File Number 7090-25
Form C28-6376

IBM Systems Reference Library

IBM 7090/7094 FORTRAN IV Compiler (IBFTC) Replacement

Specifications and Language Additions

This publication provides the programmer with the information needed to plan for the use of the new and faster 7090/7094 FORTRAN IV Compiler (IBFTC) that will replace the present FORTRAN IV compiler. In almost every respect, the new compiler will operate within the environment of the present IBJOB Processor.

The 7090/7094 FORTRAN IV language will be enlarged to include the following four language features: (1) input/output without an explicit input/output list and FORMAT statement, (2) a maximum of seven dimensions for arrays, (3) nonstandard returns from subroutines, and (4) multiple entry points to a subprogram.

PREFACE

The new 7090/7094 FORTRAN IV Compiler (IBFTC) is designed to be a replacement for the present version of the FORTRAN IV compiler. The new compiler will operate in two passes, enabling it to be faster than the present version. These two passes will be an instruction compilation pass and an assembly pass. The output of the second pass will be input to the Loader (IBLDR). Because assembly will be performed by the compiler, the Macro Assembly Program will not be used. The new compiler will accept the same source program input as does the present version of IBFTC and will produce mathematically equivalent object program coding that is at least as efficient as that produced by the present compiler.

The machine configuration for the new FORTRAN IV compiler will remain the same as for the present version of the 7090/7094 FORTRAN IV Compiler (IBFTC).

It is assumed that the reader is familiar with the contents of the following publications:

IBM 7090/7094 IBSYS Operating System: IBJOB Processor, Form C28-6275-2

IBM 7090/7094 Programming Systems: FORTRAN IV Language, Form C28-6274-1

IBM 7090/7094 Programming Systems: FORTRAN IV Language, Addenda and Errata to Form C28-6274-1, Form N28-0069

The new language feature, input/output without an explicit input/output list and FORMAT statement, is described in the publication IBM 7090/7094 FORTRAN IV Language: Input/Output Without Explicit List and FORMAT, Form C28-6377.

This publication is divided into two sections. The first section contains descriptions of the changes to the IBJOB environment; the second section contains descriptions of the new language features.

Copies of this and other IBM publications can be obtained through IBM Branch Offices.

Address comments concerning the contents of this publication to:
IBM Corporation, Programming Systems Publications,
Dept. D91, PO Box 390, Poughkeepsie, N.Y. 12602

CONTENTS

This publication was prepared for production using an IBM computer for automatic format control and updating. Page proofs for photo-offset printing were produced on an IBM 1403 Printer with an upper-lower case chain of 120 characters.

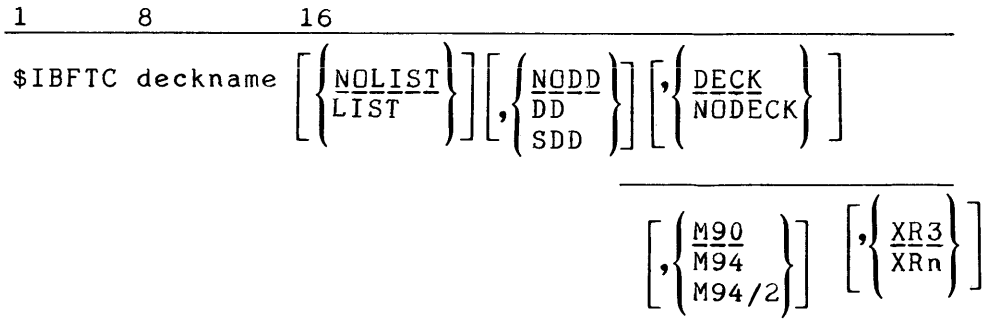
The new 7090/7094 FORTRAN IV Compiler will operate under the IBJOB Processor as described in the publication IBM 7090/7094 IBSYS Operating System: IBJOB Processor, Form C28-6275-2. There will be two changes to the IBJOB environment involving Prest decks and the $IBFTC card. These changes are described in the following text.

Prest Decks

Since the new compiler will not produce symbolic input to the Macro Assembly Program, a Prest deck of the symbolic output from the compiler cannot be obtained for a FORTRAN compilation. Therefore, if the option PREST is specified on the $ODEDIT card, it will be ignored. However, the option CPREST will still be operative. For further discussion of these options, see the section "$ODEDIT Card" in the publication IBM 7090/7094 IBSYS Operating System: IBJOB Processor, Form C28-6275-2.

$IBFTC Card

The format of the $IBFTC card will be:



where deckname identifies the deck that follows. A deck name of six or fewer alphameric characters must be punched in columns 8-13. Characters that cannot be used in the deck name are: parentheses, commas, slashes, quotation marks, equal signs, and blanks.

The variable field starts in column 16. The options in the variable field are described in the following text.

List Options

The list options are:

1. LIST--A listing of the object program, three instructions per line, is generated. Only the relative locations and symbolic information are listed.

2. NOLIST--A listing of the object program is not wanted.

If neither LIST nor NOLIST is specified, a listing is not generated.

Debug Options

The debug options are as follows:

1. NODD--The debugging dictionary is not generated.

2. DD--The full debugging dictionary is generated. All the symbols in the compiled program will appear in the debugging dictionary. For a FORTRAN IV program, this includes all statement numbers, all programmer-specified symbols, and all symbols generated by IBFTC.

3. SDD--The short debugging dictionary is generated. It will contain only the programmer-specified symbols and the statement numbers used in the FORTRAN IV program.

If neither NODD, DD, nor SDD is specified, the debugging dictionary is not generated.

Punch Options

The punch options are as follows:

1. DECK--The object program deck is written on the system peripheral punch unit for off-line punching.

2. NODECK--A punched deck is not wanted.

If neither DECK nor NODECK is specified, the object program deck is written on the system peripheral punch unit.

Instruction Set Options

The instruction set options are as follows:

1. M90--The object program uses only 7090 machine instructions. Any double-precision operations are simulated by system macros, and EVEN pseudo-operations are treated as commentary.

2. M94--The object program uses 7094 machine instructions.

3. M94/2--The object program uses 7094 machine instructions, and EVEN pseudo-operations are treated as commentary.

If neither M90, M94, nor M94/2 is specified, it is assumed that the object program uses only 7090 machine instructions.

Index Register Options

The index register options are as follows:

1. XR3--The object program uses three index registers (1, 2, and 4).

2. XRn--The object program can use up to n index registers, if they are required (n is a number from 4 through 7).

If neither XR3 nor XRn is specified, it is assumed that the object program uses three index registers.

NEW LANGUAGE FEATURES

The 7090/7094 FORTRAN IV language will be enlarged to include four new language features. These are:

1. Input/output and conversion without an explicit input/output list and FORMAT statement. This feature is described in the publication IBM 7090/7094 FORTRAN IV Language: Input/Output Without Explicit List and FORMAT, Form C28-6377

2. A maximum of seven dimensions for arrays.

3. Nonstandard returns from subroutines.

4. Multiple entry points to a subprogram.

Items 2, 3, and 4 are described in the following text.

Arrays with a Maximum of Seven Dimensions

An array may be declared to have a maximum of seven dimensions by placing it in a DIMENSION statement with the appropriate number of subscripts appended to the variable.

General Form
DIMENSION $V_1(i_1),V_2(i_2),\cdots$ where: 1. Each V_n is an array variable, and 2. Each i_n is composed of 1, 2, 3, 4, 5, 6, or 7 unsigned integer constants and/or integer variables, separated by commas. (Integer variables may be a component of i_n only when the DIMENSION statement appears in a subprogram.)

Examples:

```
DIMENSION A(1,2,3,4), B(10)
DIMENSION C(2,2,3,3,4,4,5)
```

In the preceding examples, A, B, and C are declared to be array variables with 4, 1, and 7 dimensions, respectively.

The COMMON statement or one of the Type statements (except EXTERNAL) may also be used to declare arrays with a maximum of seven dimensions.

Nonstandard Returns from Subroutines

The normal sequence of execution following the RETURN statement of a SUBROUTINE subprogram is to the next executable statement following the CALL statement in the calling program. It will also be possible to return to any executable numbered statement in the calling program by using a special return from the called subprogram. This return may not violate DO loop rules.

The following text describes the changes in coding that will be required to return from the subroutine to a statement other than the next executable statement following the CALL.

The general form of the CALL statement in the calling program is:

General Form
CALL SUBR (a_1,a_2,a_3,...,a_n) where: 1. SUBR is the name of the SUBROUTINE subprogram being called, and 2. a_i is a dummy argument of the form described in the publication IBM 7090/7094 Programming Systems: FORTRAN IV Language, Form C28-6274-1, or is of the form: nS where n is a statement number and S is the character S.

The general form of the SUBROUTINE statement in the called program is:

General Form
SUBROUTINE SUBR (a_1,a_2,a_3,...,a_n) where: 1. SUBR is the name of the subprogram, and 2. a_i is a dummy argument of the form described in the publication IBM 7090/7094 Programming Systems: FORTRAN IV Language, Form C28-6274-1, or is of the form: * where * is the character asterisk (*) and denotes a nonstandard return.

The general form of the RETURN statement in the called program is:

General Form
RETURN or RETURN i where: i is an integer constant or variable whose value, n, denotes the nth nonstandard return in the argument list, reading from left to right.

Example:

Calling Program

```
      .
      .
      .
10 CALL SUB (A,B,C,30S,40S)
20     ---
      .
      .
      .
30     ---
      .
      .
      .
40     ---
      .
      .
      .
      END
```

Called Program

```
      SUBROUTINE SUB (X,Y,Z,*,*)
           .
           .
           .
100   IF (R) 200,300,400
200   RETURN
300   RETURN 1
400   RETURN 2
          END
```

In the preceding example, execution of statement 10 in the calling program causes entry into subprogram SUB. If statement 100 is executed, the return to the calling program will be to statement 20, 30, or 40, if R is less than, equal to, or greater than zero, respectively.

Nonstandard returns may be best understood by showing that a CALL statement that uses the nonstandard return is equivalent to a CALL and a computed GO TO statement in sequence. For example,

```
CALL NAME (P,20S,Q,35S,R,22S)
```

is equivalent to

```
CALL NAME (P,Q,R)
GO TO (20,35,22),I
```

where the index I is set according to the value of the integer in the RETURN statement executed in the called subprogram. If the RETURN is blank or zero, a normal (rather than nonstandard) return is made to the statement immediately following the GO TO.

Similarly, the arguments in the associated SUBROUTINE statement will correspond to the arguments in the CALL statement as follows:

```
SUBROUTINE NAME (S,*,T,*,U,*)
```

Multiple Entry Points into a Subprogram

The normal entry into a SUBROUTINE subprogram from the calling program is by a CALL statement that references the subprogram name. The normal entry into a FUNCTION subprogram is made by a function reference in an arithmetic expression. Entry is made at the first executable statement following the SUBROUTINE or FUNCTION statement.

It will also be possible to enter a subprogram by a CALL statement or a function reference that references an ENTRY statement in the subprogram. Entry is made at the first executable statement following the ENTRY statement.

ENTRY statements are nonexecutable and, therefore, do not affect control sequencing during normal execution of a subprogram. The order, type, and number of arguments need not agree between the SUBROUTINE or FUNCTION statement and the ENTRY statements, nor do the ENTRY statements have to agree among themselves in these respects. Each CALL or function reference, however, must agree in order, type, and number with the SUBROUTINE, FUNCTION, or ENTRY statement that it references. No subprogram may reference itself directly or through any of its entry points.

The general form of the ENTRY statement in the called subprogram is:

General Form
ENTRY Name (B_1,B_2,...,B_n) where: 1. Name is the symbolic name of an entry point, and 2. Each B_i is a dummy argument corresponding to an actual argument in a CALL statement or in a function reference.

Example:

Calling Program

```
      .
      .
      .
1 CALL SUB1 (A,B,C,D,E,F)
      .
      .
      .
2 CALL SUB2 (G,H,P)
      .
      .
      .
3 CALL SUB3
      .
      .
      .
     END
```

Called Program

```
      SUBROUTINE SUB1 (U,V,W,X,Y,Z)
            .
            .
            .
   10 U = V
            .
            .
            .
      ENTRY SUB2 (T,U,V)
      GO TO 10
            .
            .
            .
      ENTRY SUB3
            .
            .
            .
          END
```

In the preceding example, the execution of statement 1 causes entry into SUB1, starting with the first executable statement of the subroutine. Execution of statements 2 and 3 also causes entry into the called program, starting with the first executable statement following the ENTRY SUB2 (T,U,V) and ENTRY SUB3 statements, respectively.

Additional Rules for Entry Points

The following rules also apply to entry points:

1. If an adjustable array name or any of its adjustable dimensions appears in an argument list for a FUNCTION, SUBROUTINE, or ENTRY statement, that array name and all its adjustable dimensions must appear in that argument list.

2. A dummy argument may not appear in any statement unless it previously appeared in an argument list of a FUNCTION, SUBROUTINE, or ENTRY statement.

3. In a FUNCTION subprogram, only the FUNCTION name may be used as the variable to carry a result back to the calling program. The ENTRY name may not be used for this purpose.

4. An ENTRY name may appear in an EXTERNAL statement in the same manner as a FUNCTION or SUBROUTINE name.

5. Entry into a subprogram initializes all references in the entire called subprogram from items in the argument list of the CALL or function reference. (For instance, if, in the example that appeared in the preceding text in this section, entry is made at SUB2, the variables in statement 10 will reference the argument list of SUB2.)

6. ENTRY statements may appear only in subprograms.

7. The appearance of an ENTRY statement does not alter the rules regarding the placement of Arithmetic Statement Functions in subroutines. Arithmetic Statement Functions may follow an ENTRY statement only if they precede the first executable statement following the SUBROUTINE or FUNCTION statement.

Printed in U.S.A. C28-6376

IBM

International Business Machines Corporation
Data Processing Division
112 East Post Road, White Plains, N.Y. 10601