PL/I → FORTRAN via INVOKE

CGTM# 46

C.T. Zahn

5 May 1968

PL/I → FORTRAN via INVOKE

Motivation for a PL/I → FORTRAN interface program

A program (INVOKE) has been written which allows a PL/I programmer to use occasional FORTRAN subprograms for reasons of convenience, efficiency etc. The following list enumerates the principal reasons for constructing such a program:

1. The programmer who uses primarily PL/I gains access to the substantial library of FORTRAN routines available. The lack of such a library for PL/I may be discouraging potential users of PL/I from adopting that language.
2. The programmer who intends to or is in the process of converting FORTRAN subprograms over to PL/I can use these programs in their FORTRAN form until the reprogramming is complete. This might reduce the temptation to produce a "FORTRAN program written in PL/I" and encourage a true exploitation of the PL/I programming facilities.
3. Special purpose programs written in Assembly Language to be callable by FORTRAN are usable by PL/I just as if they were part of the FORTRAN library.
4. When a PL/I programmer requires an extra degree of computational efficiency in a relatively small segment of his program he may program it in FORTRAN and compile it under H level 2. For a programmer not well-versed in Assembly Language this may be a very attractive alternative.

How to use INVOKE

The calling sequence for INVOKE is

```
CALL INVØKE ( F4SUB,          Fortran function or entry name
              CØDES,          Array of bytes to identify arg. types
              ARG1,
              ARG2,
              .
              .
              .
optional      ARGn,           Arguments for F4SUB
optional      FVALUE,         Return value of Function
              LAB1,
              LAB2,
              .
              .
              .
optional      LAB_p )         Abnormal return labels
```

For $1 \leq i \leq n$ where F4SUB has n arguments

CØDES(i) specifies ARGi as one of the following:

a) simple arithmetic element (SAE)
b) STRING
c) arithmetic array (AA)
d) fixed length character string array (FLCSA)
e) fixed length bit string array (FLBSA)
f) ENTRY

If F4SUB is a function subprogram then CØDES(n+1) specifies the type of the returned value, FVALUE, as one of:

a) INTEGER
b) REAL
c) COMPLEX
d) LOGICAL

If F4SUB contains any abnormal exits RETURNj and f = 1,0 according as F4SUB is or isn't a function, then CØDES(n+f+j) = LABEL for $1 \leq j \leq p$,

specifying the actual argument LABj as corresponding to the FORTRAN statement RETURNj .

Finally CØDES(n+f+p+1) = LAST indicates no more arguments.

The following PL/I declaration defines the codes:

```
DECLARE
     LAST         BIT(8) INITIAL ('00000000'B),
     SAE          BIT(8) INITIAL ('00000100'B),
     STRING       BIT(8) INITIAL ('00001000'B),
     (AA,FLCSA)   BIT(8) INITIAL ('00001100'B),
     FLBSA        BIT(8) INITIAL ('000'0000'B),
     LABEL        BIT(8) INITIAL ('00010100'B),
     ENTRY        BIT(8) INITIAL ('00011000'B),
     INTEGER      BIT(8) INITIAL ('00011100'B),
     REAL         BIT(8) INITIAL ('00100000'B),
     COMPLEX      BIT(8) INITIAL ('00100100'B),
     LOGICAL      BIT(8) INITIAL ('00101000'B),
     NDIM FIXED BINARY (8);
```

An exception occurs when an argument is an array. Whenever CØDES = AA or FLCSA or FLBSA the next code, CØDES(k+1), is an 8-bit string representing the binary form of the number of dimensions of the array in question. The variable NDIM of precision 8-bits should be assigned the integer number of dimensions and then NDIM assigned as the new value of CØDES.

The example below serves to illustrate more concretely the correct use of INVOKE.

Suppose we wish to employ the following FORTRAN subroutine:

```
SUBROUTINE  SUB(A,B,*,*)
     REAL A
     INTEGER B(3)
     A = 2 * B(1) - 1.0
     IF ( B(2) .LT. 0) RETURN 1
     IF ( B(2) .GT. 0) RETURN 2
     RETURN
     END
```

Normally SUB would be used by another FORTRAN program as follows:

```
C           MAIN PRØGRAM TØ CALL SUB
            REAL X
            INTEGER Z(3)
               .
               .
               .
            CALL SUB(X,Z,&10,&25)
            (regular return)
               .
               .
               .
 10         (return 1)
               .
               .
               .
 25         (return 2)
               .
               .
               .
            END
```

The analogous PL/I program to call SUB would be

```
MAIN:     PRØCEDURE ØPTIØNS (MAIN);
          DECLARE   X FLØAT BINARY (21),
                    Z (1:3) FIXED BINARY (31);

          DECLARE   CØDES (1:6) BIT(8),
                    SAE  BIT(8) INITIAL ('00000100'B),
                    AA   BIT(8) INITIAL ('00001100'B),
                    LABEL  BIT(8) INITIAL ('00010100'B),
                    LAST BIT(8) INITIAL ('00000000'B),
                    NDIM FIXED BINARY (8);
          DECLARE   SUB ENTRY;
          CODES(1) = SAE; CODES(2) = AA; NDIM = 1; CODES(3) = NDIM;
          CODES(4),CODES(5) = LABEL; CODES(6) = LAST;

          CALL INVØKE(SUB,CØDES,X,Z,MINUS,PLUS);

ØKAY:     (regular return)
               ;

MINUS:    (return 1)
               ;

PLUS:     (return 2)
               ;

          END MAIN;
```

Rules for PL/I → FORTRAN calls

1. The calling program is responsible for making sure that any CONTROLLED variable passed as parameter is currently allocated. This can be tested conveniently using the builtin function ALLØCATIØN(X) .

2. The following data types cannot be passed as parameters to a FORTRAN program:

   a) structures
   b) arrays of varying length strings
   c) array cross-sections
   d) interleaved arrays (in structures)
   e) complex data not double-word aligned (see final section)

3. In general, PL/I output written on FILE(SYSPRINT) and FORTRAN output written into data set 6 will appear as a single stream on the printer.

4. An array parameter $A(\ell_1:u_1, \ell_2:u_2, \ldots, \ell_n:u_n)$ in PL/I corresponds to a FORTRAN array

$AA(u_n-\ell_n+1, \ldots, u_2-\ell_2+1, u_1-\ell_1+1)$ as a result of different storage schemes in the two languages. A method for overcoming this superficial incompatibility is described in the final section.

5. The following is a reasonably complete list of the valid pairings of PL/I arguments and FORTRAN formal parameters:

PL/I	FORTRAN
FIXED BINARY(31)	INTEGER*4
(none yet)	INTEGER*2
FLOAT BINARY(21)	REAL*4
FLOAT DECIMAL(6)	"
FLOAT BINARY(53)	REAL*8
FLOAT DECIMAL(14)	"
COMPLEX FLOAT BINARY(21)	COMPLEX*8
COMPLEX FLOAT BINARY(53)	COMPLEX*16
BIT(8)	LOGICAL*1
BIT(32)	LOGICAL*4
ENTRY	(declared by use)
CHARACTER(1)	LOGICAL*1
CHARACTER(n)	LOGICAL*1 array (n)
Array of above	Array of above

Summary of INVOKE

INVOKE is an IBM/360 Assembly Language program which allows a PL/I procedure to call a FORTRAN subprogram while imposing a minimum of restrictions.

This program begins with a standard prologue for PL/I procedures and hence looks just like a PL/I program to the PL/I environment. Making INVOKE look like PL/I involves little more than calling the PL/I subroutine library program IHESADA.

The next portion of INVOKE accesses the array of 8-bit codes which has been passed as the second argument and uses each code to identify the type of the corresponding argument to be passed to the FORTRAN routine. When the type of an argument has been identified certain adjustments are made to the argument list to conform with FORTRAN conventions. When a returned value is expected from the called FORTRAN routine the type is indicated by one of the codes and INVOKE can find it when the call is complete.

When the argument list has been made acceptable to FORTRAN a SPIE macro is issued which tells the system supervisor that subsequent program interruptions will be handled by the FORTRAN error-handling package. INVOKE then calls the program whose entry point was passed as the first argument.

On return from the invoked routine the returned value (if any is expected) is picked up from the appropriate register and assigned as the value of the argument whose code indicated return-value (i.e. INTEGER, REAL, COMPLEX, LOGICAL).

A new SPIE macro is issued which delegates error-handling to the PL/I program IHEERRA as was the case on entry to INVOKE.

Register 15 is examined to determine if an abnormal exit from the called routine was taken and if so control is transferred to the appropriate label variable in the calling PL/I program using the PL/I library program IHESAFC. A normal return is handled through IHESAFB.

Special Problems and Suggested Solutions

1. Buffer Flushing

When the FORTRAN program called from PL/I issues commands to send print lines to an output data set associated with SYSOUT=A and the PL/I program is also sending print lines to the file SYSPRINT the eventual print-stream may show some lines out-of-order because the two programming environments have separate I/O buffering systems feeding into the same printer stream.

One way of getting around this problem is to flush the FORTRAN (PL/I) buffers before leaving the FORTRAN (PL/I) environment by issuing twice a blank line with the no-space carriage control. In FORTRAN one could repeat the following twice.

```
      WRITE(6,7)
    7 FORMAT('+')
```

In PL/I the statements

```
PUT SKIP(0); PUT SKIP(0);
```

would probably work.

In PL/I the statement

```
CLOSE FILE (SYSPRINT);
```

would certainly flush the buffers but will take longer than simply outputing overprinted blank lines.

2. Alignment of Complex Data

FORTRAN expects a COMPLEX*4 quantity to be aligned on a double-word boundary whereas PL/I may start it on a word boundary

which doesn't happen to be a double-word boundary. This will result in a specification interrupt during the prologue of the called FORTRAN program.

We have chosen to issue (in INVOKE) the SPIE macro which does not call for the FORTRAN error handler to deal with boundary alignment errors. There is however a SPIE macro which indicates FORTRAN will force boundary alignment and INVOKE could be changed to issue that macro. Then complex arguments could be passed regardless of where PL/I chooses to put them.

3. Passing Arrays to FORTRAN

The correspondence defining feature of PL/I affords a method of passing PL/I array arguments to FORTRAN subroutines with convenience and efficiency. Let us suppose that a PL/I program contains an array declared by

```
DECLARE  A (1:3,1:5) FLOAT BINARY(21);
```

The PL/I program wants to call a FORTRAN program which has as array parameter B specified by the statements

```
INTEGER NDIM
REAL B(NDIM,5)
```

If the following steps are taken the array A can be passed successfully to the FORTRAN program.

a) replace the declaration of A by the declarations

```
DECLARE
    AT (1:5,1:3) FLOAT BINARY(21),
    A (1:3,1:5) FLOAT BINARY(21)
        DEFINED AT (2SUB,1SUB);
```

b) insert the following statement in the PL/I calling program before the actual call

```
NDIM = 3;
```

c) call the FORTRAN program (via INVOKE) passing NDIM and AT as arguments.

This method has the advantage that the PL/I program can access

the array using the same index specification as in FORTRAN. In other words, A(I,J) in PL/I corresponds to B(I,J) in FORTRAN.

Furthermore, the access to A(I,J) and AT(J,I) take essentially the same amount of computation because the multipliers are simply in reverse order in the array-access algorithm.