules for computing can be definitely established, they can be defined as a set of machine instructions. Therefore, the starting point for an automatic computing system is clarifying these rules to fit the requirements of a general problem.

A natural way to communicate a mathematical problem to a computer is by the written equation. This can be accomplished by a system allowing a digital computing machine to accept a problem directly in equation form together with a list of input data. The elapsed time for a problem is therefore shortened because this system eliminates the tedious task of coding the machine instructions. The setup time for each problem is then more dependent on the complete understanding of the mathematics and the logic rather than on the physical characteristics of one special computer.

MANDALAY GREMS and R. E. PORTER are with the Boeing Airplane Company, Seattle, Wash.

The BACAIC System

The Boeing Airplane Company Algebraic Interpretive Computing System, commonly called BACAIC, is a means of communicating directly with a machine. It is a self-contained system for solving a mathematical problem on a digital computer. This problem must be of a type which can be completely described by a set of algebraic and logical expressions. A working record of the entire system, including a file of library subprograms, is kept on magnetic tape. The library is made up of pieces originally constructed in a consistent fashion. This is important in order to establish a general pattern of rules for a system to follow.

The integrated system performs two distinct functions for each problem:

Table I. Definition of Symbols

	Symbol	Use	Explanation
Data reference	A–Z	$A+B$	Refer to all parameters by the letters A through Z, (except K)
	$K1$–$K99$	$K1+B$	Refer to all constants by a K-number
	1–50	$1+B$	Refer to the value (computed or estimated) of an expression by its expression number
Mathematical operations	+	$X+Y$	Addition
	−	$X-Y$	Subtraction
	·	$X \cdot Y$	Multiplication
	/	X/Y	Division
	PWR	X PWR N	$(X)^N$, the quantity X raised to the power N
	SRT	SRT X	$\sqrt{X}$, the square root of the quantity X
	SQR	SQR$[X+Y]$	The quantity following this symbol is squared
Transcendental functions	SIN	SIN A	Sine of angle A. A is in radians
	COS	COS A	Cosine of angle A. A is in radians
	ASN	ASN A	Arcsine A, the angle is in radians
	ACN	ACN A	Arccosine A, the angle is in radians
	EXP	EXP X	$(e)^x$, exponential to the X
	LOG	LOG X	The natural logarithm of X
Logical control	[or $	$[A+B$	Front bracket for a term
	] or ,	$A+B]$	Back bracket for a term
	*	$A-B*Y$	A substitution symbol. Compute the quantity on the left side of the symbol,* and substitute it for the parameter, constant or expression number on the right side of the symbol
	TRN	TRN 8	Transfer to execute expression number 8
	WHN, GRT	WBN A GRT B USE 8.	When the value for A is $\geq$ the value for B, compute expression number 8 next. Otherwise, compute the following consecutive expression
	USE, LES	WBN A LES B USE 8.	When the value for A is $\leq$ the value for B, compute expression number 8 next. Otherwise, compute the following consecutive expression
Modification of data	MOD LIM	MOD $H+B$ LIM R..	Modify the value for H by adding the increment B to H to form a new H. The operation symbols +, −, ·, and / can be used with the increment. Test this new value for H against the limit R. If the limit is exceeded, additional input data for the next case are read by the card reader at the appropriate time. If the limit is not exceeded, the reading of input data is by-passed and the next case is computed using this new H value of input. This procedure is a means for computing families of cases of data when one value of input is repeatedly altered by a preset amount. If more than one "Modify and Limit" expression is tested, the last LIM tested is the effective one
Table look-up and interpolation	ARG TBL*	ARG X TBL $K2$*Y...	To find $Y=f(x)$. The number in $K2$ is the number of the table to investigate. The tables are consecutively numbered as they are read by the card reader and stored in memory. The selected table is scanned and the corresponding linearly interpolated value for the argument X is computed. This value is substituted for Y
Selection of results	TAB	TAB A $K1$ 3 29 T...	Select the values for the indicated data symbols and store the values on a tape for later printing. Multiple TAB expressions are allowed with a maximum of six symbols per card. When the computing is finished for all cases of data, the stored values are printed. Data for all cases for one TAB card are printed prior to any printing for the next following card. The data are printed as decimal numbers and in the same order as indicated on the card. TAB cards always immediately follow the final equation or control expression
	PCH	PCH B 42 A $K19$....	The PCH expression is similar to the TAB expression, except that the stored data values are punched on cards as decimal numbers rather than printed as columns, of data. PCH cards always immediately follow the final TAB card (when TAB is used) or the final equation or control expression

1. It reads the algebraic expressions and translates them to machine language.

2. It computes results from given data, using the coded machine language instructions.

The choice of one of these functions is selected manually by the machine operator through controls on the console panel. This choice causes certain portions of the system to be operative and other portions to be by-passed.

The algebraic equations and logical controls which describe a problem are punched directly on cards to be read by the machine. These expressions are then translated by the computer to machine language instructions. The resulting machine instructions are automatically punched in binary form on cards. The time required for translating and machine-coding a problem usually averages 2 to 5 minutes; e.g., 10 expressions require about 2 minutes and 50 expressions require about 5 minutes.

The machine-coded instruction cards, accompanied by a set of given values for input data, are fed to the computer whenever computing is to take place. The results of the computing for one set of input data is printed (or stored for later printing) before another set of given values for input data is read. Computing of results for one problem continues for all sets of input data which are ready in the machine. The computing time per problem is dependent on the number of sets of given data and on the complexity of the computing pattern. The computing time usually ranges from a few seconds to 1 minute for each data case.

The algebraic equations and controls for a problem are written in terms of familiar symbols for reference to data values, mathematical operations, transcendental functions, logical control, table look-up and interpolation, systematic modification of data, and selection of results for printing or punching. The mathematical symbols such as +, −, ·, /, SIN, COS, LOG, and EXP are familiar to most people and an endeavor is made to assign mnemonic symbols to other operations.

Definition of Symbols

The mnemonic symbols for writing the expressions are grouped as shown in Table I.

Input Data Form

The input data for a problem are prepared in the same manner as for desk computing; i.e., a list of the values in terms of a reference symbol, a coefficient with a decimal point, and a possible power of 10 for this coefficient. When

the power 10 is zero, the zero is omitted, e.g.

$A=3.75$

$W=0.00375\times10^{3}$

$M=375\times10^{-2}$

Preparing the data in this manner provides the opportunity for entering items of data in either a floating or stated system of notation, and eliminates the necessity for changing each item of data to a preset notation system. Once the power of 10 is established for each value of data in the system, it is automatically adjusted for all operations performed on that value.

PROBLEM SETUP

Example 1. To illustrate the ease of preparing a problem for the BACAIC system, evaluate

$$Y=e^{-x^2}\sin CX \qquad (1)$$

for values of X from -0.99 to 1.00 in intervals of 0.01, and tabulate the corresponding values for X and Y.

This problem is written as three expressions for BACAIC:

1. MOD $X+K1$ LIM $K2$
   Modify a value for X
2. EXP $[K3-X\cdot X]\cdot$SIN $[C\cdot X]*Y$
   Compute Y
3. TAB X Y Tabulate X and Y

Each expression is punched on a card and the three cards are read by the machine. These expressions are machine-coded by the BACAIC system and a resulting set of instructions is punched on cards by the system. These instruction cards are fed to the machine with the following values of input data:

$C=5.0$ $K1=0.1$ $K3=0.0$
$X=-1.0$ $K2=0.99$

The expressions for this problem are executed consecutively in the foregoing order unless otherwise indicated. The machine accepts the data and repeatedly computes values for X and Y until the limiting value for X is exceeded. When the computing is finished, the 199 sets of values for the X and Y results are tabulated.

Example 2. Compute both roots for multiple values of C with constant values for A and B.

$$AX^2+BX+C=0 \qquad (2)$$

To solve for both roots, rewrite the equations as follows:

$$\frac{-B+\sqrt{B^2-4AC}}{2A}=X$$

$$\frac{-B-\sqrt{B^2-4AC}}{2A}=Y$$

To illustrate a comparison and selection, assume the following conditions:

When the discriminant (B^2-4AC) is positive, use its true value.

When the discriminant (B^2-4AC) is negative, use a value of zero.

This problem is written as five expressions for BACAIC:

1. $B\cdot B-K4\cdot A\cdot C$
   Evaluate discriminant
2. WHN 1 GRT $K5$ USE 4
   Compare values and select
3. $K5*1$
   Substitute zero
4. $[K1\cdot B+$SRT 1$]/[K2\cdot A]*X$
   Compute X
5. $[K1\cdot B-$SRT 1$]/[K2\cdot A]*Y$
   Compute Y

The five expressions are machine-coded by the system. The punched instruction cards are fed to the machine with the given input data.
The input data are:

$B=6.0$ $K4=4.0$ $K1=-1.0$
$A=1.0$ $K5=0$ $K2=2.0$

(Case 1) $C=5.0$
(Case 2) $C=4.5$
(Case 3) $C=9.0$
(Case 4) $C=18.0$

The machine accepts the data for case 1, computes a result for each expression, and prints these five results for case 1. The machine then reads the second value for C, computes each result and prints the five results for case 2. This procedure continues for the four given values of C.

The two illustrated examples demonstrate the general plan for writing the expressions where each expression is punched on an individual card. The examples also demonstrate the difference between selective printing of results and the printing of all results for each case. There is a noticeable difference in the printing time for the two methods. This factor should be considered at setup time, as the needs of the problem or the needs of the programmer determine the type of printing.

Criteria for Coding

It may be asked how a machine can consistently interpret and translate the algebraic equations so quickly and so accurately. This idea is plausible when it is accepted that a set of rules for the machine in its own language is sufficient for translating. These rules must be definite and exact for all situations. The BACAIC system now appears straightforward and relatively simple. The present system differs considerably from the original plan, as the former includes more details and special features.

In order to establish an over-all plan for interpreting the equations directly from the cards, many decisions for writing the equations and controls had to be formulated. Some of these decisions were mandatory as they depend on the particular computer in use. The BACAIC system was written especially for the International Business Machines Corporation (IBM) Model *701*. However, much of the planning and organizing of the system can easily be transferred to another digital computer. The reader of the IBM *701* reads a maximum of 72 uppercase letters, numerals, and symbols. This dictates that one level of punching or printing is recognized by the machine, thereby eliminating the possibility of punching or printing subscripts or superscripts in the familiar way. This limitation is easily overcome by an appropriate symbol to signify the operation or meaning to the machine.

Some of the early decisions depended entirely on the anticipated types of problems to be studied and the characteristics of their data. The question of floating point arithmetic versus stated point arithmetic arose with stronger arguments in favor of the floating point system. In the floating point system, the elimination of the problem of scaling values of input data is very satisfying. The use of this arithmetical system for BACAIC is proving to be an attractive feature for inexperienced personnel. The rules for machine computing in the floating point system were firmly established at an earlier time when the library subprograms were written. These library subprograms for floating point arithmetic were incorporated in the system and the rules governing them were accepted unchanged. Fortunately, a standard pattern for the input-output to these library subprograms had been adhered to and was readily adaptable to a system.

One of the next questions to be solved concerned the values for constants and data. If the actual values of data are included in the expressions, the digits of the numbers occupy too many of the 72 available card columns, so a scheme for referring to all data by symbols was developed. The values for the corresponding symbols are entered at computing time. This scheme has the added feature of making it very convenient to alter values without rewriting the expres-

sions. Originally, the 25 alphabetic letters A through Z (except K) and the 99 K's ($K1$-$K99$) seemed sufficient for data reference, but this is proving to be inadequate for some problems. The numbers 1–50 are data reference symbols for the values of the corresponding expression results. These values are estimated values for the expressions or computed values for the expressions. A reference of this type provides a simple means for using a computed result for one expression as an input value for another expression. In the second expression of example 2

WHN 1 GRT $K5$ USE 4

the 1 refers to the result of the first expression; i.e., the value of the discriminant (B^2-4AC). This reference is especially convenient in a problem when an estimated value of a result is needed to start the computing, but after the first computation, the symbol refers to the most recently computed result.

Many mathematical problems require a choice of operations at various levels of the solution. The designers of computing machines recognize the need to select and transfer, as they invariably include machine codes for "transfer on plus," "transfer on minus," or "transfer on zero." In an automatic system, this need for a conditional transfer is even more urgent. It is the only means for describing a problem as a complete picture when part of the picture is dependent on a previous computation in the same problem. When this select and transfer feature is included in a system, problems dealing with iteration, integration, and progressive summation are easily manipulated. Without this feature, a system is very limited in its application.

The foregoing information helps to outline a general plan for an automatic computing system. After these notions are settled and accepted, the rules for writing the expressions are considered with respect to the capabilities of the machine. The limitation of any computing machine is that it executes *exactly* all instructions which it receives and it remembers *only* the information it is told to remember.

The mnemonic symbols for certain operations are readily recognized and accepted as three adjacent letters, such as SIN, TAB, LOG. This starts a pattern for mnemonic symbols for all operations, and recognition for the exact symbols is easier when the first two letters of a symbol are not the same as the first two letters of another symbol.

The use of parentheses for the grouping of terms within terms is very essential when writing equations. It is natural to use parentheses or brackets in equations for the purpose of grouping terms to be used as one operation; e.g., SIN $(A+B+C)$. It is necessary to close all bracketed groups; i.e., the brackets must travel in pairs. Therefore, a separate symbol is needed for the front bracket and a separate symbol is needed for the back bracket. This ability must be available in an automatic system, and from experience must be increased in an automatic system to include equivocal situations. In the second expression of Example 1, the sine term is coded as "SIN $[C\cdot X]$." This removes the doubtful meaning for

the sine of C to be multiplied by X

or

the sine of the product, C multiplied by X

Without the ability to group operations, a system is extremely limited in its usefulness. It is a toy and not a tool for computing.

The arithmetic operation for division is another stumbling block to a smooth system. The division concept presents a few difficulties, as up to this time all operations are assumed to be in the numerator. Obviously, an exception to the rule is necessary. The revised rule for writing expressions states that all operations are in the numerator except those following a division symbol. Then, only the symbol or bracketed term immediately following the division symbol is in the denominator. This practice is successful and is relatively simple to contend with for all situations. This rule is demonstrated in the fourth and fifth expressions for example 2, where the numerator is divided by a product.

Interpreting an Expression

The ability of a machine program to analyze a given algebraic expression and determine the unambiguous sequence of computations intended by the originator of the expression is subject both to the natural rules of algebra and to the restrictions imposed by the machine programmer. Certain restrictions result in the consistency so vital to machine programs yet impose no hardship upon the person writing an expression; e.g., the substitution of 3-letter mnemonic codes such as SIN, COS, and SQR for sine, cosine, and square. This makes machine decoding much simpler without detracting from the natural appearance of the expression. Any restrictions on the use of arithmetic symbols or the grouping of terms are more difficult to justify. The number of permissible symbols and the length of any one algebraic expression is usually influenced by the data input and internal storage capabilities of the machine used. It is in the best interests of the machine program's users to concede everything to the naturalness of writing an expression. Only the limit of the programmer's ingenuity dictates the restrictions which need apply.

The principal problem in interpreting an expression is that of defining the rules which the machine must follow to produce an unambiguous operating sequence.

A few of the contingencies encountered are illustrated in the following examples:

Example 3.

$$a \sin b+\frac{c}{d}-x+y \qquad (3)$$

Example 4,

$$ay+\left\{q+(n^z-r)(a+b)+\frac{c-d}{\frac{d}{x}+q}\right\}\text{SIN } v \qquad (4)$$

The first contingency is the "understood multiplication" illustrated in the terms "$a \sin b$" or "ay." This type of operation was eliminated from BACAIC expressions by making it illegal (the simplest way out of any coding dilemma). The rule that all arithmetic operations must be indicated by the appropriate symbol simplifies the initial translation step. It is possible to have the machine itself supply the understood operation symbols at the cost of extra programming.

The next contingency is that of having a choice as to which operation to perform first. This choice can neither be eliminated by a rule nor left to the discretion of the machine. A human computer has a choice of either of two operations when starting to compute the result of example 3. He may divide c by d or compute the sine of b. Five such choices are possible in example 4. These choices cannot be left to a machine. Instead, a way must be determined of defining an order of operations having no chance of duplication or ambiguity during machine interpretation.

The normal rule of algebra that all multiplication and division must be performed before terms are combined is only a partial answer to the problem. In example 3, the function operation "sine b" must be performed before it can be multiplied by "a" and this multiplication must be performed before the entire term (a sin b) can be added to the quotient of c divided by d. Possible ambiguities in operation sequences may be avoided by combining the normal rules of algebra

Table II. Right Operand Condition Table

Operation Requiring Operand	Adjacent Right Item	Item Following Right Item	Right Operand Corresponding to the Stated Conditions
A function such as SIN, COS, LOG	An item symbol such as A, B, $K15$	or /, + or −,], special expression termination symbols	The item indicated by A, B, $K15$
	A group of terms indicated by [ ]	or /, + or −,], special expression termination symbols	The result of the last operation performed within the brackets
Multiply or divide (· or /)	An item symbol such as A, B, $K15$	or /, + or −,], special expression termination symbols	The item indicated by A, B, $K15$
	A group of terms indicated by [ ]	or /, + or −,], special expression termination symbols	The result of the last operation performed within the brackets
	A function such as SIN, COS, LOG	Irrelevant	The result of the function operation
Add or subtract (+ or −)	An item symbol such as A, B, $K15$	· or /	The result of the last multiplication, division or function operation performed before the next add or subtract operation is encountered
		+ or −,], special expression termination symbols	The item indicated by A, B, $K15$
	A group of terms indicated by [ ]	· or /	The result of the last multiplication, division or function operation performed before the next add or subtract operation is encountered
		+ or −,], special expression termination symbols	The result of the last operation performed within the brackets
	A function such as SIN, COS, LOG	Irrelevant	The result of the function operation

Note 1. All other sequences of symbols are violations of expression writing rules.

Note 2. All operation symbols in the following right items must be in the same group as that of the operation requiring a right operand.

with the rule that operations are performed in the order they are encountered in the expression from left to right. The resulting combination is specifically stated by the following rules.

RULES FOR DETERMINING OPERATION SEQUENCE

1. Scan the expression from left to right assigning ascending operation sequence numbers to every function operation; e.g., sin b in example 3.

2. Rescan the expression from left to right continuing the assignment of ascending operation sequence numbers to every multiplication or division symbol; e.g., $a \cdot \sin b$ and c/d in example 3.

3. Again rescan the expression from left to right continuing the assignment of ascending operation sequence numbers to every addition or subtraction symbol; e.g., $a \sin b + c/d$ and that result $-x$, etc., in example 3.

The three foregoing rules are easily programmed and permit the machine to choose automatically an unambiguous sequence of operations for an algebraic expression.

Example 5.

$$\begin{matrix} ②① & ④ & ③ & ⑤ & ⑥ \\ A \cdot \text{SIN}\ B + C/D - X + Y \end{matrix} \quad \left\{ \begin{matrix} \text{Operation} \\ \text{Sequence} \\ \text{Numbers} \end{matrix} \right. \tag{5}$$

This example shows the original expression of example 3 in BACAIC form with the sequence of operations indicated above the operation symbols.

Example 6

$$\begin{matrix} ⑫ & ⑭ & ⑨ & & ① & & ③ & ⑦ & & ④ & & ⑩ & & ⑤ & & ⑧ & ② & ⑥ & & ⑬⑪ \\ A \cdot Y + [Q + [N\text{PWR}Z - R] \cdot [A + B] + [C - D]/[D/X + Q]] \cdot \text{SIN}\ V \\ & & & & 2 & & & & & 2 & & & & 2 & & & & 2 & & & \\ & & & & & & & & & & 1 & & & & & & & & & \\ & & & & & & & & & & & & & & & & & & & \\ & & & & & & & & & & 0 & & & & & & & & & \end{matrix} \tag{6}$$

Examples 3 and 5 ignore the problem which arises when operations are grouped by parenthesis or "bracket" symbols (see example 4). This grouping of operations is very necessary to the writer of an algebraic expression. It is essentially a mathematical shorthand notation which permits him to specify the general order in which he desires computations performed. Since the human computer handles these groups of terms by working "from the inside out," a machine must do the same. This is accomplished by the assignment of a "group number" to every significant symbol in the expression in accordance with the following rules.

RULES FOR GROUP NUMBER ASSIGNMENT

1. Scan all significant items of the expression from left to right assigning the same group number to each item until a left (front) bracket symbol is encountered. (Note: If no left bracket is present in an expression, all items will have the same group number.)

2. When a left (front) bracket symbol is encountered, increase the current group number by one and assign this increased number to that bracket symbol and to all successive items until another bracket symbol is encountered.

3. When a right (back) bracket symbol is encountered, assign the current group number to that bracket symbol and then decrease the current group number by one, assigning this decreased count to all successive items until either another bracket symbol or the end of the expression is reached. (Note: All groups must be completely enclosed; e.g., there must be an equal number of left and right brackets.)

The machine system is programmed to work "from the inside out" by first assigning group numbers to all expression items in accordance with the preceding rules, second determining the maximum group number and applying the "rules for determining operation sequence" to that group, third decreasing that number by one and reapplying the operation sequencing rules to this next group, etc., until all groups have received their operation sequence numbers.

Example 6 shows the original expression of example 4 in BACAIC form with the group numbers indicated below each group and the corresponding sequence numbers indicated above each operation symbol.

Observe that the group number distinguishes a level of grouping rather than a particular group; e.g., there are several group 2's in example 6. An examination of example 6 also reveals that although the operations are performed in a seemingly heterogeneous manner, the operand needed by each operation is calculated in time to permit an uninterrupted sequence of operations.

After the operation sequence is defined, the final problem is defining how the machine is to find the proper operand or operands for each operation. The most

common type of operation requires two operands, that is, a quantity both to the left and to the right of the operation symbol. Other operations such as the sine function require only one operand which is normally written to the right of the operation symbol. Therefore, as the machine examines each operation symbol, it must have a means of distinguishing those operations requiring a single operand from those requiring both a left and a right operand.

Examples 5 and 6 indicate that either operand may be the result of a previous calculation rather than an item symbolized in the original expression. They also indicate results of previous operations are not necessarily used in the next operation. These facts require that the result of every operation be stored separately within the machine for use at any later time while computing that expression. In other words, the computing sequence is such that the result of an operation cannot automatically become one of the operands for the following operation. The use of brackets in an expression requires that either operand may be the result of a group of operations as well as a single quantity or previous result. A summary of the conditions governing the selection of a right operand for the various operations and the corresponding expression context is given in Table II. A similar summary for the selection of a left operand is given in Table III.

The previously assigned group numbers and operation sequence numbers are used in the selection of operands to meet the conditions summarized in Tables II and III. Application of the following rules by the machine enables it to select the proper right operand for each indicated operation.

Table III. Left Operand Condition Table

Left Operand Corresponding to the Stated Conditions	Item Preceding Left Item	Adjacent Left Item	Operation Requiring Operand
The result of the function operation	A function such as SIN, COS, LOG	An item symbol such as A, B, $K15$	A function such as PWR, GRT, LES
The item indicated by A, B, $K15$	or /, + or −, [, start of expression		
The result of the function operation	A function such as SIN, COS, LOG	A group of items indicated by [ ]	
The result of the last operation performed within the brackets	or /, + or −, [, start of expression		
The result of the last multiplication, division or function operation performed after the first preceding add, subtract, [symbol or the start of the expression is encountered	A function such as SIN, COS, LOG, or . or /	An item symbol such as A, B, $K15$	Multiply or divide (. or /)
The item indicated by A, B, $K15$	+ or −, [, start of expression		
The result of the last multiplication, division or function operation performed after the first preceding add, subtract, [symbol or the start of the expression is encountered	A function such as SIN, COS, LOG, or . or /	A group of items indicated by [ ]	
The result of the last operation performed within the brackets	+ or −, [, start of expression		
The result of the last preceding addition or subtraction operation after the [symbol or the start of the expression is encountered	A function such as SIN, COS, LOG, or . or / or + or −	An item symbol such as A, B, $K15$	Add or subtract (+ or −)
The item indicated by A, B, $K15$	[, start of expression		
The result of the last preceding addition or subtraction operation after the [symbol or the start of the expression is encountered	A function such as SIN, COS, LOG, or . or / or + or −	A group of items indicated by [ ]	
The result of the last operation performed within brackets	[, start of expression		

Note 1. All other sequences of symbols are violations of expression writing rules

Note 2 All operation symbols in the preceding left items must be in the same group as that of the operation requiring a left operand.

Rules for Determining Right Operand

1. Beginning at the operation requiring a right operand, scan all expression items to its right having group numbers equal to or greater than that of the operation itself. Record the maximum operation sequence number encountered before:

(a) An operation sequence number, with the same group number, is encountered which is greater than that of the original operation sequence number, or

(b) A right (back) bracket symbol with a group number equal to that of the original operation's group number is encountered, or

(c) An item having a group number less than that of the operation itself is encountered, or

(d) The end of the expression is reached.

2. When no operation sequence number is recorded prior to meeting conditions 1(a), 1(b), 1(c), or 1(d), the item immediately to the right of the original operation is its proper right operand.

3. When an operation sequence number is recorded prior to meeting condition 1(a), 1(b), 1(c), or 1(d), the result corresponding to the maximum operation sequence number recorded is the proper right operand.

Similarly, the machine selects left operands for each indicated operation which requires one by applying the following rules.

Rules for Determining Left Operand

1. Beginning at the operation requiring a left operand, scan all expression items to its left having group numbers equal to or greater than that of the operation itself. Record the maximum operation sequence number encountered before:

(a) An operation sequence number, with the same group number, is encountered which is greater than that of the original operation sequence number, or

(b) A left (front) bracket symbol with a group number equal to that of the original operation's group number is encountered, or

(c) An item having a group number less than that of the operation itself is encountered, or

(d) The start of the expression is reached.

2. When no operation sequence number is recorded prior to meeting condition 1(a), 1(b), 1(c), or 1(d), the item immediately to the left of the original operation is its proper left operand.

3. When an operation sequence number is recorded prior to meeting condition 1(a), 1(b), 1(c), or 1(d), the result corresponding to the maximum operation sequence number recorded is the proper left operand.

The machine system determines the operation sequence and the corresponding operands, and records its findings in a sequence table. A 3-address operation sequence table is a familiar way of recording such information. Table IV illustrates the BACAIC Operation Sequence Table for the expression given in example 3.

Expression Interpretation Rules

The steps involved in the machine interpretation of an algebraic expression are summarized in the following rules.

1. Scan the expression's characters classifying them into operation, operand, expression result, grouping, computation control, and expression termination symbols.

2. While performing the classification, eliminate all extraneous spaces and mnemonic characters and fill in appropriate items for all "understood" symbols.

3. Assign group counts to all expression items.

Table IV. Operation Sequence Table

Seq. No.	Right Operand	Operation	Left Operand	Result
1	None	Sine	B	1
2	A	Multiply	1	2
3	C	Divide	D	3
4	2	Add	3	4
5	4	Subtract	X	5
6	5	Add	Y	6

Note 1. The contents of this table is given symbolically rather than in machine codes and storage location addresses.

Note 2. The result for the entire expression corresponds to that for the maximum sequence number in the entire expression; e.g., 6

4. Assign operation sequence numbers in accordance with these groupings and the sequence determination rules.

5. Determine the operands corresponding to each operation.

6. Record the operating data in a form from which actual machine instruction sequences may be assembled.

Translating a Sequence Table to Machine Instructions

Once a sequence table is prepared, a few more specific decisions are necessary before it can be translated to machine instructions. Probably the most important is that concerning the storage of the values of input data. A need for a convenient method of reference to either an address of a data value in the sequence table or an actual location in storage is evident. This need is handled by reserving an area in storage for values of data. This concept is similar to the need for boxes at a post office. In this reserved area, one box or location is set aside for each data reference symbol, i.e., (*A*-*Z*), (*K*1–*K*99) and (1–50).

Each box originally contains zero, and remains at zero until a value is placed in it. A value can be entered in each box either as an item of input data or as a computational result. The current value in any box is the only value available at any time.

Another decision is whether the entire contents of the master tape or only the coded machine instruction deck is available at computing time. When the entire tape is available, the machine is able to print comments and other information appropriate to any situation.

A minor detail (one which is probably assumed to be a fact) is the packing of high-speed storage in a unique fashion for each different problem. This utilizes the storage more advantageously and requires less reading of records from the master tape.

The procedure involved in translating each sequence table to machine instructions is as follows. The sequence tables are scanned for the mathematical operation codes. Each different code is recorded once and a complete list made of all the operation codes referred to in the expressions for one problem. This list of codes is incorporated in an index of information for library subprograms. A location is assigned to each required subprogram and this assigned location is stored in the index. When the location assignments are finished, this index is punched on cards. These cards are used during computing time by a relocation program to pack the specified library subprograms in working storage. The information in this index is also used to insert the addresses for the machine instructions which are dependent on the actual location of each of the library subprograms.

This index is now discarded and full attention is directed toward each sequence table and the preparation of the corresponding machine instructions for that expression. The actual locations of the input data and of the result data for each operation code are taken from the sequence table. These are stored as addresses for certain machine instructions of the library subprogram for that operation code. These machine instructions containing the references to data and to subprograms are packed adjacent to similar machine instructions for the previous operation code in the same table. This procedure of storing data locations as addresses of instructions and then packing the instructions continues for each operation code of a sequence table. The complete set of machine instructions for the one expression is punched on cards in binary form to be used at computing time. The entire process is repeated for each subsequent table. When the punching for the last expression takes place, the automatic coding for the problem is finished.

Computing Procedure

The master tape is used during computing time as it retains the bulk of the system instructions. The coded binary cards containing the instructions for a problem are used repeatedly with varied values of input data. The instruction cards are fed to the machine together with heading cards for identifying the results. At the start of computing time, the working storage is filled with those portions of the system needed to prepare the machine for computing. The library subprograms are packed adjacent to one another in working storage. An area originally set to zero is reserved for values of input data. The input data is read as decimal numbers and stored in the area reserved for the corresponding symbols. Only one value is saved for any one symbol at a time. A new value merely replaces the old value for the same symbol. After these preliminary preparations are finished, a continuous cycle of machine action takes place. The computing always starts with the first expression and ends with the last expression. Normally, the expressions are executed consecutively but a TRN or USE symbol alters this normal routine. Loops for iteration or integration can be included between the first and last expression by means of the logical control symbols. All computing is performed in floating point arithmetic. The results are available for each case after executing the last expression for that case. Whenever an intermediate result of computing is needed, it must be written as a separate expression. This cycle of reading input data, computing results, and printing or storing result data is broken when the problem is finished or when some interruption of machine action occurs.

Computing Controls

The following computing controls are necessary to increase the over-all usefulness and flexibility of a computing system.

1. The choice of an expression to execute due to the result of a comparison.

2. The systematic modification of input data when it varies by regular intervals.

3. The selective printing of input data and computed results.

4. The selective punching of input data and computed results.

5. The printing of comments with pertinent information which describes errors or points out violations in usage of the library subprograms, the input data, the expressions, or other machine instructions.

6. The interruption of computing due to an emergency and the later restoration of data for continued computing from the point of interruption.

7. A combination of the last two features; i.e., the printing of comments, the interruption of the computing, and the later restoration of the data for continued computing.

An explanation and description of these controls clarifies the benefits which they add to an otherwise incomplete system.

Comparison and Selection

It is assumed that the expressions are written in the correct sequence for computing, and that this same physical sequence is maintained throughout the problem. In other words, a reference to expression number 1 is always the first expression of the written set; and a reference to expression number 2 is always the second expression of the written set.

Often, a comparison of two values or a selection of a specific expression is desirable at some definite point in the computing which upsets this normal sequence. Suppose that the following condition is necessary, "when the value for A is greater than the value for B, use equation number 6 to compute the value for C; otherwise, use the next equation to compute the value for C." This selection is written as follows:

WHN A GRT B USE 6

A comparison of this type is one of the logical controls which can be handled by the system in the same manner as a mathematical equation. Therefore, insert this logical control in its proper sequence with the set of expressions. Each of the values to be compared can be computed prior to the comparison in the same expression. For example,

WHN $[A+R-T]$ GRT [SIN $[X+Y]\cdot W$] USE 10

Modification of Data

Some mathematical problems are of the type similar to example 1, $Y=e^{-x^2}$ sin CX, where Y is evaluated for all values of X from -0.99 to $+1.00$ in intervals of 0.01. Similar situations frequently arise and it seems appropriate for the machine to prepare its next new value of input whenever possible. The symbolic expression for this data preparation is

MOD $X+K1$ LIM $K2$

The value for the increment $K1$ is added to the value for X and the sum replaces X. This new value for X is compared with the limiting value for $K2$. If X is less than $K2$, the input data reading routine is by-passed at the beginning of the next case. If X is greater than $K2$, cards for input data are read by the machine at the beginning of the next case. The arithmetic operation attached to the increment can be +, −, ·, or /. The data values can be positive or negative since the signs are tested to insure modification in the indicated direction.

Selective Printing

The BACAIC system did not initially include selective printing. This shortcoming was immediately realized when unnecessary printing of all results for a problem took place. This complete printing of intermediate results was confusing and difficult to explain to inexperienced personnel. It was also a needless waste of valuable machine time. The symbol chosen for selective printing is TAB. A reference to any data symbol is allowed with a maximum of six references per TAB. Each TAB symbol is written as a separate expression. The TAB cards follow the equation and control cards. When the computing is finished for one case of input data, the indicated values are selected and stored on a magnetic tape. For the following expression

TAB A $K5$ 7 R 19 42

the values for A, $K5$, result 7, R, result 19 and result 42 are selected and stored. The system then by-passes all printing routines at that time and continues to compute the next case of data. At the end of computing for a problem, the selected and stored data are listed. Multiple TAB expressions are permitted, but the printing for the first TAB is completed before any printing for the second TAB takes place.

Selective Punching

Selective punching satisfies the need for a form of output which can be used as direct input to another machine program. In the BACAIC system, the data values are selected and stored the same as for selective printing. However, unlike TAB, this is an additional function of the system and does not replace another function. When the computing is finished for a problem, the selected and stored data values are punched on cards as decimal numbers. A maximum of six data reference symbols is allowed per PCH code. The PCH expression cards follow all other expression cards.

Diagnostic Assistance

There are two legitimate types of machine stops when BACAIC is controlling the machine:

1. A STOP when the operation of the machine can be continued.
2. A STOP when the operation of the machine cannot be continued.

Each of these stops can be caused by keypunching errors, computing difficulties or machine malfunction. In order to distinguish which error caused the machine to stop, a "machine trail" is printed. This "machine trail" includes a pertinent comment to state the reason for stopping and to indicate corrective measures. It also includes the exact location in the memory unit of this unexecuted instruction, the number of the expression it was examining or computing, and the next instruction to execute after the corrective measures are accomplished. This last-mentioned transfer instruction is impor-

```
SAMPLE   03 EXPRESSIONS  SAMPLE PROBLEM FOR BACAIC

1   MOD  X  +K1  LIM  K2
2   EXP  $K3 - X.X , . SIN $C . X.  * Y
3   TAB  X  Y

         X VALUE  Y VALUE


THE ORIGINAL INPUT DATA FOR CASE NUMBER  1

+ C    5.0          X   -1.1          K1 0.2            K2  1.0
1

 THE SELECTED RESULTS ARE LISTED AS  FOLLOWS.

     X VALUE          Y VALUE
    .90000000-  8-   43486215   8-
     70000000-  8-   21489906   8-
     50000000-  8-   46609057-  8-
     30000000-  8-   91164176-  8-
    100000000-  9-   47465517-  8-
    100000000   9-   47465517   8-
     30000000   8-   91164176   8-
     50000000   8-   46609057   8-
     70000000   8-   21489906-  8-
     90000000   8-   43486215-  8-
    110000000   8-   21039020-  8-

THE COMPUTING IS COMPLETED FOR ALL CASES OF DATA ENTERED IN THE MACHINE.
```

Fig. 1. A solution of a problem by the BACAIC system

tant if the computing can be continued from that point. All of these are aids to locating an error or discrepancy and to provide a written record for future reference or study of the expressions and the program. An example of a "machine trail" is:

```
54      TWO ADJACENT OPERATION CODES IN AN EXPRESSION
        DECIMAL NUMBERS                 OCTAL LOCATIONS
CONTROL PROG.    EXP. NUMBER      STOPPED AT      TRANSFER TO
      13               4             3752            1654
```

Interruption of Computing

Occasionally, the computation for multiple cases of data must be interrupted before the computing is finished for all the cases. The computing can be carried on at a later time if the values for the parameters, the constants, and the results in the memory can be restored to the identical values at the time of the interruption. When an interruption is necessary, a SENSE switch is turned ON while the machine is computing. The results for the current case of data are computed and printed or stored. In those instances where selected values are stored for later printing or punching, this printing or punching of the accumulated data also takes place. The afore-mentioned pertinent data are punched in binary cards. These same data are printed to be used as a reference for the purpose of cross-checking the data and results. When the time arrives to continue the computation, these binary cards are fed to the machine prior to the decimal input data for the unfinished cases. A console SENSE switch is turned ON which controls the reading of binary data cards and storing them in the memory unit prior to computing the first expression. The contents of the memory unit are hereby restored to the identical values at the time of the interruption. The computing is then carried on as if no break had occurred.

Interruption after Diagnostic Assistance

Sometimes a violation of a computing rule for a library subprogram is caused by an incorrect value of data. When this value is an incorrect input value, it probably is sufficient to note the error in the data, to print the current results at that computing point, and to start computing for the next case of data. Zero values are stored for that case of result values in problems which include a TAB or PCH expression. This prevents the possibility of printing or punching erroneous results unwittingly.

When the incorrect value encountered is one which was developed in the computing, an attempt to determine the cause of the error is recommended. The interruption control is activated by turning on a console SENSE switch. This causes all values in the reserved area to be printed as decimal numbers and to be punched as binary data. All data previously stored for the TAB or PCH are handled the same as for the end of computing. It is assumed that the cause of the error can be detected after examining these printer decimal data. The binary values for these data are fed to the machine at a later time so that computing continues from the beginning of the next case. An example of an error which causes the machine to stop is shown in the following:

```
11  FOO7  THE ARGUMENT IS TOO LARGE.  ADD MORE VALUES TO THE TABLE.
        DECIMAL NUMBERS                 OCTAL LOCATIONS
CONTROL PROG.    EXP. NUMBER      STOPPED AT      TRANSFER TO
      31               5             3222            5136
13 CASE RESULTS WRONG.  PUSH START FOR NEXT CASE, OR SENSE 1 FOR INTERRUPT
        DECIMAL NUMBERS                 OCTAL LOCATIONS
CONTROL PROG.    EXP. NUMBER      STOPPED AT      TRANSFER TO
      31               5             6362            5136
```

Computing Features

The addition of various extra features contributes to the operating smoothness of any computing system. The ability directly to include empirical or other functions resulting from test result correlation minimizes mathematical curve fitting and hence elapsed setup time. Complete machine identification of results saves clerical time and reduces errors resulting from misinterpretation of unidentified data. The originator of a problem needs assurance that the machine interprets his problem correctly. This is accomplished by the system comparing machine results with the results anticipated by him. Other features can be added as the need arises to expand a system.

Table Look-Up and Interpolation

A table look-up routine is needed to satisfy all those conditions of data which cannot be easily expressed by equations. In many instances, this set of table data is prepared more quickly than one equation or multiple equations for the data. Tables can be altered from one computing time to the next when table data are part of the input data rather than part of the expressions.

In cases where empirical data are used for parts of the computing, it is often advisable to resort to a table look-up and interpolation routine. At the present time, only linear interpolation is available in the BACAIC system.

The values for the table look-up routine enter the machine in a manner similar to entering values of regular input data, except that the pairs of table data are stored consecutively. The first item of each pair of values must be in consecutive ascending or descending order. The maximum size for each table is arbitrarily limited to 400 half words or 100 pairs of values in the BACAIC system. The tables are stored as consecutive records on a magnetic drum whose limit of 4096 half words is also the limit of half words for all tables. The system prepares an index for locating each table whenever it is needed.

Suppose that the first table on the drum is an X, Y, table such as:

X	Y
2.0	20.
3.0	30.
5.0	50.

Find the corresponding value for Y when $X=3.72$ and $K3=1.0$. The expression for BACAIC is as follows:

ARG X TBL $K3*Y$

The value stored in $K3$ is the number of the table used to find Y. This expression is interpreted to read "look up the argument X in the first table and substitute this value for Y." Incidentally, the argument can be computed prior to the lookup routine in the same expression. For example,

ARG[$X+Y-$SRT[SQR SIN $A+$SQR COS A]]TBL $K1*Y$

Identification of Result Values

Up to this time, little attention has been paid to identifying the quantities to be computed for each problem. During the

```
JOB 7959 38 EXPRESSIONS  DATA REDUCTION - FR 205  FOR WIND TUNNEL

          1    $K13 - K17,/K18
          2    K13/$K13 - K17,
          3    $K13 + K17,/$K18 . $K13 - K17,,
          4    $K17 + 1 . SQR E, PWR 2
          5    F/4
          6    K3 . $M - N,
          7    WHN 6 GRT K16 USE 9
          8    K16 - 6 * 6
          9    $J + K2 . $M - N,, . $K1 + 6,
         10    9 + $N - 5, . K9 + $M - 5, . K8 + $P - 5, . K7
         11    SRT$$$H/1, PWR$K17/2, - K17,/1,
         12    K17 + 1 . SQR 11
         13    12 PWR 3
         14    SRT$G + K14,
         15    SRT$K13 . K11/K12,
         16    K4 . K5 . 15 . H . 11/$14 . 13,
         17    K17 + 1 . SQR E
         18    17/12
         19    18 PWR 3
         20    K4 . K5/K6 . H/F . 11/E . 19
         21    $K18/$K13 + K17,, PWR 3
         22    K10 - K7
         23    21 . 15 . 22 . L/16/14
         24    K22/$23.K21,
         25    SQR24.SQR$SQR24, + K19.SQR$SQR24, + K20.SQR24 - K21.23.24 + K22 * Q
         26    WHN 25 GRT K16 USE 28
         27    K16 - Q  * Q
         28    WHN Q  LES K26 USE 31
         29    24 - 25/$K23.SQR$SQR24,.24 + K24.24.SQR24 + K25.24 - K21.23, * 24
         30    TRN 25
         31    K17 + 1 . SQR 24
         32    31 PWR 2
         33    L/32
         34    K17 + K13 . SQR 24
         35    K13 . SQR E . 20 . K6
         36    K13/K18 . 5 . SQR E
01266    37     10  + 33.34.22 - 5.$22 + 35,
         38    37/$36.K6,

         1        2        3        4        P0       6
         TEST     8        9        FG       MN       12
         13       14       15       16       17       18
         19       AO/AL    21       A2 - ACB 23       24
         25       26       27       28       29       30
         31       32       P2E      34       35       QO
         D INLET  CD INLET


         THE ORIGINAL INPUT DATA FOR CASE NUMBER  1

           K1 .04           K2 23.39         K3 0.0           K4 0.983
           K5 6.1575        K6 4.891         K7 0.3849        K8 1.3902
           K9  2.9732       K10  6.1575      K11  32.174     K12 53.345
           K13   1.4        K14  459.0       K15   0.2        K16   0.0
           K17   1.0        K18   2.0        K19   15.0       K20    75.0
           K21  216.0       K22  125.0       K23    6.0       K24   60.0
           K25  150.0       K26   1.0    -4  A    114.0       B     20.0
           C  1.0           D  7025.0        E  1.99          F  14.76
           G  110.          H  8.05          I  6.00          J  -288.0
         + L  9.99          M  6.80          N  3.30          P  7.41

           CASE NUMBER    1.      THE COMPUTED RESULTS

         1              2              3              4              P0             6
01266     20000000  8-   35000000  7-   30000000  7-   77037089  7-   19159602  7-

         TEST           8              9              FG             MN             12
01266          9                        82454000-  7-   47740751  7-   66182211  8-  108760170  8-


01266    13             14             15             16             17             18
01266     12864995  7-   23853721  6-   91890304  8-   96560392  8-   17920200  7-   16476804  7-

         19             AO/AL          21             A2 - ACB       23             24
01266     44732052  7-  100409966  8-   57870380  8-   57726000  7-  133139777  8-   50446148  8-

         25             26             27             28             29             30
01266                          28                             31        50446148  8-          25

         31             32             P2E            34             35             QO
01266    105089628  8-   11897598  7-   83966528  7-   13562739  7-   27227557  6-   53111758  7-

         D INLET        CD INLET
01266     72863860  7-   28049417  8-


         THE COMPUTING IS COMPLETED FOR ALL CASES OF DATA ENTERED IN THE MACHINE.
```

Fig. 2 Actual data reduction problem and results

computing of a problem, the only printed information other than the given expressions and the input data is the computed results. When each result is printed, it is identified by a corresponding result column heading.

The quantities to be computed are not the same for all problems, therefore, the result column headings are not the same for all problems. These headings must be introduced individually for each problem. They are separate from the expressions which they identify so that they do not interfere with the mathematical symbols and abbreviations and are not interpreted. These heading cards are fed to the machine in back of the expression cards. The headings are stored in the memory unit during the computing and each is available for printing whenever the corresponding value or column of values is printed.

Checking of Sample Data

In many instances, it is desirable to check the accuracy of the machine-coded instructions before computing multiple cases of data. A satisfactory check of the accuracy is a comparison of a set of anticipated (hand-calculated) results with a set of machine computed results. This comparison of results checks the accuracy of the coded instructions for:

1. The interpretation of the mathematical symbols.
2. The machine-coding of the operations.
3. Comprehensiveness of the library subprograms.

A set of sample data includes a value for each data reference symbol in the expressions for the problem. It also includes a value for the anticipated result to each expression. These anticipated results are fed to the machine in the same manner as input data.

The machine computed results for the algebraic equations are self-explanatory. The machine computed results for the logical controls of the BACAIC system are indicated as follows:

Expression	The Machine Computed Result
WHN A GRT B USE 6	....6 or the number of the next consecutive expression
WHN A LES B USE 8	.....8 or the number of the next consecutive expression
TRN 12	 12
ARG M TBL $K1 * Y$	Y the interpolated value from the table to be substituted for Y
MOD $H+X$ LIM Y.	.$[H+X]$ the incremented value for H

The computing is started in the normal manner. The computed result for expression number 1 is compared with its anticipated result. If these two values are the same (slide-rule accuracy), expression number 2 is computed and its two results are compared, then number 3, etc. When the computing is finished for all the expressions the value for the machine computed result for each expression is printed.

If the two results (anticipated and computed) for a comparison are not the same for an expression, both result values are immediately printed for that expression. Accompanying these values are appropriate comments and sufficient data to analyze the difference. These data include the values for the parameters, the values for the constants, and the values for the computed results stored in the memory unit at that time. Usually this information is sufficient to isolate or to indicate where the discrepancy occurred.

In order to locate more than one error during each check-out period, the computing is continued without interruption. The value for the anticipated result for the questionable expression is substituted for the computed result for that expression, and the computing is continued for the next expression. This test and substitution is made so that an error in one expression at the beginning of a check cannot be reflected throughout the computation. If this substitution were not made, it is possible the results of the succeeding expressions might not compare with the anticipated hand-calculated results.

If the difference (as explained) is the result of an incorrectly keypunched expression card, a new program must be coded by the machine. After the expression cards are corrected as indicated, the coding phase is repeated. If there are only small differences between the anticipated results and the computed results, the computer accepts the coding for the expressions.

When the reason for the difference between the expected and computed results is not obvious after examining the data print-out for the sample computation, the coding of the expression or expressions must be repeated. The reason for the error may be discovered if the 3-address Sequence Table is printed. This Sequence Table indicates the order of machine execution for each operation in the expression. The table is printed as follows:

Sequence Number	Address for the Left-hand Quantity	Operation Code	Address for the Right-hand Quantity	Address for the Result
1				
2				
3				

The exact sequence of the machine operations can be examined and the reason for the error determined. A possible misplaced front or back bracket symbol can alter the correct sequence.

There will be occasions when even this sequence table is not sufficient. In these cases the coding is repeated and the table of character codes for the expression is printed. This, however, is of very little use without an explanation of the character codes and is used only in extreme cases.

When the results compare favorably for each expression, the instruction cards for the program are accepted as correct.

Economic Aspects

The BACAIC system is a completely automatic system for which the only required information for a new problem is:

1. The algebraic expressions with the result column headings.
2. The decimal input data for each case.

The operation of the machine is entirely dependent on the instructions contained in the system. A few of these are controlled by the ON or OFF position of some external switches on the console panel. However, the machine operator (not the originator of the problem) is responsible for the position of these switches.

This system encourages the programmer to direct more attention toward the mathematical preparation of the expressions. This is a field in which he probably is better trained and more experienced than in the field of machine coding. More time can be spent concentrating on the phases of the problem which require human judgment and decision and less time on the tedious task of coding. Also, a minor detail which is quickly apparent after the first attempt with BACAIC, is the noticeable lack of careless errors. When a comment for a careless error is machine printed on the result sheet, a greater effort is made by the programmer to eliminate such errors before using the machine The standardizing of the procedure for the machine operator helps to avoid confusion which often results from vague or poorly written individual operating procedures.

A vital point of interest to most computer users concerns the amount of elapsed time from the outline of a problem to the time when the first production results are ready. Service to outside departments based on overnight or 24-hour planning is probably the best that any installation can reasonably strive for. When using the BACAIC system for solution of a problem, this goal is within reason. An estimate of the time required to solve a special problem can be based on the approximate time required to perform the various steps as follows:

Number of Expressions	10 Expr.	30 Expr.	50 Expr.
1. Write the expressions and prepare the data	1 hr.	2 hrs	4 hrs.
2. Key punch the expressions and data	1/2 hr.	1 hr.	1 1/2 hrs.
3. Machine code the expressions	2 min.	3 min.	5 min.
4. Machine compute one set of results	10 sec.	40 sec.	1 min.

The time allotments for the various steps are generous and can be decreased by improving the pieces of the system and also by increasing the experience of the programmers for writing the expressions. Often, a different approach to the same problem results in fewer expressions and shorter machine time.

It required approximately 18 man-months to outline the plan, develop the ideas, establish the rules, write the instructions, and "debug" these instructions for the BACAIC system. These 18 man-months were spread through an elapsed time of 1 year. At the end of the first 10 months, most of the system was in working order. The usual elusive errors and unreasonable reasoning had to be located and deleted. The majority of the subprograms were already available, and only had to be altered to include error comments rather than error stops.

The writing of the instructions for this system is proving to be a never-ending process and will only cease when more advanced ideas are not forthcoming. It could easily be in a continuous state of change if all the ideas for improvement are accepted and included.

The Adaptability of a Natural System

A 1-to 2-hour informal discussion describing the general outline and operation of BACAIC is our method of introducing it to various engineering groups. In each case, this discussion is followed by the distribution of a written digest of the BACAIC system. This digest quickly reviews the preparatory steps for writing both the expressions and data and out-

lines the operating steps for the machine procedure. Many of the rules for writing the expressions are merely a review of the fundamentals taught to a beginning student of algebra. These rules must be strictly adhered to so that both the programmer and the machine interpret each situation in the same manner. The talk accompanied by the digest proves to be sufficient information for a beginner. Naturally, there are individual questions at a later time. These questions usually concern the inclusion of additional features to improve the system. All of these new ideas are welcome and whenever possible they are included immediately.

It is worth mentioning that most improvements and additions are readily included. The BACAIC system was originally planned with that desired flexibility in mind. It is fairly easy to include an additional mnemonic symbol but, if the symbol is to refer to a library subprogram for its operation, the library subprogram must also be available.

The specific information for each problem; i.e., the expressions and the input data, can be relayed by means of the telephone from another department to the computing facility. This service is possible because of the standard procedure of the BACAIC system after the expressions are formulated. The availability of telephone service tends to decrease the elapsed time for solving a problem, and also tends to increase the correctness of the written expressions. Individual pride and reputation play an important part in reducing careless errors. Usually, the "debugging" of the expressions is possible in a few minutes prior to any machine operation. This fact is of tremendous advantage economically since without an automatic system considerable machine time is spent on check-out for each problem.

Conclusions

A natural computation language eliminates the machine coding details currently responsible for the expenditure of large amounts of man and machine time. This language can include complete machine operating directions as well as the mathematical problem statements. The time now spent in digital computer problem preparation can be reduced by as much as 90 per cent through the use of machine self-coding systems. A fundamental computation language makes evolutionary machine changes possible without extensive personnel retraining. The man-hours required to construct an autocoding system are no more than those formerly spent on a subprogram library. The basic interpretive principles for an algebraic computing system are applicable to most present-day stored program digital computers. These principles can be incorporated in the hardware of future machines.

Lincoln Laboratory Utility Program System

H. D. BENNINGTON C. H. GAUDETTE

THIS paper discusses a utility program system to assist the coding, check-out, maintenance, and documentation of large-scale control programs. A typical program contains 50,000 instructions, 1,000,000 bits of data storage, and is prepared by a staff of 20 to 40 programmers, many relatively inexperienced. The utility system requires 25,000 registers.

H. D. BENNINGTON and C. H. GAUDETTE are with the Lincoln Laboratory of the Massachusetts Institute of Technology, Lexington, Mass.

An Automatic Supervisor for the IBM 702

BRUSE MONCREIFF

VERY little experience has been accumulated in the operation of a large commercial data-processing center. However, reflection on the subject has led to the conclusion that, in the large-scale operation of such a system, there will be a different emphasis from the one usually present in the operation of a large-scale computing installation. The general administrative problem in both cases is, of course, to keep both staff and equipment operating efficiently. In the latter case, however, the emphasis is on new problem preparation, while in the case of the business application the emphasis must be on the efficient day-after-day operation of the same routines. The automatic supervisory routine described here is an attempt to solve those operating and programming problems peculiar to this "routine-dominated" situation.

The excuse for solving these problems with a machine program, rather than by instructions to the operator, is twofold:

1. The human operator cannot compete in speed with the machine in making routine decisions and in controlling the processing operations.

BRUSE MONCREIFF is with The Rand Corporation, Santa Monica, Calif.