A COMPUTER FOR DIRECT EXECUTION OF
ALGORITHMIC LANGUAGES

James P. Anderson
Burroughs Corporation
Paoi, Pennsylvania

INTRODUCTION

The so-called general-purpose computer is, in a sense, an incomplete design for any given problem. The program is the device which renders the hardware of such a machine a special-purpose computer for a particular problem. In a loose sense, then, programming can be considered a form of machine logical design for specific problems, but at a higher level of abstraction than that generally considered in logical design. This loose equivalence between program and computer has long been recognized by some [1] but exploitation of the relationship has apparently received little impetus, except in a trivial way. Practical benefits of this equivalence have, thus far, been limited to such things as index registers for automatic order modification, built-in floating-point operations, and the like. The justification for implementing such features as additional hardware usually stems, at present, from an inordinately long run time for a particular function, or from excessive difficulty in performing the operations programmatically.

One area yet to be exploited by machine designers is, in fact, that of programming itself. Justification for doing so abounds, with high-priced, high-speed machines frequently employed up to two-thirds of total operating time in the nonproductive tasks of compilation and debugging [2].

The machine outlined in this paper represents another of a recent series of efforts to make computers more easy to use [3]. The major problem which has been addressed is that of programming. It is hoped that the machine described will become a forerunner of a class of computers for which problems are so readily programmed, that programmers can begin to view problems again, rather than details of machine organization and operation.

Although this paper stems from work in the general area indicated in the title, ALGOL '60 was chosen as the basis for the machine because of the well-defined syntax it offers as a guide in structuring similar machines for other algorithmic languages[4].

OUTLINE OF ALGOL '60 FEATURES

ALGOL '60 contains, as separate entities, statements of several forms, each statement representing an abstraction of a computational form. These statements include declarations usually associated with the definition of data in a program, as well as those associated with the definition of subroutines (procedures). The imperative statements of the language can be considered to be of two classes. One class includes the assignment statements and procedure calls which are the computational instructions of ALGOL. All other imperative statements constitute a second class which can be characterized as
control statements (conditional statements, go-to statements, for statements, compound statements, dummy statements) used to direct or mark the path of computation, to define loops, and to determine the scope of other control statements.

ALGOL, like many other algorithmic languages, is based upon the sequential execution of a series of statements or statement groups. The flow of control is normally from one statement to another. This sequencing through the statements of an ALGOL program is interrupted when encountering: a) a conditional statement, where the evaluation of a Boolean expression causes a new sequence to be initiated, b) an unconditional transfer of control (go-to statement), and c) iterated statements, where the range of the iterated variable is defined in the structure of the program statements that follow. It is evident, then, that the ALGOL structure abstracts the control form of "general-purpose" computers. It differs in that the "Instructions" of ALGOL (assignment statements) may be arbitrarily complex to the extent of being entire programs (procedures), and the name of a procedure or function for obtaining a data value may be used in any expression where that data is required.

CONTROL STRUCTURE

The reference language provides syntactic rules for joining different elements of the language to make meaningful statements. These same rules permit the recognition of syntactic elements of statements and the assignment of meaning (through execution of the appropriate operations) to the syntactic elements.

In addition to abstracting more conventional coding structures, ALGOL '60 is recursive. That is, any of the control statements may appear nested within others or themselves to an arbitrary depth. As an example,

If if (Boolean expression) then r else s then

begin

for i: = 1 step 1 until n do

begin

(assignments or procedure statements)

end

(assignments or procedure statements);

end else (statement);

illustrates the recursive use of a conditional statement and a compound statement. The control for sequencing ALGOL statements must reflect the recursive structure of the language.

A pushdown list, or current control-state stack, can be used to keep track of the current control state, or statement type, effectively recording the structure of a program. A stack is a first-in, last-out device. When data is to be placed in a stack, all previous data is pushed down one position, and the new data entered into the top of the stack. Data is obtained destructively from a stack; data removed is erased, and all previous data is pushed up one position. The entries into this stack, which can be considered "states" of the program, determine the interpretation (in terms of control) of subsequent program symbols.

The sequence of delimiters in an ALGOL program establishes control states corresponding to the type of control or computational statement involved. In some cases (for example, for statements) subcontrol states are established. The handling of iteration and conditional statements can be accomplished by manipulating only the top of the current control-state stack and the next sequential control operator or separator from the program. In a similar manner, the control
necessary for assignment and procedural statements can also be obtained.

Syntactic recognition of elements can be illustrated in matrix form, indicating, for a given current control state, the action to be taken by the machine upon encountering each of the symbols of the program. Figures 1 through 4 are matrices indicating the control states in the machine. The rows of the matrices correspond to the current control state, while the columns correspond to the various delimiters and identifiers that can occur in a program. In each of the figures, only those delimiters that are important to the states in question are shown. The occurrence of delimiters other than those shown in either undefined or an error condition. The control state names are roughly suggestive of their corresponding control functions.

The following notation is used for the control sequences in the recognition and control matrix and subsequent sections:

<table>
<thead>
<tr>
<th>Notation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enter (state name)</td>
<td>Push down the control-state stack, and record the control state named in the stack; establish the control state named upon completion of the sequence.</td>
</tr>
<tr>
<td>Replace with (state name)</td>
<td>Record the control state named in the top of the stack; establish the control state named.</td>
</tr>
<tr>
<td>READ</td>
<td>Obtain the next program symbol.</td>
</tr>
<tr>
<td>REPEAT</td>
<td>Push up the control-state stack, and establish the new control state of the control state uncovered; execute the control operation specified by the new state and the old program symbol.</td>
</tr>
</tbody>
</table>

**EXPRESSION CONTROL**

In general, actual computation is carried out in the EXPN state. In this state, two additional stacks are used, in order to execute expressions with due regard for the precedence of the operators. This precedence is necessary to resolve apparent ambiguities in expressions where parentheses are omitted. If the operators are ordered as follows:

\[
\begin{align*}
\text{Next program symbol} & <, \\
\text{Next program symbol} & =, \\
\text{Next program symbol} & >,
\end{align*}
\]

then there are essentially three control sequences to be handled. The sequences are based upon the contents of the top of an operator stack, and the next operator obtained from the program [5],[6],[7]. All identifiers, regardless of the state of the operator stack, are entered in the operand stack.

The control sequences based upon the relative precedence of the next program symbol (operator) and the operator in the top of the operator stack are as follows:

<table>
<thead>
<tr>
<th>Notation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enter in operand stack; READ; execute the instruction in top of operator stack using operand(s) in top of operand stack; REPEAT</td>
<td></td>
</tr>
<tr>
<td>Execute instruction in top of operator stack using operand(s) in top of operand stack; replace top of operator stack with new symbol; READ</td>
<td></td>
</tr>
<tr>
<td>Enter in operator stack; READ</td>
<td></td>
</tr>
</tbody>
</table>

The notations \(>\), \(<\), and \(=\) are read "greater precedence," "lower precedence," and "equal precedence." The notation for \(<\), is similar to the REPEAT used in the control sequences. REPEAT here means "raise the operator stack, and determine the next control sequence upon the basis of the new operator in the operator.
A Computer for Direct Execution of Algorithmic Languages / 187

Figure 1. Iteration Control

stack and the program symbol previously found." The EXPN state is left (via a REPEAT) whenever one of the delimiters in Figures 1 through 4 is encountered.

MACHINE LANGUAGE

The machine language is based upon the specification for ALGOL '60. Programs are strings of symbols drawn from a 2048-character alphabet making up declarations and statements imbedded in the ALGOL control structure. In general, the attempt has been to implement the major computational aspects of the language without unduly complicating the control. The following is a partial list of the deviations from the ALGOL reference language:

a) No own declarations
b) No arithmetic expressions as actual parameters
c) Omission of comment from programs
d) Omission of strings as data elements
e) Limited use of designational expressions.

COMPUTER ORGANIZATION AND ELEMENTS

The major elements of the system shown in Figure 5 are a program memory, a value memory, an arithmetic unit (with associated arithmetic registers), three stack memories used for interpretation and control, and an address table. The arithmetic unit and value memory are much like those in conventional
processors. The three stacks are used for control and temporary storage of the operand names (identifiers) and operators, for proper execution of expressions.

The program memory stores the individual symbols of a program, one to a word, or syllable. One bit is used to specify to the machine whether a syllable is an identifier or a delimiter. Associated with the program memory is a symbol counter (SC). The symbol counter, which corresponds to a program counter in a conventional machine, addresses successive symbols of the program in the program memory, and places these symbols in a staticizing register, or window register (W). There, the type of symbol is determined, and, depending upon the control state, is entered into the appropriate stack. The two auxiliary registers (IVR and RVR) are used in iteration.

One of three stack memories (C), with its associated stack pointer (K), records the current control state. The control delimiters of the program cause generation of the appropriate control state. This state, entered into stack memory C, then determines a new control regime. The other two stack memories (O and A), with their associated stack pointers (S and H), record arithmetic operators and identifiers, respectively.

The address table is a mechanism for relating the identifiers (names) of data with the locations in the value memory where the values are stored. The entries in the address table...
<table>
<thead>
<tr>
<th>NEXT SYMBOL</th>
<th>END</th>
<th>THEN</th>
<th>IF</th>
<th>ELSE</th>
<th>;</th>
</tr>
</thead>
<tbody>
<tr>
<td>STATE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CONS</td>
<td>(preceding EXPN true) replace ST3 enter ST READ</td>
<td>count skip counter up by 1 READ</td>
<td>(if skip counter is zero) replace ST2 READ</td>
<td>(if not) count skip counter down by 1</td>
<td>(if skip counter is now zero) replace ST2 READ</td>
</tr>
<tr>
<td>ST1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST3</td>
<td>REPEAT</td>
<td>replace FS7 REPEAT READ</td>
<td>REPEAT</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 3. Conditional Statement Control

The words of the value memory contain the values corresponding to identifiers in the program. Either a uniform representation of integer and real values similar to that found in the Burroughs B5000 system, or an integer floating-point tag on each data word is required, since the operators do not differentiate between integer and real values. Two counters (P and Q) are used with the value memory to record the extent of data and to file control words. Data storage starts at the low end of memory and runs toward the high end, as shown in Figure 5. In addition to arithmetic variables, the value memory is used to store control words that are, in the main, declared variables, the identifier of which has been used again in a lower level block of
the program. Control words are recorded starting at the high end of memory, and are run toward the low end.

The arithmetic unit proper is conventional, except, perhaps, that the specification of arithmetic type is carried in the data word rather than in the instruction. The arithmetic registers (T and U) are the accumulator and extension register as well as the top two positions of an arithmetic stack.

**OPERATION OF THE SYSTEM**

**Declarations**

Declarations encountered in the program are executed to reserve space in the value memory corresponding to the identifiers used. The sequence upon encountering real, integer, or Boolean declarations causes the location in the value memory assigned to that identifier to be recorded in the address table in the position corresponding to the identifier named. An array declaration causes a block of storage corresponding to that identifier to be reserved by advancing the data storage counter (P) by the amount indicated by the bound pair list. The old value of the address table is recorded as a control word in the value memory if any identifiers have been used at a higher level before the new storage position is recorded.

For example, an integer declaration of the form

```
integer a;
```

has the following control effect:

1. Upon encountering integer in the program, cause a fill switch to be set and the RS state to be entered.
2. Upon encountering the identifier a in the window register (W),
   a. (H) + 1 → H increase the A stack list pointer by one
   b. (W) → H* enter W into the A stack
3. Upon encountering the semicolon,
   a. (P) + 1 → P advance P counter
   b. (H*) → ATR A stack → address table register
   c. (H) - 1 → H reduce A stack pointer by one
   d. if (ATR*(L)) if address table position has been used, do e and f, else go to g
   e. (Q) - 1 → Q advance Q counter
   f. (ATR*) → Q* record previous address table entries control word
   g. (P) ⊕ (L) → ATR* record P and L in address table
   h. type mark set type mark (integer, real) in data word

---

**Figure 4. Assignment and Block Structure Control**
In these expressions, the asterisk is read as "addressed by," such that H* means the location (in the A stack) addressed by the contents of H. The symbol ( ) means "the contents of," such that (A) is read as "the contents of A," and (A*) is read as "the contents of the location addressed by A." The symbol ⊕ means that the adjacent words are concatenated, or merged into a single word.

In a manner similar to the above, other declarations are executed to reserve storage and to retain the previous identifier in the address table.

Arithmetic and Boolean Expressions

Arithmetic and Boolean expressions, occurring in assignment statements, designational expressions, and the like, describe the computations to be performed. The computations represented by these syntactic forms are executed in the EXPN state. In this state, the identifiers enter the A stack, and the operators enter the O stack. The sequence of execution is determined by the precedence of the arithmetic and logical operators.

For example, an assignment statement of the form

\[
C = a + b;
\]

is executed via the following sequences (where W, again, is the identifier encountered in the window register):
Iterated Statements

Iterated statements are controlled by statements of the form
for <variable> := <iteration list> do.
The scope of the running variable is determined by the structure of the subsequent statements. The two auxiliary registers (IVR and RVR) hold the appropriate place in the iteration list of the for statement. When the scope of the running variable is terminated, the symbol counter is set to the appropriate value (contained in IVR or RVR) to obtain the next element of the iteration list. The process continues until the end of the list is reached, at which time the statement to which the running variable applies is skipped over.

Conditional Statements

In a like manner, conditional statements are executed. The CONS state is recorded in the C stack, and the EXPN state is then entered to evaluate the Boolean expression. Upon termination of the evaluation, the truth or falsity of the expression is determined in the CONS state, and the appropriate state is entered to execute either the true statement(s) or the false statement(s). In either case, provision is made for skipping over the alternate statement.

Procedures

The procedural form of ALGOL '60 is an abstraction of subroutine structure, and differs substantially from ordinary machine subroutines in that it permits recursive calls on subroutines. This provision implies a mechanism for assigning storage at runtime. Such a mechanism exists in this machine inasmuch as the execution of variable declarations reserves storage for each variable, when encountered. The PRO state merely records in the address table the position of the procedure in the program memory and skips over the body of the procedure. Execution of the procedure causes a change of level, a recording of the return point, and entry of the program memory location of the procedure in the symbol counter. The specifiers and variables for the procedure are executed, previously used labels being filed in the value memory as control words. Procedures declared within procedures are handled in a similar manner.

RELATION OF MACHINE STRUCTURE TO ALGORITHMIC LANGUAGES

Several features of this machine have generality beyond ALGOL '60. These features can be identified in relation to characteristics of the language. The control stack is a device for recording the static structure of a program as a series of states which determine the interpretation of subsequent symbols of the program. This device takes advantage of the sequentiality of computation processes. Such a structural device is inadequate, of itself, for representing control of parallel computational processes, a subject that has become of increasing interest recently. Any language form that provides
indicators for parallel processes may be handled by multiple arithmetic and control units in a manner similar to that employed in the Burroughs B5000 computer system.

The address table feature of the machine is a direct analogue of the tag tables maintained in an algebraic translator. In this version of the machine, the address table is a limited form of associative memory. To provide the feature of variable-length identifiers, a link-list memory would suffice, but at some sacrifice in efficiency.

The A and 0 stacks for identifiers and operands are another analogue borrowed from the construction of compilers. These stacks are necessary because of the precedence attached to various operators. Without the A and 0 stacks, some kind of pre-execution translation would be necessary for the machine.

For similar languages, somewhat different recognition logic may result, but the structure of the machine would remain essentially the same.

CONCLUSION

The task of working toward an isomorphism between machine organization and the manner in which a user expresses a problem has occupied a major segment of the computer industry for many years. The machine outlined represents another step in this process. More than anything, the machine illustrates the enormous strides made in constructing algorithmic languages that properly abstract programming techniques. It further illustrates the functional requirements necessary to directly implement current forms of algorithmic languages. The emphasis has been on control rather than arithmetic expressions, because of the large body of work that already exists on the latter.

The organization outlined above is perhaps an oversimplification, but since good programming languages properly reflect the concepts and abstractions for a particular problem class, their efficient implementation can be a useful design goal when considering a new machine.

ACKNOWLEDGMENTS

The author is indebted to E. L. Glaser and K. Speierman of the System Research Department, Burroughs Laboratories, for providing the milieu and many valuable suggestions in connection with this work. A particular debt is owed R. Barton, a consultant for Burroughs Corporation, for the valuable insights in connection with this machine and others that placed many aspects of the programming abstractions represented by ALGOL in their proper perspective.

REFERENCES