NATIONAL PHYSICAL LABORATORY

TEDDINGTON, MIDDLESEX, ENGLAND

PAPER 2-3

AUTOMATIC PROGRAMMING PROPERTIES AND PERFORMANCE OF FORTRAN SYSTEMS I AND II

by

J. W. BACKUS

BIOGRAPHICAL NOTES

John W. Backus received his A.M. degree (mathematics) at Columbia University in 1950. Since 1950 he has been with International Business Machine Corporation working on programming, computer systems design, and automatic programming systems design. He is at present manager of the Programming Research Department.

He is author of "The IBM 701 Speedcoding System" (1953) and joint author (with G. M. Amdahl) of "The System Design of the IBM Type 704" (1955) presented to meetings of the Association for Computing Machinery.

AUTOMATIC PROGRAMMING:

PROPERTIES AND PERFORMANCE OF FORTRAN SYSTEMS I AND II

by

J. W. BACKUS

SUMMARY

A BRIEF general discussion of the goals and methods of automatic programming techniques is followed by a somewhat detailed description of the input languages of FORTRAN Automatic Coding Systems I and II. The statements of these input languages provide (in part) concise means for writing algebraic expressions, for specifying iterative repetitions of portions of a procedure, for referring to one, two and three dimensional arrays of data, for defining new functions and procedures, and for specifying input and output procedures. Recent extensions of the input language are described.

FORTRAN I has been in use for a year and a half. Over half the instructions being written for some sixty 704 installations are thought to be produced by FORTRAN. The cost of programming and "debugging" is reduced by about 4-to-1. A few other statistics and applications are cited. The final section discusses the relationship of automatic programming and the mechanization of thought processes.

INTRODUCTION

AUTOMATIC programming techniques seek to make it easy to do a desired job on a computer. This is accomplished by providing a language in which the statement of the desired process is natural and concise, together with some means of causing a machine to carry out the stated process. Given a machine equipped to accept statements of procedures in a language other than the "hardware language", one may ask the same questions about this "synthetic" machine that are used to evaluate a real machine: (1) how easy is it to prepare programs for, and (2) how rapidly does it execute the programs for which it will be used? In general, automatic programming systems provide synthetic machines which are far superior to their real counterparts with respect to question (1) and range from slightly inferior to greatly inferior to their real counterparts on question (2).

Automatic programming systems have been widely used in the United States for a number of years. The purpose of the early systems was to provide synthetic machines which had floating-point operations and often index registers (B-tubes), since the real machines did not. In many instances the synthetic machines were only one-half to one-tenth as fast as their real counterparts, but although their input languages were machine-like, they were considerably easier to program.

In the past eighteen months a number of new automatic programming systems have been completed, and others are being developed, which have strikingly similar input languages. These languages use mathematical notation and a variety of statements which make the writing of most procedures of numerical computation a natural and concise process. The synthetic machines provided by these systems range in speed from virtually the speed of the corresponding real machine to perhaps one-half of that speed. FORTRAN is one of these systems. Considerable effort was devoted to making the speed of the synthetic FORTRAN machine as close as possible to that of the IBM 704, its real counterpart. This paper will describe the language and some of the recent developments of this system and report on some of the effects which it has had on the computing work at a number of 704 installations.

In the field of commercial data processing a number of automatic programming systems have recently been under development which provide considerable simplifications in specifying jobs in this area. Automatic programming systems for data processing, however, still face several difficult problems:

(1) Existing languages are often too rigid to describe various desirable procedures and situations and, in many instances, are not sufficiently concise.

(2) The need for synthetic machines to be virtually as fast as their real counterparts is much greater in the data processing area than in computing and the difficulty of achieving this is greater.

PROPERTIES OF FORTRAN

GENERAL

The FORTRAN synthetic computer is realized by a 704 program, called the FORTRAN translator, which accepts programs written in FORTRAN language and produces corresponding 704 machine language programs. The translator program was written over a period of two and one-half years with an expenditure of about 18 man-years of effort; it comprises a program of about 25,000 instructions. The translator program was made available to all 704 installations in April, 1957. A report of the usage of the system since that time appears in a subsequent section of this paper.

Since the initial distribution of the FORTRAN system, certain extensions have been made to the input language and the translator has been considerably modified to accept this extended language. The resulting system is known as FORTRAN II; it has recently been distributed to 704 installations†.

FORTRAN I INPUT LANGUAGE

The following description-by-example of the FORTRAN I input language is extracted from a previously published paper *(ref. 1)*.

Arithmetic Statements

Example 1: Compute:

$$\text{root} = \frac{-(B/2) + \sqrt{(B/2)^2 - AC}}{A}$$

FORTRAN Program:

```
ROOT = (-(B/2.0) + SQRTF((B/2.0)**2 - A*C))/A
```

Notice that the desired program is a single FORTRAN statement: an arithmetic formula. Its meaning is: "Evaluate the expression on the right of the = sign and make this the value of the variable on the left." The symbol * denotes multiplication and ** denotes exponentiation (i.e., A**B means A^B). The program which is generated from this statement effects the computation in floating-point arithmetic, avoids computing (B/2.0) twice and computes (B/2.0)**2 by a multiplication rather than by an exponentiation routine. Had (B/2.0)**2.01 appeared instead, an exponentiation routine would necessarily be used, requiring more time than the multiplication.

The programmer can refer to quantities in both floating-point and integer form. Integer quantities are somewhat restricted in their use and serve primarily as subscripts or exponents. Integer constants are written without a decimal point. Example: 2 (integer form) vs 2.0 (floating-point form). Integer variables begin with I, J, K, L, M, or N. Any meaningful arithmetic expression may appear on the right-hand side of an arithmetic statement, provided the following restriction is observed: an integer quantity can appear in a floating-point expression only as a subscript or as an exponent or as the argument of certain functions. The functions which

† The new sections of the FORTRAN II translator for the 704 were written by G. E. Mitchell, P. B. Sheridan, B. Brady and L. May with the assistance of the authors of the original FORTRAN system.

the programmer may refer to are limited only by those available on the library tape at the time, such as SQRTF, plus those simple functions which he has defined for the given problem by means of function statements. An example will serve to describe the latter.

Function Statements

Example 2: Define a function of three variables to be used throughout a given problem, as follows:

```
ROOTF(A,B,C) = (-(B/2.0) + SQRTF((B/2.0)**2 - A*C))/A
```

Function statements must precede the rest of the program. They are composed of the desired function name (ending in F) followed by any desired arguments which appear in the arithmetic expression on the right of the = sign. The definition of a function may employ any previously defined functions. Having defined ROOTF as above, the programmer may apply it to any set of arguments in any subsequent arithmetic statements. For example, a later arithmetic statement might be

```
THETA = 1.0 + GAMMA*ROOTF(PI, 3.2*Y + 14.0, 7.63)
```

DO Statements, DIMENSION Statements, and Subscripted Variables

Example 3: Set Q_{max} equal to the largest quantity $P(a_i+b_i)/(P(a_i-b_i)$ for some i between 1 and 1000 where $P(x) = c_0 + c_1x + c_2x^2 + c_3x^3$.

FORTRAN Program:

```
1       POLYF(X) = C0+X*(C1+X*(C2+X*C3))
2       DIMENSION A(1000), B(1000)
3       QMAX = -1.0E20
4       DO 5 I = 1, 1000
5       QMAX = MAXF(QMAX, POLYF(A(I) + B(I))/POLYF(A(I) - B(I)))
6       STOP
```

The program above is complete except for input and output statements which will be described later. The first statement is not executed; it defines the desired polynomial (in factored form for efficient output program). Similarly, the second statement merely informs the executive routine that the vectors A and B each have 1000 elements. Statement 3 assigns a large negative initial value to QMAX, $-10x10^{20}$, using a special concise form for writing floating-point constants. Statement 4 says "DO the following sequence of statements down to and including the statement numbered 5 for successive values of I from 1 to 1000." In this case there is only one statement 5 to be repeated. It is executed 1000 times; the first time reference is made to A(1) and B(1), the second time to A(2) and B(2), etc. After the 1000th execution of statement 5, statement 6 -- STOP -- is finally encountered. In statement 5, the function MAXF appears. MAXF may have two or more arguments and its value, by definition, is the value of its largest argument. Thus on each repetition of statement 5 the old value of QMAX is replaced by itself or by the value of POLYF(A(I)+ B(I))/POLYF(A(I) - B(I)), whichever is larger. The value of QMAX after the 1000th repetition is therefore the desired maximum.

Example 4: Multiply the $n \times n$ matrix a_{ij} ($n \leq 20$) by its transpose, obtaining the produce elements on or below the main diagonal by the relation

$$c_{i,j} = \sum_{k=1}^{n} a_{i,k} \cdot a_{j,k} \qquad \text{(for } j \leq i\text{)}$$

and the remaining elements by the relation

$$c_{j,i} = c_{i,j}.$$

FORTRAN Program:

```
        DIMENSION A(20,20), C(20,20)
        DO 2 I = 1, N                                  P
        ↓                                              ↓
      ┌──────────────────────────────────────────────────┐
      │ DO 2 J = 1, I                                  Q │
      │ ↓                                              ↓ │
      │┌───────────────────────────────────────────────┐ │
      ││C(I,J) = 0.0                                   │ │
      ││                                               │ │
      ││DO 1 K = 1, N                                 R│ │
      ││↓                                             ↓│ │
1     ││┌─────────────────────────────────────────────┐│ │
      │││C(I,J) = C(I,J) + A(I,K)*A(J,K)              ││ │
      ││└─────────────────────────────────────────────┘│ │
2     ││C(J,I) = C(I,J)                                │ │
      │└───────────────────────────────────────────────┘ │
      └──────────────────────────────────────────────────┘
        STOP
```

As in the preceding example, the DIMENSION statement says that there are two matrices of maximum size 20x20 named A and C. For explanatory purposes only, the three boxes around the program show the sequence of statements controlled by each DO statement. The first DO statement says that procedure P, i.e., the following statements through statement 2 (outer box) is to be carried out for I = 1 then for I = 2 and so on up to I = N. The first statement of procedure P(DO 2 J = 1, I) directs that procedure Q be done for J = 1 to J = I. And of course each execution of procedure Q involves N executions of procedure R for K = 1, 2,..., N.

Consider procedure Q. Each time its last statement is completed the "index" J of its controlling DO statement is increased by 1 and control goes to the first statement of Q, until finally its last statement is reached and J = I. Since this is also the last statement of P and P has not been repeated until I = N, I will be increased and control will then pass to the first statement of P. This statement (DO 2 J = 1, I) causes the repetition of Q to begin again. Finally, the last statement of Q and P (statement 2) will be reached with J = I and I = N, meaning that both Q and P have been repeated the required number of times. Control will then go to the next statement, STOP. Each time R is executed a new term is added to a product element. Each time Q is executed a new product element and its mate are obtained. Each time P is executed a product row (over to the diagonal) and the corresponding column (down to the diagonal) are obtained.

READ, PRINT, FORMAT, IF and GO TO Statements

Example 5: For each case, read from cards two vectors, ALPHA and RHO, and the number ARG. ALPHA and RHO each have 25 elements and ALPHA(I)≤ ALPHA(I+1), I = 1 to 24. Find the SUM of all the elements of ALPHA from the beginning to the last one which is less than or equal to ARG [assume ALPHA(1)≤ARG<ALPHA(25)]. If this last element is the Nth, set VALUE = 3.14159*RHO(N). Print a line for each case with ARG, SUM, and VALUE.

FORTRAN Program:

```
        DIMENSION ALPHA(25), RHO(25)
  1     FORMAT(5F12.4)
  2     READ 1, ALPHA, RHO, ARG
        SUM = 0.0
        DO 3 I = 1, 25
        IF (ARG - ALPHA(I))4, 3, 3
  3     SUM = SUM+ALPHA(I)
  4     VALUE = 3.14159*RHO(I - 1)
        PRINT 1, ARG, SUM, VALUE
        GO TO 2.
```

The FORMAT statement says that numbers are to be found (or printed) 5 per card (or line), that each number is in Fixed-point form, that each number occupies a field 12 columns wide and that the decimal point is located 4 digits from the right. The FORMAT statement is not executed; it is referred to by the READ and PRINT statements to describe the desired arrangement of data in the external medium.

The READ statement says "READ cards in the card reader which are arranged according to FORMAT statement 1 and assign the successive numbers obtained as values of ALPHA(I) I = 1, 25 and RHO(I) I = 1, 25 and ARG." Thus "ALPHA, RHO, ARG" is a description of a list of 51 quantities (the size of ALPHA and RHO being obtained from the DIMENSION statement). Reading of cards proceeds until these 51 quantities have been obtained, each card having five numbers, as per the FORMAT description, except the last which has the value of ARG only. Since ARG terminated the list, the remaining four fields on the last card are not read. The PRINT statement is similar to READ except that it specifies a list of only three quantities. Thus each execution of PRINT causes a single line to be printed with ARG, SUM, VALUE printed in the first three of the five fields described by FORMAT statement 1.

The IF statement says "If ARG - ALPHA(I) is negative go to statement 4, if it is zero go to statement 3, and if it is positive go to 3." Thus the repetition of the two statements controlled by the DO consists normally of computing ARG - ALPHA(I), finding it zero or positive, and going to statement 3 followed by the next repetition. However, when I has been increased to the extent that the first ALPHA exceeding ARG is encountered, control will pass to statement 4. Note that this statement does not belong to the sequence controlled by the DO. In such cases, the repetition specified by the DO is terminated and the value of the index (in this case I) is preserved. Thus if the first ALPHA exceeding ARG were ALPHA(20), then RHO(19) would be obtained in statement 4.

The GO TO statement, of course, passes control to statement 2, which initiates reading the 11 cards for the next case. The process will continue until there are no more cards in the reader. The above program is entirely complete. When punched in cards as shown, and compiled, the translator will produce a ready-to-run 704 program which will perform the job specified.

Other Types of FORTRAN Statements

In the above examples the following types of FORTRAN statements have been exhibited.

Arithmetic statements
Function statements
DO statements
IF statements
GO TO statements
READ statements
PRINT statements
STOP statements
DIMENSION statements
FORMAT statements

The explanations accompanying each example have attempted to show some of the possible applications and variations of these statements. It is felt that these examples give a representative picture of the FORTRAN language; however, many of its features have had to be omitted. There are 23 other types of statements in the language, many of them completely analogous to some of those described here. They provide facilities for referring to other input-output and auxiliary storage devices (tapes, drums, and card punch), for specifying preset and computed branching of control, for detecting various conditions which may arise such as an attempt to divide by zero, and for providing various information about a program to the translator. A complete description of the language is to be found in "Programmer's Reference Manual, the FORTRAN Automatic Coding System for the IBM 704".

Preparation of a Program for Translation

The translator accepts statements punched one per card (continuation cards may be used for very long statements). There is a separate key on the keypunching device for each character used in FORTRAN statements and each character is represented in a single column of the card. Keypunching a FORTRAN program is, therefore, a process similar to that of typing the program.

Translation

The deck of cards obtained by keypunching may then be put in the card reader of a 704 equipped with the translator program. When the load button is pressed one gets either 1) a list of input statements which fail to conform to specifications of the FORTRAN language accompanied by remarks which indicate the type of error in each case; 2) a deck of binary cards representing the desired 704 program; 3) a binary tape of the program which can either be preserved or loaded and executed immediately after translation is complete; or 4) a tape containing the output program in symbolic form suitable for alteration and later assembly.

FORTRAN II INPUT LANGUAGE

The principal shortcoming of the FORTRAN I input language is that the programmer is unable to define new statements in terms of the given ones. He can introduce only functions which he can define by an arithmetic expression. The input language of FORTRAN II is identical to that of FORTRAN I except that certain additional statements are provided. With these the programmer may create a FORTRAN language subprogram, assign a name to it, and indicate those variables in the subprogram which are to be regarded as the inputs and/or outputs. He may invoke a subprogram in any other program by giving its name and supplying appropriate parameters. There is provision for indicating that a subprogram defines either a function (which may be employed in arithmetic expressions in other programs) or a subroutine (which may be invoked as a statement in other programs).

Example 1: Write a subprogram which clears the first *N* elements of a vector.

FORTRAN II Subprogram:

```
      SUBROUTINE CLEAR(A,N)
      DIMENSION A(1000)
      DO 1 J = 1, N
1     A(J) = 0.0
      RETURN
```

The first statement indicates that the entire program following is a subroutine, that its name is CLEAR, and that its parameters are the vector A and the integer variable N. The last statement indicates that the process which constitutes the subroutine is complete.

Example 2: Write a subprogram which realizes the function:

$$g(a,x) = \sum_{i=1}^{10} a^i \left(\sum_{j=1}^{10} x_{ij} \right)$$

FORTRAN II Subprogram:

```
      FUNCTION G(A,X)
      DIMENSION X(10,10), R(10)
      CALL CLEAR (R,10)
      G = 0.0
      DO 2 I = 1, 10
      DO 1 J = 1, 10
1     R(I) = R(I)+X(I,J)
2     G = G+A**I*R(I)
      RETURN
```

The first statement says that the following subprogram defines a function G of A and X. The CALL statement invokes the subroutine of example 1 to clear the vector R. Notice that the next statement and subsequent ones refer to a variable whose name is that of the function. The value of the variable G when RETURN is executed is the value of the function.

The above subprograms may be translated independently. The output in each case will be a deck of punched cards having the machine language program, with the appropriate prologue for linkage, in a form which may be easily adjusted by a loading program to operate anywhere in the store. In this form the names themselves of called subprograms are retained in the deck representing the calling program and are used by the loading program to establish the appropriate references of each program to its subprograms as they are being loaded. This complete independence of a program from its calling and called programs until the point of loading for execution has several benefits:

(1) Each program and subprogram may be checked out and/or recompiled independently. Thus each correction in the FORTRAN language statement of a process will involve retranslation of only that subprogram in which the change is made.

(2) Each subprogram is produced in precisely the form suitable for inclusion in the library. Inclusion is accomplished simply by placing the output deck in the library deck. When loaded this deck will produce a new library tape which is available to the translator. Library programs, of course, require no translation.

(3) Since each subprogram may be obtained without the inclusion of the subprograms which it calls, the use of two subprograms, which both call a third, need not result in a copy of the third accompanying each of them in the store.

(4) Since the machine-language calling sequences resulting from input language references to functions or subroutines are quite standard, it is immaterial whether a required subprogram is generated by FORTRAN or is written in machine-language. Thus, for example, a special hand-coded input routine may be invoked by a CALL statement.

Example 3: Compute $g(sin(g(\pi,r))^2,s)$ for pairs of 10x10 matrices r, s read from cards, and print the result for each case, where g is the function of example 2.

FORTRAN II Program

```
      DIMENSION R(10,10), S(10,10)
3     READ 1, R, S,
1     FORMAT (10F6.3)
      RESULT = G(SIN(G(3.14159,R))**2,S)
      PRINT 2, RESULT
2     FORMAT (E12.4)
      GO TO 3
```

When this program is translated one may request that any called subprograms which are on the library tape plus a copy of the loading program be incorporated in the output deck. In this case the given program and the subprogram for the sine function might be contained in the output. When the outputs from translating the programs of the first two examples are added to this deck, it is ready to be loaded and run.

FURTHER DEVELOPMENTS IN FORTRAN

In addition to the facilities provided by FORTRAN II, work on the translator program for the 704 has been almost completed to provide the following facilities†.

Use of Symbolic Machine Instructions as Statements in a FORTRAN Program

This facility, of course, makes it possible for the programmer to use operations and to deal with units of information not provided for in the FORTRAN language. Certain restrictions are necessary to effect an appropriate relationship between machine language references to information and references in FORTRAN statements within the same FORTRAN program.

Example: Write a subroutine which sets B(I) = 0.0 or B(I) = 1.0 according as the Ith bit of cell A is 0 or 1 for I = 1, 2,..., N where $N \leq 36$.

FORTRAN III Program:

```
      SUBROUTINE EXPAND (A,B,N)
      DIMENSION B(36)
S     LDQ A
      DO 1 I = 1, N
S     CLM
S     LGL 1
S     TZE*1
      B(I) = 1.0
S     1 STO B(I)
      RETURN
```

In the above program those statements preceded by an "S" are symbolic 704 instructions. For example, the first such statement is "Load the Q-register from cell A". The asterisk in the fourth S-statement indicates

† The principal additions to the translator to provide these facilities were planned and written by I. Ziller and R. A. Nelson.

that the number following is a statement number, which in this case is that of the fifth S-statement. Note that the address part of the latter is a subscripted variable. For those who care to understand this not-very-good program the operation codes have the following meanings:

CLM:	clear the accumulator.
LGL:	shift the contents of the 73 bit register comprising the accumulator (left 37 bits) and the Q-register (right 36 bits) one bit to the left.
TZE:	transfer control when contents of accumulator are zero.
STO:	store contents of accumulator.

Note the services provided by the translator even for essentially non-numerical processes, in this case: generation of a prologue, assignment of storage, loop formation, referencing of arrays.

Use of Boolean Expressions

Provision has been made to permit writing "expressions" in which the operators are "and", "or" and "not" rather than the usual arithmetic operations. When two words are combined under the operation "and" the resulting word has a zero-bit in every position except those in which the two operand words both have a 1-bit. Analogous results are given by "or" and "not", the latter operation having a single operand. "And", "or" and "not" are designated respectively by "*", "+" and "-".

Example: Let A, B, C, D be four quantities which are zero or one according as conditions a, b, c, d respectively, hold or not. Write a statement which makes E zero or one according as the condition

$$a + (-(b*c)*(-d))$$

holds or not.

FORTRAN III Statement

```
B     E = A+(-(B*C)*(-D))
```

The "B" preceding the statement indicates that the operators are to be interpreted as Boolean operators. "B" may be applied in this way to any statement which contains an expression.

Thus,

```
B     IF(A*B) 0,5,14
```

will cause control to pass to statement 5 only if A*B (A "and" B) is zero; otherwise control will pass to statement 14.

Use of Function or Subroutine Names as Parameters of Functions or Subroutines

FORTRAN II permits a function *value* to appear as a parameter of a function or subroutine, [e.g., SIN(COS(X)) meaning the sine of the number, cos(X)]. FORTRAN III has provision for specifying the name of a particular function or subroutine as a parameter of another function or subroutine which refers to one or more *arbitrary* functions or subroutines.

Example: Write a subprogram, R, which, for any arbitrary functions f and g, replaces each element, A_i, $i = 1, 2, \ldots, N$, of a vector A, by

$$f(A_i)^2 + g(A_i+1)^2$$

FORTRAN III Subprogram:

```
      SUBROUTINE R(A,N,F,G)
      DIMENSION A(1000)
      DO 1 I = 1, N
1     A(I) = SQRT(F(A(I))**2+G(A(I+1))**2)
      RETURN
```

Write a subprogram, S, which replaces X_i, $i = 1, 2, \ldots 10$, by

$$sin(x_i)^2 + cos(x_i+1)^2$$

FORTRAN III Subprogram:

```
      SUBROUTINE S(X)
      DIMENSION X(11)
F     SIN, COS
      CALL R(X, 10, SIN, COS)
      RETURN
```

The first of the subprograms above is unusual only in that the names of two functions, F and G, are listed as parameters. The second subprogram calls the first and specifies that SIN and COS are to be used for F and G respectively. The statement preceded by "F" makes it possible to correctly interpret the CALL statement by indicating that "SIN" and "COS" are names of subprograms rather than names of variables, and, therefore, must be retained in symbolic form (up to loading time).

In addition to the above mentioned facilities for stating procedures which the FORTRAN III translator will accept [namely, (1) use of symbolic machine instructions as statements, (2) use of Boolean expressions, (3) use of subprogram names as parameters] there are facilities provided for writing procedures to manipulate alphabetic and other symbolic information. No further discussion of the latter is given here.

Operation of Programs Which Require More than the Available High-Speed Storage Space

The preceding paragraphs describe certain new properties of the FORTRAN translator program. The present section concerns a program called SLAM† (Subroutine Loader And Monitor). SLAM is concerned with the subprograms from which a program is built. At load time it splits these subprograms into groups, each of which will fit into the core space available after the data area has been reserved. It writes these blocks onto tape and at the same time rearranges the linkage between subprograms. Then at execution time the monitor part of SLAM puts in cores, at the appropriate moment, that group of subprograms necessary for the execution of the program. The monitor takes care of the transfer of arguments between subprograms.

EXPERIENCE WITH FORTRAN

Since its distribution over a year ago, the use of FORTRAN I has been steadily increasing. A survey in April of this year of twenty-six 704 installations indicates that over half of them use FORTRAN for more than half of their problems. Many use it for 80% or more of their work (particularly the newer installations) and almost all use it for some of their work. The latest records of the 704 users' organization, SHARE, show that there are some sixty installations equipped to use FORTRAN (representing 66 machines) and recent reports of usage indicate that more than half the machine instructions for these machines are being produced by FORTRAN. SHARE recently designated the FORTRAN language as the second official medium for transmittal of programs within the organization (the other is symbolic machine language).

Programming Economies

Most 704 installations cannot provide accurate information regarding programming costs. However, the Service Bureau Corporation, which does a lot of contract programming, has kept careful records of programming time, machine time used in debugging, and so forth, for many jobs coded in symbolic machine language and many others in FORTRAN. Their analysis of these costs shows that the average cost per debugged instruction obtained from FORTRAN programs is about one-fourth the corresponding cost per instruction obtained from symbolic programs. (A FORTRAN-produced program and a symbolic program for a given job are about the same length.) Since users have found that a FORTRAN-coded program runs at about the same speed as the corresponding hand-coded program, considerable savings are

---

† Written by M. deV. Roberts.

effected. (Where symbolic programming is used, the cost of programming and debugging is usually 1/2 to 3/4 the total cost of solving a problem.) Elapsed time for programming has often been reduced by factors of 10 to 1.

The average ratio of the number of FORTRAN statements in a program to the number of resulting machine instructions (exclusive of all subroutines) is 1 to 7. The translator produces about 100 instructions per minute.

Scope of Applications

Calculations have been programmed using FORTRAN in all the major areas of scientific computation (e.g., jet engine design, airframe design, reactor design, numerical weather prediction, X-ray diffraction analysis, and a multitude of others). It is obvious from the nature of the input language that FORTRAN offers numerous conveniences to programmers in these areas. It is interesting to note the conveniences offered, and hence, the use of the system, in the case of processes which are essentially non-numerical. For example, FORTRAN programs have been or are being written (with considerable payoff in simplicity) to do the following: simulate the behaviour of a control mechanism for a proposed computer (J. Cocke, IBM), play chess (J. McCarthy, MIT), print isobaric weather maps (H. A. Zartner, Jr., U. S. Weather Bureau), and prove theorems in geometry (H. Gelernter and N. Rochester, IBM). Many users who have not made much use of FORTRAN I report that FORTRAN II will greatly enlarge the system's usefulness for their work.

FORTRAN Translators for Various Machines

The preceding paragraphs have referred to various FORTRAN systems for the 704. There are also translator programs for other machines which accept FORTRAN-language programs and produce programs for the machine in question. FORTRANSIT I, II and III are translator programs for three configurations of the IBM 650. Some 265 different 650 installations have requested some of the FORTRANSIT translator programs. The actual extent of use which they make of the system is not known.

Work on FORTRAN translators for the IBM 705 and for the IBM 709 is nearing completion.

RELATIONSHIP OF AUTOMATIC PROGRAMMING TO THE MECHANIZATION OF THOUGHT PROCESSES

In the following remarks "mechanization of thought processes" or "thinking" for a machine will be taken to mean simply "information processing of such a nature and complexity that it leads to behaviour which in a person would be called thinking". In these terms the technique of automatic programming has three roles in relationship to the study of machine thinking: one as recipient, two as contributor.

Certainly the process of devising an efficient machine program to perform some previously stated complex task may be taken to require thinking. As more general and more effective techniques are found for the mechanization of various thought-like processes, their application in the automatic programming area will undoubtedly permit the users of machines to communicate requests for programs with increasing ease and brevity and obtain more efficient programs to perform each desired task. Thus automatic programming will benefit from progress in the mechanization of thought processes.

Conversely, as a special topic in the field, efforts to find better languages for stating procedures and better techniques for translating them into machine language will contribute to the general understanding of complex information-processing methods. Already various translators such as FORTRAN perform some of the more complex information-processing tasks presently done by machines.

Almost by definition the mechanization of thought processes requires the construction of very large and complicated programs. Therefore, it would appear that progress in this area will depend heavily on the availability of automatic programming systems which will make it possible to construct many experimental programs of great length. Some of the people who have used FORTRAN for writing programs, such as the geometry program or the chess program mentioned earlier, feel that the task of writing them would be virtually impossible if they had to concern themselves with the complexities of machine coding in addition to those of the process itself. For example, it is expected that the geometry program will comprise about 16,000 machine instructions and that these will result from 3,000 to 4,000 FORTRAN statements. In this way automatic programming will enlarge the horizon of research in the mechanization of thought processes by permitting us to study processes which would otherwise be beyond our power to realize.

REFERENCE

1. BACKUS, J. W., BEEBER, R. J., BEST, S., GOLDBERG, R., HAIBT, L. M., HERRICK, H. L., NELSON, R. A., SAYRE, D., SHERIDAN, P. B., STERN, H., ZILLER, I., HUGHES, R. A., and NUTT, R. The FORTRAN Automatic Coding System. *Proceedings of the Western Joint Computer Conference, Los Angeles, California,* (Feb., 1957).