TIPS ON CONVERTING IBM FORTRAN PROGRAMS TO THE HP 3000

BY

GARY ANDERSON AND DEEPAK SINHA

McMASTER UNIVERSITY

A. INTRODUCTION

This paper will be useful to anyone wishing to embark on the task of converting IBM FORTRAN software to the HP 3000 Series II or Series III computers. It should also be helpful to anyone who wishes to consider the general question of program portability to the HP 3000.

The material presented here is based primarily on the experience of the authors resulting from the successful conversion of the BMDP and SPSS statistical systems to HP 3000 Series II.

The problems encountered during these projects arose largely from the following four sources:

1) Incompatibilities between IBM/FORTRAN and HP 3000/FORTRAN.

2) Architectural features of HP 3000 Series II computer which impose certain restrictions on programs it can run.

3) Difference between the EBCDIC and ASCII character sets.

4) Difficulties with some library functions on the HP 3000.

The following sections attempt to cover the above problem areas and our proposed solutions in detail.

B. FORTRAN incompatibilities between IBM and the HP 3000

B.1) Type declaration order:

The order in which type declarations can be made is more restrictive on the HP 3000 than on IBM. All type declarations must be made before any DATA statements on the HP 3000. Further, the data declarations cannot be interspersed with type specifications. For example, consider the following FORTRAN code for IBM and its equivalent on the HP 3000:

```
IBM                            HP 3000

SUBROUTINE EX1                 SUBROUTINE EX1
 .                              .
REAL B(3)/1.,2.,3./,A,D        REAL B(3), A,D
INTEGER*2 I, J/9/,K            INTEGER*2 I,J,K
DATA K/5/,A/4.5/               INTEGER*4 INTI, INT2
INTEGER*4 INTI, INT2           DATA B/1.,2.,3./
 .                             DATA J/9/, K/5/, A/4.5/
 .                              .
                                .
END                            END
```

B.2) REAL and INTEGER specifications:

IBM FORTRAN allows REAL*8 and REAL*4 type specifications. On the HP 3000 REAL*8 must be replaced by DOUBLE PRECISION and REAL*4 simply by REAL wherever they occur.

The use of INTEGER*4 and INTEGER*2 specification on IBM is compatible with the HP 3000 and needs no change.

It must be pointed out that the default integer size on IBM is 32 bits, i.e. any variable specified as INTEGER*4, INTEGER, or any integer constant (e.g. 1,2,99, etc.) has a length of 32 bits. On the HP 3000, the default integer length is 16 bits. This incompatibility can be easily removed, however, by including the $INTEGER*4 command ahead of the source-code before compilation. Consider the following equivalent examples on the two machines.

```
IBM                       HP 3000

PROGRAM EX2               $INTEGER*4
REAL*8 A,B                PROGRAM EX2
REAL*4 C,D                DOUBLE PRECISION A,B
INTEGER I,X               REAL C,D
INTEGER *4 J,K            INTEGER I,X
INTEGER *2 L,M,N          INTEGER*4 J,K
 .                        INTEGER*2 L,M,N
 .                         .
J=K+10                     .
 .                        J=K+10
 .                         .
END                        .
                          END
SUBROUTINE XYZ            SUBROUTINE XYZ
INTEGER I,J,A             INTEGER I,J,A
 .                         .
 .                         .
END                       END
                          SUBROUTINE ABC
SUBROUTINE ABC            INTEGER X,Y
INTEGER X,Y                .
 .                         .
 .                        END
END
```

The lengths of the REAL and DOUBLE PRECISION variables on both the machines are 32 bits and 64 bits respectively. (Remember, though, that the length of DOUBLE PRECISION variables on the HP 3000 CX machine is 48 bits).

B.3) Mixed specifications:

IBM FORTRAN allows double precision and real variables to be declared in the same statement by appending *2 or *4 to the variable name. The IBM convention of appending *(number) to a variable or function name is totally unacceptable on HP 3000. Consider the following equivalent examples:

```
    IBM                             HP 3000

REAL FUNCTION EX3*8(N)       FUNCTION EX3(N)
REAL *8 A,B*4,C              DOUBLE PRECISION A,C,EX3
INTEGER I,J*2,K*4            REAL B
 .                           INTEGER*4 I,K
 .                           INTEGER*2 J
END                           .
INTEGER FUNCTION INT*2(I)     .
 .                           END
 .                           FUNCTION INT(I)
END                          INTEGER*2 INT
                              .
                              .
                             END
```

B.4) Logical variables:

There are two kinds of logical variables in IBM FORTRAN; One byte logical variables, which are typed using the LOGICAL*1 declaration, and 4 byte logical variables, which are typed using the LOGICAL declaration. All logical variables in HP 3000 FORTRAN have a length of 2 bytes.

LOGICAL*1 variables or arrays in IBM programs are often used to store character strings only, and logical tests are not performed on them. In this case, these variables or arrays can simply be typed as CHARACTER*1 on HP 3000 and they become exactly equivalent.

When LOGICAL*1 and LOGICAL variables or arrays are being used in the logical context (i.e. logical tests are being performed on them and they are set to TRUE or FALSE, etc.) then there are clearly two options:-

a) Declare them as LOGICAL on HP 3000, but with caution. What if those variables or arrays are equivalenced with some other arrays? Or, what if they are a part of some other big array, the starting address being passed through a subroutine call? Clearly, the IBM calculations for the space required by the array will need readjustment in HP 3000 programs. Also, since the lengths of IBM LOGICAL, REAL and INTEGER words are the same, an array declared as REAL or INTEGER in one subroutine can be declared as LOGICAL and interpreted logically in another subroutine and vice-versa. This operation is obviously invalid on the HP 3000 because of the length difference of a logical variable.

b) Declare a LOGICAL*1 variable as CHARACTER*1 and a LOGICAL variable as an INTEGER or REAL. All the logical tests, initializations and assignment statements will then have to be changed. Consider the following equivalent examples for this case:

```
  IBM                  HP 3000

PROGRAM EX4         PROGRAM EX4
LOGICAL*1 A,B       CHARACTER A,B,CTRUE, CFALSE
LOGICAL X,Y         INTEGER X,Y,ITRUE, IFALSE
IF (A) G=G+1        DATA CTRUE/%1C/, CFALSE/%0C/
IF (X) Y=FALSE      DATA TRUE/1/, IFALSE/0/
B = FALSE           IF (A.EQ. CTRUE) G = G + 1
 .                  IF (X. EQ. ITRUE) Y = IFALSE
 .                  B = CFALSE
END                  .
                     .
                    END
```

Another fact to note is that the bit representation for the constant TRUE is different on the two machines:

```
          IBM                                    HP 3000

Bit  0   1     30   31                   Bit  0   1      14   15
    [0 | 0|...|0 |  1 ] (=1) TRUE            [1 | 1|...|1 |  1 ](=-1)
    [0 | 0|...|0 |  0 ] (=0) FALSE           [0 | 0|...|0 |  0 ] (=0)
```

In view of the above facts it is not possible to give a pat solution for every problem arising out of the use of LOGICALs but it is recommended that every situation involving these variables should be examined carefully.

B.5) Hexadecimal constants:

IBM FORTRAN allows initialization of variables using hexadecimal constants (e.g. Z18005024 etc.). They are not allowed in HP 3000 FORTRAN and must, therefore, be replaced by octal or some other equivalent constant.

B.6) Branching control in subroutine calls:

The character "&" in an IBM subroutine call statement is used to control branching on return from the subroutine. It must be replaced by "$" on HP 3000. Consider the examples:

```
         IBM                     HP 3000

      PROGRAM EX5             PROGRAM EX5
      DATA A/Z12345678/       DATA A/%0221505317R/
      CALL SUB (A,B, &10)     CALL SUB (A,B, $10)
       .                       .
       .                       .
   10 I = I + 1           10 I = I + 1
       .                       .
       .                       .
      END                     END
```

B.7) Type incompatibilities:

IBM FORTRAN allows incompatibility between the types of the actual arguments (those provided in a CALL statement) and the dummy arguments (those within a subroutine). For example, a REAL argument can be passed to a subroutine whose corresponding dummy argument is INTEGER. The HP 3000 allows such incompatibilities only if a $CONTROL CHECK=2 statement has been included ahead of the subroutine before compilation. The following equivalent examples further clarify this point.

```
         IBM                     HP 3000

      PROGRAM EX6             PROGRAM EX6
      REAL A                  REAL A
      CALL SUB (A)            CALL SUB (A)
       .                       .
       .                       .
      END                     END
      SUBROUTINE SUB (I)      $CONTROL CHECK = 2
      INTEGER*4 I             SUBROUTINE SUB (I)
                              INTEGER*4 I
       .                       .
       .                       .
      END                     END
```

B.8) Array bounds:

A dynamic array bound must be a dummy argument of the subroutine statement in HP 3000 FORTRAN. Two equivalent examples are given below and some comments are made:

```
IBM                              HP 3000

SUBROUTINE EX7 (A,B)             SUBROUTINE EX7 (A,B)
REAL A(N), B(M), C(L)            REAL A(1), B(1), C(1)
COMMON/ABC/M,N                   COMMON/ABC/M,N
READ (5,10) A                    READ (5,10) (A(I), I=1,N)
WRITE (6,11) B                   WRITE (6,11) (B(I), I=1,M)
 .                                .
 .                                .
ENTRY DUMB (C,L)                 ENTRY DUMB (C,L)
 .                                .
 .                                .
END                              END
SUBROUTINE XYZ (P,I)             SUBROUTINE XYZ (P,I)
REAL P(I)                        REAL P(I)
 .                                .
 .                                .
END                              END
```

It should be noted in the above examples that subroutine XYZ needed no modification because I, the dynamic bound of array P, is a dummy argument of subroutine XYZ; but M, N and L were not dummy arguments of subroutine EX7 and hence could not be used for dimensioning A, B, and C. Note that L is a dummy argument of entry DUMB but that is not sufficient. The implicit length of the arrays A, B, and C was changed to 1 but this required the modification of all the READs, WRITEs or any other statements using the implicit length of the arrays.

B.9) Basic external functions:

All the basic external functions in the HP 3000 FORTRAN, particularly the double precision functions, require explicit typing in a program. For example, DSQRT must be declared as DOUBLE PRECISION in the HP program, but this is not necessary in an IBM program.

B.10) Scientific subroutine library functions:

Certain IBM scientific subroutine library routines like GAMA, DGAMA, LGAMA and DLGAMA are available on HP 3000 but under the names GAMMA, DGAMMA, LGAMMA and DLGAMMA respectively. (Note the two M's in the spellings).

B.11) Length of common blocks:

IBM FORTRAN allows the length of a labelled (or named) common block to vary from one subroutine to another. HP 3000 FORTRAN permits this phenomenon only for a single blank (or unnamed) common block. All the labelled common blocks must be of the same length in every subprogram or else a segmenter error results. The following equivalent examples contain one proposed solution.

```
IBM                          HP 3000

PROGRAM EX9                  PROGRAM EX9
COMMON/A/A,B,C(10)           COMMON/A/A,B,C(10), PAD (90)
 .                            .
 .                            .
END                          END
SUBROUTINE ONE (I,J,K)       SUBROUTINE ONE (I,J,K)
COMMON/A/P                   COMMON/A/P, PAD (101)
 .                            .
 .                            .
END                          END
SUBROUTINE TWO (X)           SUBROUTINE TWO (X)
COMMON/A/A,Y,Z(100)          COMMON/A/A,Y,Z(100)
 .                            .
 .                            .
END                          END
```

In order to determine the maximum length of a common block, the source should be compiled with the $CONTROL MAP, CROSSREF ALL option. This enables one to know which surbroutines use a particular common block and what the lengths are of the common blocks in those subroutines.

C. RESTRICTIONS DUE TO HP 3000 SYSTEM ARCHITECTURE

C.1) Variables in COMMON or DATA:

The total number of variables declared in different COMMON blocks or DATA statements must not exceed 255 on HP 3000. The reason for this is that the compiler places the address of each such variable or array in the primary DB area of the stack and the DB relative addressing is not allowed to exceed 255 words by the system.

The problems imposed by this restriction have proven to be extremely difficult, especially in the conversion of a big system like SPSS.

One solution is:

a) Eliminate a common block by including all or some of its variables in the argument list of the affected subroutines.

b) Eliminate a DATA declaration by initializing its variables using assigment statements in the code. This process, however, cannot be carried out indiscriminately. There is one very important consideration to keep in mind when comtemplating this change. A DATA variable is like a global variable, in other words it retains its value throughout the program execution. For example, suppose that a subroutine, when entered once, set the value of some variable A; now, when it was entered again, if A was not a global variable, its value would be indefinite. By being removed from a DATA declaration, the status of a variable is changed from global to local. Hence, before removing a variable from a DATA statement one needs to understand the program logic to ascertain whether or not this particular variable needs to be global. Only those DATA variables, not required to be global, may be initialized by assignment statements.

It is the experience of the authors that this process can be tedious, time consuming and has the potential for introducing an unending string of "bugs" in a program.

A new capability likely to be available in the future version of the FORTRAN compiler may remove this restriction, at the expense of execution time, by providing the $MORECOM compiler option.

C.2) Subroutine arguments:

The maximum number of arguments in a single subroutine on HP 3000 cannot exceed 54. The reason for this limitation is that whenever a subroutine or function is called, the address or values of the actual arguments and a four word stack marker are placed on the stack and Q-minus addresses are assigned to them. Q-minus addressing cannot exceed 63 words, hence the limit.

C.3) Local variables:

On an IBM machine, the local variables within a subprogram retain their values between calls to this subprogram, unless this subprogram happens to be overlaid with another subprogram. In particular, programs with only one overlay always retain the values for local variables throughout a run of the program. This is never true on HP 3000 because the system stack is dynamically increased when a subroutine is called and decreased when the subroutine is exited. Consequently, all the local variables are lost with the updating of Q register.

The only solution to this problem is to understand the program logic so as to determine which variables need to be removed from the list of local variables and be made global. Once identified, these variables must be placed in a common block.

C.4) Addressing:

IBM and many other machines use a byte oriented addressing scheme. This means that given an address, the system can identify the proper byte in memory which may or may not be at a word boundary. HP 3000 system uses related but different addressing schemes for words and bytes. The right most bit of a byte address indicates whether it is the left or the right byte of the word whose address is given by the remaining bits. The implication is that given an address, the system also has to know whether it is a word or byte address.

The HP 3000 FORTRAN compiler generates word addresses for REAL, INTEGER, DOUBLE PRECISION or LOGICAL variables but byte addresses for CHARACTER type variables or strings. For this reason, it is not possible in HP 3000 FORTRAN, as opposed to IBM FORTRAN, to pass a character string or variable in a subroutine CALL statement when the corresponding dummy argument is not of a CHARACTER type. The converse also holds true.

One can use equivalencing of variables to solve this problem but a more innovative solution, and one that has been used in the SPSS and BMDP-77 conversion projects, is to use an SPL program to modify the address in question and pass this modified address to the called subroutine. The following examples should clarify this concept.

```
IBM                         HP 3000

PROGRAM EX10                PROGRAM EX10
REAL A                      REAL A
CALL SUB1 (A)               CALL SUBIM(A)
 .                           .
 .                           .
END                         END
SUBROUTINE SUB1 (B)         SUBROUTINE SUB1(B)
LOGICAL*1 B(1)              CHARACTER B(1)
 .                           .
 .                           .
END                         END
                            $CONTROL SUBPROGRAM
                            BEGIN
                            PROCEDURE SUB1(B);
                            BYTE ARRAY B;
                            OPTION EXTERNAL;
                            PROCEDURE SUBIM(B);
                            REAL B;
                            BEGIN TOS:= @ B & LSL(1);
                            SUB1(*); END;
                            END.
```

Conversion of a word address to a byte address is straight forward but while converting a byte address to a word address one must note that if the byte address is not at a word boundary then there is no way it can be converted to a word address. However, such a situation rarely arises.

C.5) Code segmentation:

As opposed to specifying overlays on an IBM system, one needs to define code segments on HP 3000. All the code must belong to some code segment. There is an allowed maximum size and an allowed maximum number of code segments which are fixed at the time of system configuration.

The implication is that there is a limit to how large a program that runs on HP 3000 can be and how large a subprogram can be. If a subroutine, or any other subprogram cannot fit within one code segment then it must be split up into two or more subprograms. The task of splitting subroutines is a difficult one and generally introduces "bugs" in the program. We have found that a code segment size of 8K bytes will generally handle the largest of FORTRAN subroutines or programs.

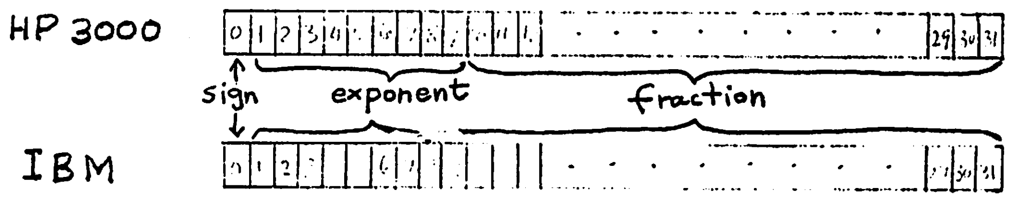
C.6) Data stack limitation:

The entire data stack used by a HP 3000 program must not exceed 32,767 words. This restricts the size of scratch areas and other arrays in a program. In order to use the stack judiciously, one must have as few global variables and arrays as possible so that there is more space available for local variables. There is a permanent allocation of stack for the global variables but a dynamic allocation for the local variables. This is one reason why a lot of DATA declarations in SPSS had to be eliminated.

It may be noted that in order to use the maximum stack size, the program should be prepped or run with STACK parameter set to 819 and MAXDATA parameter set to 32767.

C.7) Real word configuration:

The exponent and fraction parts of a real word on the two machines are as shown here:



In addition, on the HP 3000, a 1 is always implied to left of the binary point. Real 0 is the only exception to this rule.

It should be obvious that the range of magnitude for the real numbers is more on HP but the precision is smaller by one bit compared to IBM.

C.8) Distinction between BLANK and ZERO:

The IBM formatter upon detecting blanks in a field being read using Fw.d specification, turns the left most bit of the real word ON (the remaining bits are 0). This real word, while being distinct from +0, has an arithmetic value equal to 0. The HP 3000 formatter returns a +0 upon detecting blanks under the same conditions. Hence there is no way of distinguishing between blanks and zeroes while reading a field under Fw.d specification.

It may be noted that even if one, somehow, manages to turn the left most bit ON, the value of the real word will not be -0, it will be -.863617 E-77 because of the biased exponent and an implied 1 to the left of the binary point.

D. INCOMPATIBILITIES DUE TO EBCDIC & ASCII CHARACTER SETS

D.1) Alphabetic and numeric tests:

Very often, in order to determine whether a character is alphabetic or numeric, its numerical value is tested. As one would expect, the numerical values for the EBCDIC character set used by IBM are different from those for the ASCII character set used by the HP 3000. The following table compares some of the values assuming that the character in question occupies the right most byte of the word.

	EBCDIC	ASCII
A - Z	193 - 233	65 - 90
0 - 9	240 - 249	48 - 57
BLANK	64	32

The following equivalent examples include one instance of such a test.

IBM

```
    PROGRAM EX11
    LOGICAL *1 A(4)
    INTEGER *4 I
    EQUIVALENCE (I,A)
    DATA I/4Hbbb9/
    IF (I.GE.240 AND
    I.LE.249) GO TO 10
     .
     .
10  INUM = I - 240

    END
```

HP 3000

```
    PROGRAM EX11
    CHARACTER A(4)
    INTEGER *4 I
    EQUIVALENCE (I,A)
    DATA I/4Hbbb9/
    IF (I.GE.48 AND
    I.LE.57) GO TO 10
     .
     .
10  INUM = I - 48
     .
     .
    END
```

D.2) Alphabetic sorting:

Imagine a real or double precision array which needs to be sorted in alphabetic order. One method is to compare the numerical values of the words and take the appropriate action. The left most bit of every EBCDIC alphabet or numeral is 1, which in the position of a sign bit makes a number negative. The implication is that the value of A is greater than the value of B etc. In ASCII character set the left most bit of alphabets and numerals is 0 which implies that the numerical value of A is less than the numerical value of B etc. Hence, the logic of all the tests, performed on EBCDIC character strings to sort them in an ascending alphabetical order, will have to be reversed to sort ASCII character strings in the same order.

E. PROBLEMS WITH HP 3000 SCIENTIFIC SUBROUTINE LIBRARY FUNCTIONS

E.1) DFLOAT:

This double precision function, on HP 3000, always returns a zero regardless of the input. One has to supply his or her own DFLOAT function in the program or replace DFLOAT(I) by DBLE(FLOAT(I)) type of conversion.

E.2) LGAMMA:

This function on HP 3000 aborts when the input is smaller than 7.0. One way to get around this problem is to use DLGAMMA which seems to work correctly.

E.3) Initialization of DOUBLE PRECISION words:

As the HP 3000 FORTRAN manual suggests, it is not possible to initialize a double precision word using %"ABCDEFGH"D form. Only the first four characters of the string are stored. To get around this problem, one has to equivalence the double precision word with real words and initialize the real words.

F. CONCLUSION

Although there are many potential areas of incompatibilities between IBM FORTRAN code and that on the HP 3000, the potential gains from converted software can make the effort well worthwhile. There is a great wealth of well written and thoroughly debugged software on IBM systems which will run on the HP 3000 once converted. We feel that our effort in converting the SPSS and BMDP-77 systems, for example, has been an extremely successful one. In fact, we are actively pursuing other FORTRAN software packages to convert to the HP 3000 as we have found that programs compiled from FORTRAN seem to run surprisingly efficiently on the HP 3000 computer system.

G. ACKNOWLEDGMENTS

The authors would like to acknowledge the contributions of the members of the software group in the Computation Services Unit at McMaster and their assistance in finding solutions to the conversion problems described in this document. Their general support role in making these two rather massive conversion projects successful is also greatfully acknowledged. Dr. Khursheed Ahmed's assistance in the initial assessment of the projects and in the conversion design as well as his ongoing support have been most helpful. Mr. Kim Clark has proved to be a rich source of useful suggestions. Maria Wong has provided valuable input all through the projects with user interfacing problems and with testing and improving the final products.

Lastly, the authors would like to acknowledge the support of Hewlett Packard, SPSS Inc. and the personnel of the Health Sciences Computing Facility at UCLA.

H. REFERENCES

1) Anderson, G. D., "Converting IBM 360 and 370 FORTRAN to the HP 3000 Series II" Journal on the HP 3000 Users Group, Vol 1, #3, Sept./Oct. 1977.

2) Anderson, G. D., "BMDP Program Conversion to the Hewlett Packard System 3000 Computer" Internal document available from Department of C.E.B., McMaster University, Hamilton, Ontario, L8S 4J9.

3) HP 3000 Series II Computer System, FORTRAN Reference Manual.

4) IBM System/360 and System/370 FORTRAN IV Language Reference Manual.