North American Response Centers

HP 3000 APPLICATION NOTE #15

FORTRAN LANGUAGE CONSIDERATIONS

A GUIDE TO COMMON PROBLEMS

HEWLETT PACKARD

October 1, 1986
Document P/N 5958-5824/2640

HP 3000 APPLICATION NOTES *are published by the North American Response Centers twice a month and distributed with the Software Status Bulletin. These notes address topics, where the volume of calls received at the Centers indicates a need for addition to or consolidation of information available through HP support services. Notes to date include:*

Note #	*Published*	*Topic*
1	2/21/85	*HP 3000 Printer Configuration Guide (superceeded by note #4)*
2	10/15/85	*Terminal Types for HP 3000 HPIB Computers (superceeded by note #13)*
3	4/01/86	*HP 3000 Plotter Configuration Guide*
4	4/15/86	*HP 3000 Printer Configuration Guide - Revised*
5	5/01/86	*MPE System Logfile Record Formats*
6	5/15/86	*HP 3000 Stack Operation*
7	6/01/86	*Cobol II/3000 Programs: Tracing Illegal Data*
8	6/15/86	*KSAM Topics: Cobol's Index I/O; File Data Integrity*
9	7/01/86	*Port Failures, Terminal Hangs, TERMDSM*
10	7/15/86	*Serial Printers - Configuration, Cabling, Muxes*
11	8/01/86	*System Configuration or System Table Related Errors*
12	8/15/86	*Pascal/3000 - Using Dynamic Variables*
13	9/01/86	*Terminal Types for HP 3000 HPIB Computers - Revised*
14	9/15/86	*Laser Printers - A Software and Hardware Overview*
15	10/01/86	*Fortran Language Considerations - A Guide to Common Problems*

FORTRAN LANGUAGE CONSIDERATIONS

A Guide to Common Problems

FORTRAN has been with us for a while, and has seen several revisions. In addition to the ANSI standard changes, Hewlett-Packard has provided extensions which give the programmer access to some powerful features on HP 3000 computer systems.

As with any existing product, there are several areas which bear occasional examination, to enable the programmer to produce a better, more timely, and more cost-effective result - that being the application program. This Application Note discusses some commonly asked questions as well as some frequently encountered problems, including:

- Address representation and variable initialization.
- Conversion from REAL to INTEGER.
- Compiler symbol table.
- Local Addressability.
- The FORTRAN Logical Unit Table (FLUT) and I/O.

Unless explicitly specified, references in this note to FORTRAN apply equally to FORTRAN/3000 and FORTRAN 77.

VARIABLE REPRESENTATION AND INITIALIZATION

FORTRAN/3000 integer and double integer items are stored in two's complement format. The range of a single-word integer is -32,768 to 32,767. For a double integer, the range is -2,147,483,648 to 2,147,483,647.

FORTRAN/3000 real numbers are stored in two words, comprised of three fields. These are the sign bit, exponent, and mantissa. The exponent is an integral power of two, applied to the fractional mantissa. The format is "excess 256", which is that the exponent is biased by 256 (%400). The true exponent is found by subtracting the bias of 256 from the stored exponent, and can range from -256 to +255. The mantissa is 22 bits long, and allows a range of .863617*(10**-77) to .1157920*(10**78). Double precision items follow the same format, but allow 54 bits for the mantissa. This brings the range to .8636168555094445*(10**-77) to .1157920892373161*(10**78).

It's important to remember that the internal representation of a floating-point (real or long) is a signed binary number raised to some power of *two*. This can make examination of floating-point numbers using DEBUG fairly difficult. In addition, these values are stored according to the HP 3000 floating point format, *not* the ANSI floating point format. More information about HP 3000 floating point format can be found in the *Machine Instruction Set Reference Manual.*

A logical variable is a 16 bit unsigned quantity. These can be used as parameters to system Intrinsics, masks for logical operations, or in their normal two states. The states are .TRUE. (all bits set to 1) or .FALSE. (all bits set to 0). FORTRAN, unlike SPL, does not provide logical (unsigned) arithmetic.

Character values are represented in standard ASCII code, with two bytes per computer word. Character, integer, and logical types can be addressed using substring and partial word designators. Further information on these can be found in the *FORTRAN/3000 Reference Manual*, chapter III. This chapter also covers constant definition for the various types, and the internal representations of all types, graphically depicted.

FORTRAN 77 supports the same data structures, with slight changes in nomenclature. Single-word integers and logicals are "short integer" and "short logical". The requirement of specifying this for all variables can be overridden with the compiler $SHORT directive.

The data structure format for FORTRAN 77 types is described in the *FORTRAN 77 Reference Manual*, Appendix E, pages 51 and following.

By default, neither FORTRAN/3000 nor FORTRAN 77 provide variable initialization. However, both compilers do provide facilities for initializing all program variables to known values at the start of the run. If you include $CONTROL INIT (FORTRAN/3000) or $INIT ON (FORTRAN 77), all numeric data will be set to zero and all character items will be set to ASCII null (%00). Note that character variables are *not* initialized to blanks (%40). The FORTRAN 77 manual notes that the use of $INIT ON may reduce the portability of the program.

To initialize variables to specific non-zero values, use the DATA statement. A BLOCK DATA subprogram and statements are available for initializing variables contained in labelled COMMON blocks. These constructs are defined in all FORTRANs and therefore do not reduce program portability.

CONVERSIONS FROM REAL TO INTEGER

Since an integer contains no decimal places, conversion from floating-point format to integer format must dispose of the fractional part of the floating point number in some way. The programmer must either truncate (discard) the fractional portion or round the real number to its closest integer.

FORTRAN does not provide automatic rounding on conversion from real to integer; such conversions always truncate the fractional part of the real. This can cause unexpected results when certain real numbers are converted to integer. The reason is that, as described above, a floating point number is some value raised to a power of two. This may not be *exactly* eqaul to the intended decimal result. When used in calculations, such numbers may cause small errors of precision to be introduced, which can result in anomolies such as 5.1 - 0.1 = 4.99999. (This happens because 5.1 cannot be represented exactly as a number raised to a power of two.) On assignment to an integer, however, the result (4.99999) is truncated by dropping the fractional part, leaving the integer value 4!

This can be corrected in all (positive) cases by adding 0.5 to the value before assignment. Using the example above, we see that 4.99999 + 0.5 = 5.49999, which is truncated to yield 5. If the value were 4.49999, adding 0.5 yields 4.999999, which truncates to 4.

For negative numbers, application of this algorithm would round the number toward zero (-3.9999 + 0.5 = -3.4999 = integer -3). If this is not the desired rounding, you must *subtract* 0.5 from the real value before the integer assignment.

SYMBOL TABLE

Compiles of very large FORTRAN programs may fail with SYMBOL TABLE OVERFLOW. This internal compiler table is used to keep track of all programmer-assigned names (symbols): variables, subroutines, functions, intrinsics, and labels. In addition, DO loops and certain compiler options require entries in the table. Each entry requires 4 words + 1 word for each two characters in the name. In FORTRAN/3000, the table is 8191 words long, and it is 'recycled' for each program unit.

Overflow can be addressed by making the program more modular (incorporate more subroutines), as the space is recycled for each subroutine or function subprogram. Other ways of reducing usage are reducing the length of variable and array names, use of arrays instead of simple variables, avoidance of compiler

options, and the aforementioned use of subroutines. Refer to the *FORTRAN/3000 Reference Manual*, Appendix F, for more information on the symbol table and program optimization in general.

FORTRAN 77 uses different structures to achieve the same result. There are a symbol table and string table. These are dynamically allocated, along with other internal structures. The design of compiler greatly reduces the likelihood of an overflow condition, and the error reported will probably be STRING TABLE OVERFLOW. The same corrective actions can be taken as in a symbol table overflow in FORTRAN/3000.

LOCAL ADDRESSABILITY EXCEEDED

In the HP 3000's stack architecture, register Q points to the beginning of the current procedure's local data. This register is used as the 'base register' for addressing all local simple variables and arrays, as well as for addressing the procedure's parameters. The address range provided by the instruction set permits direct addressing of memory locations Q-63 through Q+127. (Procedure parameters are always in Q- locations, whereas procedure-local data exists in the Q+ area.) Since the compiler must allocate one Q+ location for each local data element, it is possible to overflow the address range of the Q register. When this occurs, the compiler prints the message LOCAL ADDRESSABILITY EXCEEDED and flushes the program unit. (Before this error occurs, the compiler attempts to recover by placing some simple variables in the unused 0th element of arrays and by creating an artificial array to hold other simple variables. This array is usually pointed to by Q+127, indirect.)

If you receive this error, it's possible to recover by reducing the number of labelled COMMON, DATA statements, arrays, equivalences, and directed GOTOs. Another option is to place some of the arrays, or items being equivalenced, into labelled COMMON. This reduces the number of Q relative locations being used by the number of items put into COMMON, while adding only one entry for the labelled COMMON block. As always, there is a tradeoff. This will require DB-relative, or global, storage, and reduce the available global storage. The ideal solution would be to reduce the number of items requiring Q relative storage.

FORTRAN LOGICAL UNIT TABLE and I/O TECHNIQUES

When you use FORTRAN's READ, WRITE, and FORMAT statements to perform I/O to and from logical units, the FORTRAN logical unit table (FLUT) is used to correlate a logical unit to an MPE file system file number. The default options taken when FORTRAN opens a file are defined in the *FORTRAN/3000 Reference Manual* on page 8-2.

The most frequent source of trouble has to do with the parameters supplied in the compiler-generated FOPEN. For units other than 5 (input) and 6 (output), most FOPEN parameters are omitted, which results in the creation of a new binary disc file with a record size of 128 words and a capacity of 1023 records in 8 extents. This file is opened for exclusive read/write access and is deleted when it is closed. Of course, any or all of these characteristics may be overridden with a :FILE command. This command must be issued before any statement which references the specified unit is executed. (Exception: any :FILE command issued for units 5 or 6 must be executed before the program is run, as these units are opened as soon as the program starts.) These :FILE commands can be issued by a UDC or job stream prior to the RUN command (for all files), or can be issued through use of the COMMAND intrinsic (for all units except 5 and 6).

Alternatively, you can use the MPE intrinsics (declared as SYSTEM INTRINSIC) to perform I/O to disc and device files. FOPEN allows the programmer to specify non-default access and file options at open, eliminating the need for an explicit FILE command. Greater control over I/O can also be obtained, and you can use optional file system features, such as NOWAIT I/O. The READ and PRINT intrinsics

specifically address input from and output to $STDIN and $STDLIST, eliminating the need to FOPEN these files. These methods do, however, have the drawback of tying the program closely to the HP 3000 and MPE operating system, and due consideration needs to be given to future growth needs.

Once opened via FOPEN, the file can be accessed via FREAD and FWRITE or it can be assigned to a FORTRAN unit number with FSET, allowing you to use FORTRAN's READ and WRITE statements. Conversely, you can obtain the MPE file number of an open FORTRAN logical unit using FNUM. This allows you to intermix FORTRAN READs and WRITEs with MPE intrinsic calls, providing greater file handling flexibility. For example, you can use FCONTROL to set read timeouts, change parity, or change the termtype of a terminal port. These are covered in the *MPE V Intrinsics Reference Manual* under FCONTROL. Further discussion is available in the *MPE File System Reference Manual.*

Certain characteristics of the file can be manipulated with the UNITCONTROL procedure. These procedures are discussed in the *FORTRAN/3000 Reference Manual* in chapter 8. They are also covered in the *FORTRAN 77 Reference Manual* on page E-44 and following. FORTRAN 77 also provides an OPEN statement, which increases flexibility in the opening of a file. There are options available, and the file can be opened at a time determined by the programmer. With FORTRAN/3000, the files are opened when first accessed, units 5 and 6 excepted.

As in all cases where MPE-specific options and extensions are used, the programmer must be aware that the program is deviating from ANSI standard and becoming tied to the HP 3000. Portability of code is thus limited.

The FORTRAN logical unit table is not accessible to the application programmer except for the FSET and FNUM procedures. This table is simply an array, where each entry is a pair containing the unit number in the first byte and the MPE file number in the second byte. It is illustrated and discussed in the *FORTRAN/3000 Reference Manual*, page 8-1, paragraph 8-2.

CONCLUSIONS

While not necessarily comprehensive, this note has described issues which are fairly commonly encountered. Further discussion of these and other issues can be found in the manual appropriate to the version of FORTRAN you use. You may wish to refer to comparisons of FORTRAN/3000 and FORTRAN 77, the appropriate ANSI standards, and other HP FORTRAN implementations, such as the HP 1000. This note should provide quick information on more commonly encountered problems, and assists in structuring your program to best utilize your computer.