Pubblicazioni del

CENTRO STUDI CALCOLATRICI ELETTRONICHE

del C. N. R.

presso l'Università degli studi di Pisa

n. 34

The Internal Structure of the FORTRAN CEP Translator

O. G. Mancino and M. Morandi Cecchi
Centro Studi Calcolatrici Elettroniche, C.N.R., Pisa, Italia

The FORTRAN CEP translator converts a source program written in the FORTRAN CEP language into an object program written in the language of the CEP computer.

In this paper, after an outline of the CEP computer, the internal structure of the translator is described. Emphasis is on the compilation of expressions, of input/output lists, and of subscripted variables.

1. Introduction

The FORTRAN CEP translator converts a source program written in the FORTRAN CEP language into an object program written in the language of the CEP computer.

The FORTRAN CEP language is similar to FORTRAN II, although it allows the use of a large variety of modes in the expressions and in the input/output lists [1].

The CEP is a parallel binary computer with a word length of 36 bits. The main magnetic core memory has 8192 words. The auxiliary memory is composed of a magnetic drum with 16,384 words and 8 magnetic tape units. An instruction occupies a full word, with a single-address doubly modifiable by means of ordinary memory cells which are called *parametric cells*. There are 128 instructions and 219 pseudo-instructions which refer to arbitrary closed subroutines [2]. Arithmetic is fixed- or floating-point. Input is by paper tape. Output is by typewriter as well as paper tape and printer.

2. Overall Structure

The translator is divided into three parts.

In the first part the source program is read statement by statement. If the statement is of declarative type, but not an arithmetic statement function or END, it is translated into table entries and/or symbolic declarations [3]. If the statement is of executable type or an arithmetic statement function or END, it is translated into a string of intermediate statements and, when it is the case, into table entries. In an intermediate statement the single variables, the matrix elements, the Hollerith arguments and noninteger constants are replaced by internal names which specify their kind and relative address in the corresponding tables.

In the second part the string of intermediate statements is translated by means of the tables into a string of symbolic statements [3]. These, when referring to quantities mentioned in the source program, retain the names of those quantities in their addresses.

In the third part the symbolic program produced by the

first and second part is converted into a relocatable program in the CEP language.

Only same features of the translator will be dealth with in this paper. For a complete description see [4].

3. Expressions, Lists and DOs in the First Part

The expressions and the input/output lists undergo an essential compilation from the source language to the intermediate language. An expression is scanned from left to right and the compilation is done only when, according to the well known operator hierarchy, an operator with precedence lower than the last one met is found. Thus intermediate expressions and formulas, without parentheses, are obtained. These may be of three kinds. The operation signs appear only in those of the first kind and in each of these they have the same hierarchy number except the unary minus. Library or arithmetic statement function names appear only in those of second kind and FORTRAN function names only in those of the third kind.

A mode indicator is associated to each intermediate expression or formula, and it specifies the mode of noninteger variables or functions depending on the expression or formula being of the first or second and third kind.

A list is scanned from left to right and for each left parenthesis, excepting subscript parenthesis, a DO is generated with indexing information given immediately before the matching right parenthesis and with the range extending up to that indexing information. An intermediate statement is produced for each list variable, neither index nor indexing parameter, and a mode indicator is associated to it.

In an expression or list the compilation of the subscripted variables is especially important. The most general form of the subscripted variable S in the source program is:

$$M(a_1*I_1\pm b_1\ ,\ a_2*I_2\pm b_2\ ,\ a_3*I_3\pm b_3)$$

where M is an array name; I_1 , I_2 , I_3 are integer single variables; a_1 , a_2 , a_3 , b_1 , b_2 , b_3 are integer constants without sign. If the array M has precision p, rows d_1 , columns d_2 , layers d_3 , and its elements are stored by columns and by increasing locations, the current address of the subscripted variable S is:

$$M + p(a_1\cdot I_1+a_2\cdot d_1\cdot I_2+a_3\cdot d_1\cdot d_2\cdot I_3) + p[(\pm b_1-1) + (\pm b_2-1)d_1 + (\pm b_3-1)d_1d_2].$$

We call *variable part* and *constant part* of the relative address of the S variable $p(a_1\cdot I_1+a_2\cdot d_1\cdot I_2+a_3\cdot d_1\cdot d_2\cdot I_3)$ and $p[(\pm b_1-1) + (\pm b_2-1)d_1 + (\pm b_3-1)d_1d_2]$, respectively. Moreover we call pa_1 , pd_1a_2 , $pd_1d_2a_3$ *multiplicative constants* of I_1 , I_2 , I_3 , respectively. When a subscript I is not controlled by a DO, a Pseudo-DO (PDO) of index I is created, so that any subscript variable may be considered under the control of a DO or a PDO. A PDO of index I is defined as a DO of index I having (1) only one statement as range, (2) indexing parameters equal to the index, and (3) no statement number in its structure. A DO is translated into table entries and into an intermediate statement whose structure differs from the original one because the statement number of range is replaced by the serial number of the DO in the source program. When a CONTINUE statement occurs or after another statement that closes a DO has been compiled, an intermediate statement of closure of DO is generated. This takes the serial number of the corresponding DO. A PDO is translated into table entries and into an intermediate statement which has the same structure as the original one. After the unique statement in the range of the PDO has been compiled, an intermediate statement of closure of PDO is generated.

4. Some Tables Produced in the First Part

MODT table. A line of MODT contains a letter denoting a mode and the corresponding precision.

SAT table. A line of SAT contains a dummy argument of the SUBROUTINE- or FUNCTION-type subprogram being defined.

FDT table. A line of FDT contains a distinct name of an arithmetic statement function.

FDAT table. A line of FDAT contains a dummy argument of the arithmetic statement function being defined.

SVT table. A line of SVT contains a distinct name of single variable, its precision, and further informations when the single variable is index of a DO.

NET table. A line of NET contains for each DO or PDO: (1) the statement number of the last statement in the range or a mark respectively, (2) the relative address of the line of SVT containing the index, and (3) the serial number z. An entry is made in this table for each DO or PDO of a nest: the relative address of the entry is the same as the *level number* l that starts from 0 increasing or decreasing by 1 at any opening or closing of a DO or PDO of the nest.

DIMT table. A line of DIMT contains a distinct matrix name, and the numbers p, pd_1 , pd_1d_2 , $pd_1d_2d_3$. If the matrix is two- or one-dimensional, d_3 or d_2 and d_3 have value 1.

MET table. A line of MET contains for each distinct subscripted variable: (1) the relative address of the DIMT line containing the name of the array of which the subscripted variable is an element, (2) the relative address of the VPT line concerning the subscripted variable, and (3) the value of the constant part of the relative address of the subscripted variable. Subscripted variables formally equal are considered distinct if contained in different ranges of DO or PDO.

VPT table. A line of VPT contains: (1) the multiplicative constants c_1 , c_2 , c_3 of the subscript variables controlled by DO and/or PDO of levels l_1 , l_2 , l_3 in order of increasing magnitude, and (2) the serial number z of the innermost DO or PDO containing in its range the subscripted variable. If the matrix is two- or one-dimensional c_1 and l_1 or c_1 , c_2 , l_1 and l_2 are zero. An entry is made for each distinct variable part. Two variable parts are considered distinct if they have different lists: c_1 , c_2 , c_3 , l_1 , l_2 , l_3 , z.

DOT table. A line of DOT contains for each DO or PDO: (1) the relative addresses that specify the zone of VPT containing the variable parts concerning the matrix elements in the range of that DO or PDO, and (2) a mark if the current value of the DO index must be saved after a normal exit.

SNT table. A line of SNT contains a statement number and, if the statement labeled by it is in the range of a DO, also the level of this DO.

5. Expressions, Lists and DOs in the Second Part

If the intermediate statement is an expression of the first kind, its translation into a string of symbolic statements is straightforward since every operation once combined with the mode indicator gives the code of the right instruction or pseudo-instruction.

If the intermediate statement is an expression of the second kind, a search in FDT table permits the call of an arithmetic statement function to be distinguished from that of a library function. In the first case there are generated a return-jump to the entry of the subroutine performing the evaluation of the function, and a group of instructions to transmit the actual arguments. In the second case a call is made as below.

If the intermediate statement is an expression of the third kind a group of pseudo-instructions is generated for the subroutine call and for the transmission of the actual arguments [5]. With every dummy argument in each subroutine called the address of a parametric cell of a group, which is connected to the same subroutine, is associated in order to eliminate the prologue [5].

If the intermediate statement concerns input/output of a quantity, a loop on a pseudo-instruction or a pseudo-instruction is generated according to whether the quantity is an array or not. This pseudo-instruction has a code terminating generally with a letter which denotes the mode of the transmitted quantity, and provides for conversion from one numerical system into another, for reading, printing, punching and for transmission between the main and the auxiliary memory.

If the intermediate statement is a DO the instructions for initializing, testing and, if it is the case, calculating the values of the variable parts, are generated. If the intermediate statement is a PDO the instructions for initializing and, if it is the case, calculating the values of the variable parts, are generated. Now, a certain zone of VPT is located by the numbers stored in that line of DOT which corresponds to the DO or PDO just mentioned. When, by scanning that zone, lines are found where l_3 is equal to the level of the DO or PDO, then the above said instructions for evaluation are generated. The calculated values of the variable parts are stored in memory cells whose names are stored in the lines of VPT corresponding to these variable parts.

If the intermediate statement is a closure of DO, the instructions for advancing and jumping to the test instructions of the DO are generated. If the intermediate statement is a closure of PDO, no instruction is produced.

A quantity used in the source program occurs in the instruction or pseudo-instruction which refers to it, as said below.

An instruction or a pseudo-instruction, which operates on a quantity, has the form:

$$L \# C, P, Q, A$$

where the field L contains the symbolic location of the instruction or pseudo-instruction; the field C contains the symbolic code of the instruction or pseudo-instruction; each of the fields P and Q contains the symbolic or numeric address of a parametric cell or, if it is missing, the character "–"; the field A contains a symbolic address, made up by a name followed in case by an integer with sign, or an integer, or else, the character "–".

An integer constant appears without its sign in A. No search in a table is made to obtain such a constant because it appears literally in the intermediate statements.

Every other constant, occurring in an expression, is referred to by its internal name placed in A.

A single variable that is not a dummy argument is referred to by its own name in A. A single variable that is a dummy argument is referred to by the associated parametric cell address symbolically indicated by the own name of the argument if it is listed in a FUNCTION or SUBROUTINE statement.

Searching in SVT and SAT tables allows the variable name used by the programmer and the field where this name is to be placed to be obtained. A subscripted variable is referred to by (1) the own name of the array in P or A according to whether the matrix is or is not a dummy argument, (2) the value of the variable part in a parametric cell in Q and (3) the number, which expresses the value of the constant part, in A after the array name if there is one. The array name, the memory cell where the value of the variable part is stored and the number which expresses the value of the constant part, are obtained by means of searches in MET, DIMT and VPT tables, while the name of the array is regarded as a dummy argument or not by means of searches in SAT.

Acknowledgments. The FORTRAN CEP translator is the result of the work done by O. G. Mancino, M. Morandi Cecchi, L. Spanedda, N. Wolkenstein with the further cooperation of A. Andronico and M. Martelli, in order to modify, complete and program a preceding draft of the FORTRAN CEP translator [6].

RECEIVED DECEMBER, 1963; REVISED NOVEMBER, 1964

REFERENCES

1. MANCINO, O. G. Characteristics of the FORTRAN CEP language. *Comm. ACM 7* (July 1964), 423–424.
2. Manuale delle istruzioni CEP. Centro Studi Calcolatrici Elettroniche, Pisa, Italy, 1960.
3. La programmazione simbolica per la CEP. Centro Studi Calcolatrici Elettroniche, Pisa, Italy, 1961.
4. MANCINO, O. G., AND MORANDI CECCHI, M. The FORTRAN CEP translator. Centro Studi Calcolatrici Elettroniche, Pisa, Italy, 1964.
5. CARACCIOLO DI FORINO, A., MERCATANTI, M., ANDRONICO, A., AND GALLIGANI, I. Struttura generale e chiamata dei sottoprogrammi per la CEP. Proc. Automazione e Strumentazione, Italy, Aug., 1963.
6. CARACCIOLO DI FORINO, A., AND GALLIGANI, I. Sulla soluzione di alcuni problemi relativi alla traduzione di un programma FORTRAN per la CEP. Presented at VI Cong. Automazione e Strumentazione, Italy, 1961.