CONSULTING

CGTM # 51
J. Ehrman
July 18, 1968

Fortran H Goodies Revisited (or, Son of Fortran H)

There are many useful functions hiding in the Fortran H compiler that can be used by daring and resourceful coders to give that added boost in performance to their programs. In the version I compiler (up to release 13), these functions were available merely by naming them: they are described in CGTM 41 (see reference 1). This, however, led to the problem that there were hidden reserved words in the Fortran language, which is of course unacceptable to the common everyday user. In view of this restriction on users of this compiler, IBM decided to remove these functions from the second version of the compiler (releases 14 and after). (Due to an oversight, they were still there in the release 14 version, but have been removed from later releases.) Does this mean that such wonderful functions are lost forever? No! Superkludge comes to the rescue!

Because the H compiler itself is compiled in Fortran H with heavy use of the hidden goodies, and because nobody expects that the release 15 version is bugproof, it is clear that there must be some means to recompile the compiler without at the same time burdening the merely mortal programmer with a host of extraneous names he must avoid. An extensive analysis of core dumps, old listings, and scraps of paper snitched from wastebaskets reveals that there is indeed a way to make use of the functions listed in Table 1: include "XL" among the parameters in the PARM field when invoking the compiler. I repeat the warning from CGTM 41:

> THERE IS NO ASSURANCE THAT THE HIDDEN FUNCTIONS IN FORTRAN H WILL NOT IN FUTURE BECOME SO WELL HIDDEN AS TO BE INVISIBLE; AND FURTHERMORE, ANYONE WHO USES THEM SHOULD UNDERSTAND THAT HE IS COMPLETELY ON HIS OWN, SINCE THERE IS NO COMMITMENT ON THE PART OF IBM OR SLAC TO PROVIDE LIBRARY ROUTINES TO PERFORM THESE FUNCTIONS IN THE EVENT THAT THEY ARE REMOVED FROM THE COMPILER. IF YOU REALLY NEED THEM, AND YOUR PROGRAM WILL BE MODIFIED REGULARLY, USE THEM IN A SUBROUTINE THAT CAN BE EASILY RECODED IN MACHINE LANGUAGE IF THE SPECIAL FUNCTIONS

VANISH. DURING DEBUGGING, USE THE LIST OPTION TO CHECK THE GENERATED CODE. THIS IS YOUR LAST WARNING.

First, we will tabulate the functions available, and then discuss the STRUCTURE statement, which is of great use in constructing lists and other data containing addresses as data.

TABLE I

NAME	TYPE	ARGUMENTS	ACTION	
LAND	I*4	any*4,any*4	Logical AND	
LOR	I*4	any*4,any*4	Logical OR	
LXOR	I*4	any*4,any*4	Logical EXCLUSIVE OR	
LCOMPL	I*4	any*4	Bitwise Complement	
AND	R*4	any*4,any*4	Logical AND	
OR	R*4	any*4,any*4	Logical OR	
COMPL	R*4	any*4	Bitwise Complement	
MOD24	any*4	any*4	Clear High-order Byte (mod 2**24)	
SHFTL	I*4	any*4,I*4	Shift Left Logical	
SHFTR	I*4	any*4,I*4	Shift Right Logical	
TBIT	note(1)	any,Integer	see note(1)	
BITON	note(2)	any,Integer	see note (2)	Set Bit On (to 1)
BITOFF	note(2)	any,Integer	see note (2)	Set Bit Off (to 0)
BITFLP	note (2)	any,Integer	see note (2)	Invert Bit

Notes:

(1) The TBIT function can be used to give either a Logical or an Integer result. For example, one can write K=TBIT(X,5) and K would be set to 0 or 1 depending on whether the 5th bit after the address of X is 0 or 1. To use the logical "value" of the TBIT function, one can write IF(TBIT(X,5))GO TO 7 and the code generated would consist of a TM followed by a BC. The same applies to IF(.NOT. TBIT(X,5))GO TO 7, so that efficient bit-tests and branches can be coded this way.

(2) To obtain correct code from these three functions, one must write the same variable on the left-hand side of the assignment statement as is used for the first argument. That is, write A(J) = BITON(A(J),3). No assignment is made to the variable other than the implied bit manipulation.

The notation used in the table is as follows:

I*4	fullword integer (for arguments, variables or constants)
R*4	short real
any*4	any fullword-aligned quantity
any	any variable name
Integer	an integer constant, e.g. 5

The STRUCTURE statement (the analogue of the ALGOL W RECORD statement) allows the programmer to make use of base-displacement addressing in a very natural way. Suppose there is a block of data in memory which has its own data layout; it is cleaner to be able to refer to the components by name rather than as part of a larger array containing the entire workspace. If there is a way in Fortran to obtain addresses as data types, then structured variables can be used. This of course implies that a machine-language routine must be used to establish the addresses to be used as base addresses, since it is not possible to generate addresses as values of variables under normal circumstances.

The syntax of the STRUCTURE statement is

STRUCTURE // list-of-variables // list-of-variables // etc.

The double slashes have the effect of setting the displacement to zero; then as the variables in the list are scanned, the displacement is incremented by the lengths of the variables. This naturally implies that ALL VARIABLES APPEARING IN STRUCTURE STATEMENTS MUST BE PREDEFINED. For example, the statements

```
LOGICAL * 1     BYTE, MARKER
INTEGER * 2     ISN, TYPE, MODE
INTEGER  CHAIN,  VALUE

STRUCTURE // BYTE // CHAIN, TYPE, MODE // VALUE
STRUCTURE // MARKER
```

define blocks in which the first byte has the names BYTE and MARKER, the first fullword has the names CHAIN and VALUE, and the third and fourth halfwords have the names TYPE and MODE.

To refer to a structured variable, one must have available an integer variable whose value is the address of the beginning of the block to be referenced, and then use it as a subscript for the structured variable.

For example,

```
NPTR = IADDR(X(258))
IF (TBIT(BYTE(NPTR),7) TYPE = TYPE + 4
```

would test the value of one of the bits in BYTE and modify TYPE accordingly, assuming that the function IADDR returns the address of the desired block which is assumed to lie at X(258). It is a syntax error to use a structured variable without a subscript. Similarly, J=MODE(NPTR) would retrieve the appropriate halfword from the structure whose address is in NPTR.

References:

1) Computation Group Technical Memorandum #41: "Hidden Goodies in Fortran H Version 1", G.A. Robinson and J.R.Ehrman, March 1968

2) SLAC Building L Room 2 Trash Can on July 13, 1968.

3) Fortran H Compiler PLM.