900 Series HP 3000 Computers

HP FORTRAN 77/iX Migration Guide

HP Part No. 31501-90004
Printed in U.S.A. June 1992

Second Edition
E0692

Print History

The following table lists the printings of this document, together with the respective release dates for each edition. Many product releases do not require changes to the document. Therefore, do not expect a one-to-one correspondence between product releases and document editions.

Edition	Date	Software Version
First Edition	November 1987	31501A.01.04
Second Edition	June 1992	31501A.04.31

.

Preface

This manual explains how to run FORTRAN 66/V and HP FORTRAN 77/V programs on the MPE/iX operating system and how to covert them to HP FORTRAN 77/iX programs. It is written for experienced FORTRAN programmers.

This manual contains the following chapters:

Chapter 1	Explains terminology used in this manual. Explains migration to compatibility mode versus migration to native mode.
Chapter 2	Discusses the scope of Part I and describes the FORTRAN 66/V to HP FORTRAN 77/V Migration Aid.
Chapter 3	Describes the conversions performed by the migration aid.
Chapter 4	Describes changes that the migration aid cannot make.
Chapter 5	Explains how to run the migration aid and includes a sample migration.
Chapter 6	Describes how to modify the migration aid according to your needs.
Chapter 7	Describes the scope of Part II
Chapter 8	Compares HP FORTRAN 77/iX to HP FORTRAN 77/V, outlining the changed, missing, and new features.
Chapter 9	Explains changes to an HP FORTRAN 77/V program that may be required before it can run properly on the MPE/iX system.
Chapter 10	Describes data file conversions that allow you to take full advantage of MPE/iX performance improvements.
Chapter 11	Summarizes factors affecting migration and offers tips for avoiding migration problems.

Additional Documentation

This manual does not discuss the MPE/iX operating system in detail. See the appropriate operating system or language manual for complete information about those subjects. The following is a partial list of the operating system and language manuals:

Manual Title	Number to Use to Order Manual
HP FORTRAN 77/iX Reference	31501-90010
HP FORTRAN 77/iX Programmer's Guide	31501-90011
HP Link Editor/iX Reference Manual	32650-90030
HP FORTRAN 77 Programmer's Guide	5967-4686
HP FORTRAN 77 Quick Reference Guide	5957-4687
HP FORTRAN 77 Reference Manual	5957-4685
HP FORTRAN 77/V Programmer's Guide Supplement	31501-90005
HP FORTRAN 77/V Reference Manual Supplement	30000-90294
FORTRAN/3000 Referecne Manual	30000-90040
HP FORTRAN 77 Self Study Guide	22999-90548
Supplement to the HP FORTRAN 77 Self Study Guide	22999-90549
MPE/iX Commands Reference Manual	32650-60002
MPE/iX Intrinsics Reference Manual	32650-90028
MPE/iX Utilities Reference Manual	32033-90008

Conventions

`CASE`

In a syntax statement, commands and keywords are shown in uppercase and lowercase characters. The characters must be entered in the order shown; however, you can enter the characters in either uppercase or lowercase. For example:

```
SHOWJOB
```

can be entered as any of the following:

```
showjob        Showjob        SHOWJOB
```

It cannot, however, be entered as:

```
shojwob        Shojob         SHOW_JOB
```

italics

In a syntax statement or an example, a word in italics represents a parameter or argument that you must replace with an actual value. In the following example, you must replace *filename* with the name of the file:

`RELEASE` *filename*

Italics font is also used to emphasize a *word* or *words*.

punctuation

In a syntax statement, punctuation characters (other than brackets, braces, vertical bars, and ellipses) must be entered exactly as shown. In the following example, the parentheses and colon must be entered:

(*filename*) : (*filename*)

underlining

Within an example that contains interactive dialog, user input and user responses to prompts are indicated by underlining. In the following example, "yes" is the user's response to the prompt:

```
Do you want to continue? >>  yes
```

`{ }`

In a syntax statement, braces enclose required elements. When several elements are stacked within braces, you must select one. In the following example, you must select either `ON` or `OFF`:

```
SETMSG { ON  }
       { OFF }
```

Commands listed in braces are called *command lists* throughout this manual.

Conventions (continued)

[ ]

In a syntax statement, brackets enclose optional elements. In the following example, `,TEMP` can be omitted:

`PURGE` *filename*`[,TEMP]`

When several elements are stacked within brackets, you can select one or none of the elements. In the following example, you can select *devicename* or *deviceclass* or neither. The elements cannot be repeated.

`SHOWDEV` [*devicename* / *deviceclass*]

[...]

In a syntax statement, horizontal ellipses enclosed in brackets indicate that you can repeatedly select the element(s) that appear within the immediately preceding pair of brackets or braces. In the example below, you can select *itemname* zero or more times. Each instance of *itemname* must be preceded by a comma:

`[,`*itemname*`] [...]`

In the example below, you only use the comma as a delimiter if *itemname* is repeated; no comma is used before the first occurrence of *itemname*:

`[`*itemname*`] [,...]`

| ... |

In a syntax statement, horizontal ellipses enclosed in vertical bars indicate that you can select more than one element within the immediately preceding pair of brackets or braces. However, each particular element can only be selected once. In the following example, you must select `A`, `AB`, `BA` or `B`. The elements cannot be repeated.

{ `A` / `B` } `| ... |`

... ⋮

In an example, horizontal or vertical ellipses indicate where portions of the example have been omitted.

Δ

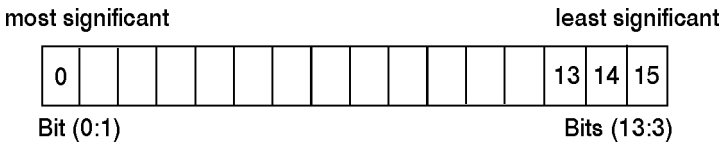
In a syntax statement, the space symbol `Δ` shows a required blank. In the following example, *modifier* and *variable* must be separated with a blank:

`SET [(`*modifier*`)]Δ(`*variable*`);`

[ ]	The symbol [ ] indicates a key on the keyboard. For example, [RETURN] represents the carriage return key.
[CNTL]*char*	[CNTL]*char* indicates a control character. For example, [CNTL]Y means you press the control key and the Y key simultaneously.

Conventions (continued)

Comment	Explains an operator entry or debug message.
>	The HP Symbolic Debugger prompt.
|	Represents "or".
;	Separates commands in a command list.
base prefixes	The prefixes %, #, and $ specify the numerical base of the value that follows: *%num* specifies an octal number *#num* specifies a decimal number *$num* specifies a hexadecimal number If no base is specified, decimal is assumed.
`Bits` (*bit:length*)	When a parameter contains more than one piece of data within its bit field, the different data fields are described in the format `Bits` (*bit:length*) *bit* is the first bit in the field and *length* is the number of consecutive bits in the field. For example, `Bits (13:3)` indicates bits 13, 14, and 15:



Contents

Figures

Tables

1

Introduction

This guide explains how to run FORTRAN 66/V and HP FORTRAN 77/V programs on the MPE/iX oprerating system and how to convert them to HP FORTRAN 77/iX programs.

Terminology

This guide discusses three versions of FORTRAN:

- The version of FORTRAN 66 that runs under MPE V and under MPE/iX in compatibility mode is called FORTRAN 66/V.
- The version of FORTRAN 77 that runs under MPE V and under MPE/iX in compatibility mode is called HP FORTRAN 77/V.
- The version of FORTRAN 77 that runs under MPE/iX in native mode is called HP FORTRAN 77/iX.

Direct Migration of Object Code (Compatibility Mode)

The fastest way to move your programs from an MPE V system to MPE/iX is to simply transfer the object code from one system to the other. Your programs can then be run on MPE/iX in compatibility mode. This works for programs produced by either the FORTRAN 66/V or HP FORTRAN 77/V compiler. If there are data files associated with your program, you can move them also (without needing to change the representation of the data).

Use the MPE V STORE command to transfer the files onto magnetic tape. Next, on the MPE/iX system, use the RESTORE command to move the files from the tape. You can now use the RUN command to run your program just as you would on the MPE V system. Note, however, that programs run in compatibility mode do not take advantage of the optimizing compiler nor of the improved performance capabilities of HP's Precision Architecture. To take advantage of these improvements, you must run your programs in native mode, which is explained in the next section.

Source Code Migration (Native Mode)

Another way to move your programs from an MPE V system onto the MPE/iX operating system is to make necessary modifications to your source code and then recompile and create new native mode program files. Your programs can then be run in native mode. Programs run more quickly and efficiently on the MPE/iX operating system in native mode than in compatibility mode.

If your program is written in FORTRAN 66/V, we recommend that you first convert from FORTRAN 66 to HP FORTRAN 77, using the migration aid and making the manual changes described in Part I of this guide. Then make the manual changes described in Part II to convert the HP FORTRAN 77/V source code to HP FORTRAN 77/iX source code. If your program is already written in HP FORTRAN 77/V, you can convert it to HP FORTRAN 77/iX source code by making manual changes only.

Migration Paths

Figure 1-1 and Figure 1-2 show each type of program and its path to compatibility mode and native mode.

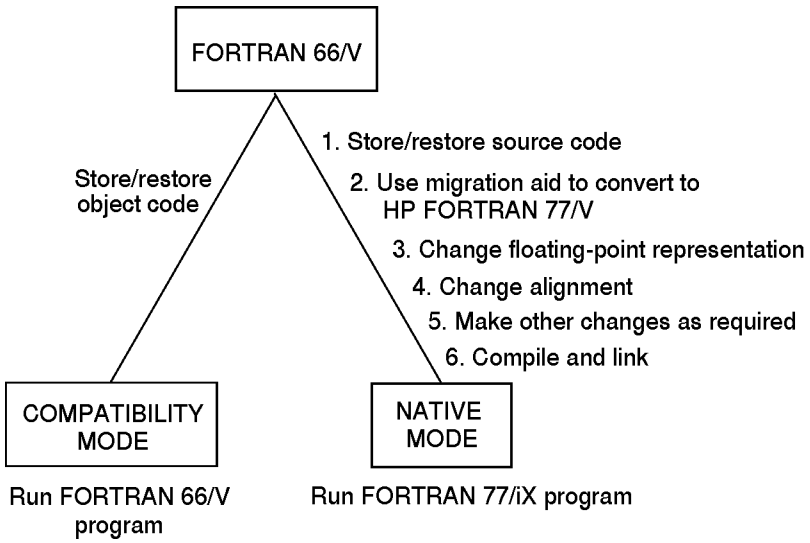


Figure 1-1. Migration Paths from FORTRAN 66/V to HP FORTRAN 77/iX

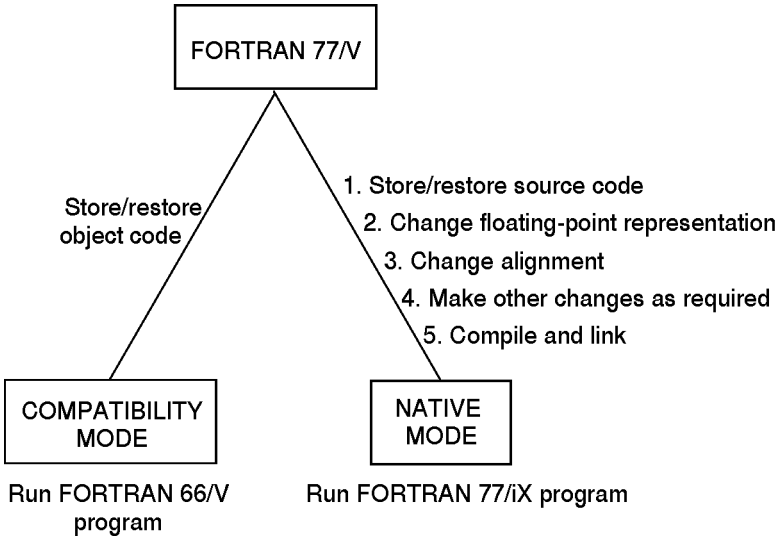


Figure 1-2.
Migration Paths from HP FORTRAN 77/V to HP FORTRAN 77/iX

Finding Information on the Migration Paths

This manual contains two parts:

Part 1 (Chapters 2 through 6) describes migration of FORTRAN 66/V programs to HP FORTRAN 77/V (that is, FORTRAN 66 to FORTRAN 77), both running under MPE V. If your programs are written in FORTRAN 66/V, it's best to first convert them to HP FORTRAN 77/V and then to HP FORTRAN 77/iX. The FORTRAN 66/V to HP FORTRAN 77/V Migration Aid, described in Part I, performs many conversions automatically.

Part II (Chapters 7 through 11) describes migration of HP FORTRAN 77/V programs to HP FORTRAN 77/iX. Skip to Part II if you are migrating directly from HP FORTRAN 77/V to HP FORTRAN 77/iX.

2

Introduction to Part I

Part I compares FORTRAN 66/V to HP FORTRAN 77/V and explains how to use the FORTRAN 66/V to HP FORTRAN 77/V Migration Aid. It also describes manual changes your programs may require.

The Migration Aid

Several changes must be made to a FORTRAN 66/V program before it can be compiled by the HP FORTRAN 77/V compiler. The FORTRAN 66/V to FORTRAN 77/V Migration Aid allows you to make many of these changes automatically. The migration aid reads a FORTRAN 66/V source file, performs a number of conversions, and puts the results in a new file. The new file can then be processed by the HP FORTRAN 77/V compiler. At compilation, constructs that differ in HP FORTRAN 77/V and FORTRAN 66/V but are not detected by the migration aid cause the compiler to generate a warning or an error message. You can then manually change the source file in those places.

You can also customize the migration aid so that it automatically performs specific changes your programs require.

3

What the Migration Aid Does

The migration aid is a powerful editor that accepts commands from a file, applies those commands to a source file, and generates an output file containing the appropriate changes. The migration aid can make many changes, which are described below. In addition, you can easily modify the command file to make specific changes your programs require (see the chapter "Customizing the Command File"). However, since the migration aid is an editor and not a compiler, it does not recognize symbols, variables, and expressions as a compiler does. Therefore, certain changes must be made manually; these are described in the next chapter ("What the Migration Aid Does Not Do").

How the Migration Aid Works

The migration aid consists of a program file (`FTNCVT.PUB.SYS`) and a second file (`FTNCMDS.PUB.SYS`) containing all the commands that could potentially be applied to a source file. Each group of related commands in the command file contains optional text and the commands themselves. The following is an example of a group of commands that affect character constants wherever they are found, except in `FORMAT` statements.

```
$ The next changes will affect character constants of the form %nnC.
$ They will be converted to CHAR(nn) except if they are found on the
$ same line as a FORMAT statement.
$ Note that numbers of the form %nn,%nnJ,%nnL,%nnR,%nnD won't be
$ changed.
n/@format@% *[0-7]+ *c//&/
g/% *{[0-7]+} *c//CHAR(&1B)/
```

The lines starting with dollar signs explain the changes performed if the command group is selected. See Chapter 6 ("Customizing the Command File") for an explanation of the command syntax. The lines without dollar signs contain the commands themselves.

When the migration aid is run, the following prompt occurs after each explanation:

```
apply these commands? (expected Y,N,A)
```

These are the valid responses:

`Y`	Performs the described conversion wherever the construct occurs in the source file.
`N`	Performs no conversions of the kind described. (A carriage return assumes this response.)
`A`	Asks the user before performing each conversion.
`E`	Exits the program without performing any conversions of the kind described.

(Note that both uppercase and lowercase are accepted.)

No conversions are done until you have responded to all command prompts. Then, for each group of commands to which you responded `Y`, the migration aid applies the commands to all appropriate lines and displays each converted line, followed by the line with its changes.

If you selected `A` (ask) for a group of commands, the migration aid displays the original line, then the line with the changes made so far, and prompts you with:

```
apply it? (expected Y,N)
```

The line is then changed (or not) according to your answer.

The output file contains the source file with the changes performed by the migration aid. The lines that were changed are left as comments (a `C` is inserted at column 1), and the label `MIGF66` is placed in columns 73 through 77. The label `MIGF77` is placed in columns 73 through 77 of new lines. This label identifies each line that has been modified or added by the migration aid. Note that the converted file will be 10 to 20 percent larger than the original, since the old versions of the converted lines are retained as comments; the actual percentage of increase depends on the number of lines converted.

See the chapter "Using the Migration Aid" for a complete example of the migration process.

The Commands of the Command File

The following are the commands contained in the command file `FTNCMDS.PUB.SYS`. They are listed in the order they appear when the migration aid executes.

Free Format

If free formatting has been used in a FORTRAN 66/V program, you must change the source file to fixed format before the migration aid can perform any other conversions. However, since most FORTRAN 66/V programs are written in fixed format, this command does not usually need to be applied.

Applying this command to a program that is already in fixed format can produce incorrect changes. Therefore, if you plan to convert many files that are already in fixed format, you may want to make a copy of the file `FTNCMDS` and delete the format conversion commands from the copy. Then use the modified command file on the program files that are in fixed format. This will prevent you from applying the format conversion command incorrectly.

Directives That Are Changed

This command first encloses include files in single quotes. It then makes the substitutions listed in Table 3-1.

Some FORTRAN 66/V compiler directives have an equivalent directive available in the HP FORTRAN 77/V compiler. Other directives do not have exact equivalents but may have close approximations.

The following FORTRAN 66/V directives are changed to the HP FORTRAN 77/V directives shown:

Table 3-1. Conversions of Directives

FORTRAN 66/V	HP FORTRAN 77/V
$CHECK=*n*	$CHECK_FORMAL_PARM *n*
$SEGMENT=*sname*	$SEGMENT *'sname'*
$LIST	$LIST ON
$NOLIST	$LIST OFF
$CODE	$LIST_CODE ON
$NOCODE	$LIST_CODE OFF
$MAP	$TABLES ON
$NOMAP	$TABLES OFF
$BOUNDS	$RANGE ON
$WARN	$WARNINGS ON
$NOWARN	$WARNINGS OFF
$LOCATION	$CODE_OFFSETS ON
$NOLOCATION	$CODE_OFFSETS OFF
$INIT	$INIT ON

Directives That Have No Equivalent

The following FORTRAN 66/V directives have no equivalent in HP FORTRAN 77/V. They are deleted by the migration aid.

```
$EDIT
$TRACE
$CONTROL STAT
$CONTROL NOSTAT
$CONTROL SOURCE
$CONTROL NOSOURCE
$CONTROL LABEL
$CONTROL NOLABEL
$CONTROL FIXED
$CONTROL FREE
$CONTROL FILE=n-m
$CONTROL FILE=n
$CONTROL ERRORS=n
$CONTROL CROSSREF
$CONTROL CROSSREF ALL
```

$CONTROL is changed to $OPTION. ($CONTROL and $OPTION are interchangeable, but the change prevents the migration aid from reconverting the line.) If all options on a line are deleted and only $CONTROL remains, the line is deleted.

IF and SET Directives

The IF and SET directives are changed as follows:

Table 3-2. Conversions of IF and SET Directives

FORTRAN 66/V	HP FORTRAN 77/V
$SET X*n* ON	$SET (X*n*= .TRUE.)
$SET X*n* OFF	$SET (X*n*= .FALSE.)
$IF X*n*=ON	$IF (X*n*)
$IF X*n*=OFF	$IF (.NOT.X*n*)
$IF	$ENDIF

Octal Constants

Octal constants have a different notation in HP FORTRAN 77/V. Selecting Y (yes) for this set of commands changes constants of the form %*nn*Δ (where Δ represents a blank), %*nn*J, or %*nn*L to *nn*B. Other types, such as %*nn*R, are not affected.

Character Constants

Character constants also have a different notation in HP FORTRAN 77/V. Selecting Y (yes) for this set of commands changes constants of the form %*nn*C to CHAR(*nn*B) unless they are found in a FORMAT statement line, in which case it is assumed that they represent carriage control constants.

Logical Constants

Logical constants of the form %"*chars*"L are not allowed in HP FORTRAN 77/V. The migration aid converts logical constants of this form to Hollerith constants. For example, the following octal constant:

```
%"ABD"L
```

is converted to

```
3HABC
```

The migration aid converts only logical constants of the form %"*chars*"L in which *chars* is a character string of up to four characters. If you need to convert longer logical constants, you should modify this section of the migration aid.

Condition Code, Parameters Passed by Value, and Alternate Returns

When this command is applied, the following changes occur:

- Condition code constructs of the form `.CC.` are changed to `CCODE()`.
- The backslashes (\) that precede parameters passed by value are deleted.

Note

In HP FORTRAN 77/V, passing parameters by value requires that an ALIAS directive be inserted in the source code. This insertion must be done manually.

- The dollar signs ($) in alternate returns are changed to asterisks (*). For example,

```
CALL SUBRTN(A,$1,$2)
```

is changed to

```
SUBRUTN(A,*1,*2).
```

ACCEPT and DISPLAY Statements

The migration aid converts the `ACCEPT` statement to `READ` and the `DISPLAY` statement to `PRINT`. The syntax of these statements is changed as well as their names. For information on `READ` and `PRINT`, see the *HP FORTRAN 77/iX Reference* manual.

INTEGER and LOGICAL Type Declarators

`INTEGER*2` is the default size of integers in FORTRAN 66/V, while `INTEGER*4` is the default in HP FORTRAN 77/V. The migration aid converts `INTEGER` to `INTEGER*2` and `LOGICAL` to `LOGICAL*2` if no other length is specified. Declarators already having length specifiers are not converted.

PARAMETER and CHARACTER Type Declarators

When this command is applied, the migration aid converts the syntax of `PARAMETER` statements by enclosing the parameter constants in parentheses.

Note

HP FORTRAN 77/V requires that parameter constants be explicitly typed (as INTEGER, CHARACTER, etc.). This typing must be done manually.

The migration aid also converts the syntax of `CHARACTER` type declarators. For example,

```
CHARACTER BUF*10(20)
```

is changed to

```
CHARACTER BUF(20)*10.
```

Parameterless System Intrinsics

The system intrinsics listed below are changed to parameterless system intrinsics.

CALENDAR	FATHER	GETPRIVMODE	TERMINATE
CAUSEBREAK	FREELOCRIN	GETUSERMODE	TIMER
CLOCK	GETJCW	PROCTIME	
DEBUG	GETORIGIN	RESETDUMP	

For example,

```
date=CALENDAR
```

is changed to

```
date=CALENDAR()
```

If your program contains variables with names identical to these system intrinsics, answer `A` so that you are prompted before each change is made.

New Function Names

The following are the FORTRAN 66/V intrinsic functions whose names or parameter types are changed by the migration aid. For example, in FORTRAN 66/V the function `IABS` has an `INTEGER*2` parameter, whereas in HP FORTRAN 77/V its parameter type is `INTEGER*4`. Therefore, `IABS` is changed to `HABS`, which has an `INTEGER*2` parameter.

Table 3-3. Conversions of Function Names or Types.

FORTRAN 66/V	HP FORTRAN 77/V
IABS	HABS
JABS	IABS
JINT	INT
JDINT	IDINT
FLOATJ	FLOAT
INUM	ICHAR
JNUM	ICHAR
JMOD	MOD
MOD	HMOD
AJMAX0	AMAX0
JMAX0	MAX0
JMAX1	MAX1
AJMIN0	AMIN0
JMIN0	MIN0
JMIN1	MIN1
JFIX	IFIX
ISIGN	HSIGN
JSIGN	ISIGN
IDIM	HDIM
JDIM	IDIM

Conversion of Constructs of the Form VAR[i:j]

In FORTRAN 66/V, substring designators and partial word designators have the same form: `VAR[i:j]`. Only a compiler can determine whether a construct is a substring designator or a partial word designator. Therefore, the migration aid splits the conversion of this construct into two parts. The first part asks whether to treat such constructs as substring designators; the second asks whether to treat them as partial word designators.

Note

Be sure to coordinate your responses to the substring designator and partial word designator commands. If you respond `Y` to one, you should not respond `Y` to the other. If you are not sure whether a response would be appropriate for all occurrences, answer `A` (ask) so that you are prompted as each occurrence is found.

Substring Designators

This command converts constructs of the form `VAR[i:j]` to substrings. Note that whereas in FORTRAN 66/V `j` represents the number of characters in the substring, in HP FORTRAN 77/V `j` represents the position of the last character.

Applying this command can sometimes result in awkward algebraic expressions. The section "Eliminating Awkward Algebraic Expressions" discusses a partial solution to this problem.

Partial Word Designators

This command converts `VAR[i:j]` to a call to `MVBITS` or `IBITS` depending on the left side of the assignment statement.

Note

The partial word designators in FORTRAN 66/V start counting the bits from the left, whereas the bit manipulation routines in HP FORTRAN 77/V start counting the bits from the right. For example, `I=IVAR[1:2]` indicates the second and third bits from the left in FORTRAN 66/V, but in the functions `MVBITS` and `IBITS` it indicates the second and third bits from the right. This difference may result in awkward algebraic expressions that, although correct, may be difficult to read. The next section discusses a partial solution to this problem.

Eliminating Awkward Algebraic Expressions

Applying the partial word designator or substring designator command can result in awkward algebraic expressions such as

```
...,16-0,...
```

or

```
...,-1+1,...
```

These expressions are inefficient and visually awkward. This command improves the form of these expressions.

Replacing the STR Function

If you select the `STR` conversion command, the migration aid converts expressions of the form

variable1`=STR(`*variable2*`,`*format*`)`

to internal WRITE statements of the form

`WRITE(`*variable1*`,`*format*`)` *variable2*

Deleting the 'END=' Specifier in WRITE Statements

`'END='` specifiers are not allowed in HP FORTRAN 77 WRITE statements. The migration aid removes `'END='` specifiers from `WRITE` statements if you select this command.

4

What the Migration Aid Does Not Do

The migration aid cannot perform all transformations necessary to properly compile a FORTRAN 66/V program with the HP FORTRAN 77/V compiler; some changes must be performed manually. This chapter discusses the manual changes.

Some features of HP FORTRAN 77/V are incompatible with similar features in FORTRAN 66/V. This chapter also discusses these incompatibilities.

After processing a source file with the migration aid and then checking it for the constructs listed here, you can compile the program with the HP FORTRAN 77/V compiler. Any remaining problems should then be flagged by the compiler as errors or warnings.

Data Type Word Length

FORTRAN 66/V defaults to two bytes for the `INTEGER` and `LOGICAL` data types, whereas HP FORTRAN 77/V defaults to four bytes for these types. To account for these differences, the migration aid converts types defined as `INTEGER` to `INTEGER*2` and types defined as `LOGICAL` to `LOGICAL*2`. However, you may want to explicitly define some data type lengths. In deciding which length is required, take the following into account:

- Parameter-passing problems that might occur when calling external procedures (including system intrinsics) that expect data of one length or the other.
- The effects of equivalencing.
- The range of values being used.
- The amount of stack space being used.
- The FORTRAN intrinsic functions being used.

Word Length of Passed Integer and Logical Parameters

When calling subprograms that pass integer and logical parameters by *reference*, make sure that the actual and formal parameters have the same word length. (They should both be single integer or double integer.) This is especially important when calling subprograms in other languages. If parameters passed by reference are not the same, the Segmenter issues an error message. When parameters passed by `value` do not have the same word length, the HP FORTRAN 77/V compiler internally converts single integers to double.

Logical Variables

The implementation of logicals is very different in FORTRAN 66/V and HP FORTRAN 77/V. In FORTRAN 66/V, the binary representation of a `LOGICAL*2` variable that evaluates to true is

```
0000000000000001    (the least significant bit)
```

In HP FORTRAN 77/V the variable is represented as

```
0000000100000000    (the least significant bit in the high-order byte)
```

These are some other differences:

- When testing for the value true, HP FORTRAN 77/V checks only the low-order bit of the high-order byte.
- When making assignments to logical variables, HP FORTRAN 77/V manipulates only the high-order byte.
- HP FORTRAN 77/V allows bit manipulation only on integer variables, not on logicals.
- HP FORTRAN 77/V does not support assignments of octal constants to logical variables.

Functions Not Found in HP FORTRAN 77/V

The following FORTRAN 66/V functions don't exist in HP FORTRAN 77/V. If your program uses one of these functions, replace or supply it as described below, or find another way to perform the task.

```
BOOL        CSINH        CCOSH        CTANH
```

Replacing the BOOL Function

Replace the `BOOL` function in one of two ways, depending on how it is used. If FORTRAN 66/V program uses `BOOL` as a masking function, make the changes illustrated in the following example:

```
FORTRAN 66/V        i = (BOOL(i) .AND. 7L)
HP FORTRAN 77/V     i = i .AND. 7
```

If the FORTRAN 66/V program uses `BOOL` to convert to a logical (expecting logicals to be in the FORTRAN 66/V format), make the changes illustrated in the following example:

FORTRAN 66/V	`IF (BOOL(i)) GOTO 10`
HP FORTRAN 77/V	`IF (BTEST(i,0)) GOTO 10`

Note that `BTEST(i,0)` tests the low-order bit of `i`, according to the FORTRAN 66/V format for logicals. If the value of `i` has been adjusted to the HP FORTRAN 77/V format for logicals, the bit defined by HP FORTRAN 77/V must be tested.

Supplying the CSINH Function

Supply the CSINH function of FORTRAN 66/V by adding the following lines to your source code:

```
COMPLEX FUNCTION CSINH(c)
COMPLEX c
x = real(c)
y = aimag(c)
CSINH = cmplx(sinh(x) * cos(y), sin(y) * cosh(x))
RETURN
END
```

Supplying the CCOSH Function

Supply the `CCOSH` function of FORTRAN 66/V by adding the following lines to your source code:

```
COMPLEX FUNCTION CCOSH(c)
COMPLEX  c
x = real(c)
y = aimag(c)
CCOSH = cmplx(cosh(x) * cos(y), sin(y) * sinh(x))
RETURN
END
```

Supplying the CTANH Function

Supply the CTANH function of FORTRAN 66/V by adding the following lines to your source code:

```
COMPLEX FUNCTION CTANH(c)
COMPLEX  c
x = real(c)
y = aimag(c)
denom = cosh(2.0 * x) + cos(2.0 * y)
CTANH = cmplx(sinh(2.0 * x) / denom, sin(2.0 * y) / denom)
RETURN
END
```

Evaluation of Mixed Mode Expressions

Mixed mode expressions are evaluated differently in the two compilers. In operations of the same precedence, FORTRAN 66/V evaluates the same types within an expression first, while HP FORTRAN 77/V evaluates strictly from left to right.

The following example program produces different results in the two compilers:

```
INTEGER*4 j
j = 2000000000
WRITE(6,*) 1.0+j-j
END
```

The result returned in FORTRAN 66/V is 1.0. The result returned in HP FORTRAN 77/V is 0.0.

In FORTRAN 66/V, the expression `j-j` is evaluated first. In HP FORTRAN 77/V, the expression `1.0+j` is evaluated first. Since the constant `1.0` is a single precision real, only six digits are available to hold the partial result, and the last four digits of 2000000000 are not stored.

You can use parentheses to force the order of evaluation you want. For instance, if you want the HP FORTRAN 77/V program to yield the answer 1.0, make this change:

```
WRITE(6,*) 1.0+(j-j)
```

Alternately, you could explicitly type the constant as double precision, as shown:

```
WRITE(6,*) 1.0D0+j-j
```

The S Edit Descriptor

FORTRAN 66/V's `S` edit descriptor for character data can be changed to HP FORTRAN 77/V's `A` or `R` descriptor. In no case should `S` be left as a character descriptor, because `S` controls the plus sign in ANSI standard FORTRAN 77.

Named Constants in PARAMETER Statements

In the `PARAMETER` statement of FORTRAN 66/V, the type of a named constant is determined solely by the constant itself and not by the initial letter of its name. In HP FORTRAN 77/V, the type is determined by the initial letter of the name. For this reason, a named constant should be explicitly typed before using the `PARAMETER` statement.

Passing Character Variables

When HP FORTRAN 77/V passes character variables to subroutines, two parameters (or two words) are actually passed. The first parameter is a pointer to the beginning of the character data; the second parameter is the length of the character data. This second parameter must be taken into account when passing character data from an HP FORTRAN 77/V subroutine to a FORTRAN 66/V subroutine. To accommodate the parameter, do one of the following:

- Include the `$FTN3000_66 CHARS` directive in the HP FORTRAN 77/V subroutine.
- Include the `$ALIAS` directive in the HP FORTRAN 77/V subroutine.

To pass a character variable from a FORTRAN 66/V subroutine to an HP FORTRAN 77/V subroutine, do one of the following:

- Include the `$FTN3000_66 CHARS` directive in the HP FORTRAN 77/V subroutine and then recompile.
- Change the parameter list in the FORTRAN 66/V subroutine to include a dummy length parameter (which is passed by value).

Parameter Limit

On MPE V the number of parameters that may be passed from one program unit to another is limited to 63. (This is true for all languages on MPE V.) However, HP FORTRAN 77/V passes two parameters for every argument that is a character variable: one parameter is the length and the other is a pointer to the beginning of the variable. Therefore, when converting FORTRAN 66/V programs having long parameter lists in which some parameters are character variables, the converted program may exceed the 63 parameter limit. To reduce the parameter number, move some parameters into common.

DO Loops

HP FORTRAN 77/V does not allow jumping into the middle of a `DO` loop. `DO` loops with this feature in FORTRAN 66/V require a logic change in HP FORTRAN 77/V.

The migration aid automatically inserts the `CONTROL ONETRIP` compiler directive in converted programs. However, new programs must specify this directive if they require all loops to be executed at least once.

Composite Numbers

HP FORTRAN 77/V does not allow composite numbers. They should be changed to the most convenient alternative format (decimal, octal, hexadecimal, etc.). For example,

```
DATA KEYDEF/ %[4/1,12/16], %[8/%200] /
```

could be changed to

```
DATA KEYDEF/ 10200B, 128 /
```

In this example, the composite number `[4/1,12/16]` has been changed to the octal `10200B`, and `[8/%200]` has been changed to the decimal `128`.

Alternate Returns

Alternate returns are implemented differently in the two compilers. Because of this difference, a FORTRAN 66/V program cannot call an HP FORTRAN 77/V subroutine using the alternate return mechanism. Conversely, an HP FORTRAN 77 program cannot call a FORTRAN 66/V subroutine using an alternate return.

Free-Format Internal Reads

HP FORTRAN 77/V does not currently support free-format internal reads. An example of a free-format internal read is

```
READ (char,*) I,A
```

where *char* is a character variable and not a unit number.

Format Statements

Format statements should be changed as follows:

Table 4-1. Format Conversions

Change These	To These
`2/`	`//`
`3/`	`///`
`2"-"`	`2("-")`

Recognition of End of Data

When reading from a terminal, FORTRAN 66/V recognizes end of data when it encounters either a colon (`:`) or (`:EOD`) in input data. HP FORTRAN 77/V recognizes end of data only when it encounters `:EOD`, it does not recognize a colon as end of data. This feature of HP FORTRAN 77/V prevents logging off if an input line of `:EOF` is entered, and allows input of lines containing a colon in the first position. A colon can be read as an ordinary character in the input data.

HP FORTRAN 77/V can allow input of a colon because it opens unit FTN05 (unit 5) differently from FORTRAN 66/V. Both FORTRAN 66/V and HP FORTRAN 77/V open unit 5 with the FOPEN system intrinsic. However, in FORTRAN 66/V the *foptions* parameter designates `$STDIN`, whereas in HP FORTRAN 77/V the *foptions* parameter designates `$STDINX`. `$STDIN` and `$STDINX` both represent the standard input device, but `STDINX` does not recognize a colon as end of data.

If your program requires that a colon indicate end of data when reading from a terminal (as FORTRAN 66/V does), use the following file equation before running your program:

```
:FILE FTN05=$STDIN
```

This equation designates `$STDIN` as the standard input device, instead of `$STDINX`.

List-Directed READ Statements

HP FORTRAN 77 handles list-directed input differently from FORTRAN 66/V. FORTRAN 66/V allows you to input fewer values than the number of variables in a READ statement. HP FORTRAN 77/V requires that you either supply a value for each variable in the READ statement or append a slash (`/`) after the last value.

For example, for the source lines

```
REAL A,B
READ (5,*) A,B
```

in FORTRAN 66/V you can input a value for `A`, followed by a carriage return. In HP FORTRAN 77/V, you must enter values for `A` and `B` or else input a value for `A` and follow it with a slash (`/`).

If your program uses list-directed `READ`s from a data file and any lines in the data file contain fewer input values than the `READ` expects, append a slash to the end of each line (this can be done with a global command through your editor).

5

Using the Migration Aid

This chapter explains how to use the migration aid and shows an example of a migration. Please read the entire chapter before using the migration aid.

Checking Your Catalog

The `CONVERT.UDC` must exist in your catalog before you can use the migration aid. To list the contents of your catalog, enter the command `SHOWCATALOG`. If `CONVERT` is not listed, execute this command:

```
:SETCATALOG FTNUDC.PUB.SYS[,otherudc]
```

where

otherudc is the list of any other `UDC` files that you are using.

Running the Migration Aid

To run the program `FTNCVT.PUB.SYS` on a FORTRAN 66/V source file, use the following command:

```
:CONVERT oldsource[,[newsource][,commandfile]]
```

where

oldsource is the name of the FORTRAN 66/V source file to be converted to an HP FORTRAN 77/V source file.

newsource is the name of the HP FORTRAN 77/V file to be created. If *newsource* is not specified, the file defaults to `$STDLIST`; the converted file is displayed to the terminal and no source file is saved.

commandfile is the name of the command file to be used. This is either the command file supplied with the utility or a command file of your own making. If *commandfile* is not specified, the conversion operations supplied in the file `FTNCMDS.PUB.SYS` are used.

The supplied command file can be modified, or new command files created to your specifications. Refer to Chapter 5 for more information.

Example Conversion

Here is an example that shows the steps in converting a FORTRAN 66/V source program to an HP FORTRAN 77/V source program.

This is the program as originally written in FORTRAN 66/V:

```
$control uslinit
      program test
      system intrinsic dateline, calendar
      parameter prompt = "today is ...."
      character cctrl,datebuf*27
      character*12 string
      integer today,age,daym(12)
      integer dbirth, mbirth, ybirth
      integer dtoday, mtoday, ytoday
C     calendar  jan,feb,mar,apr,may,jun,jul,aug,sep,oct,nov,dec
      data daym/ 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31/
      call dateline(datebuf)
      display prompt,datebuf[1:17]
      cctrl=%320C
      write(6,1100) cctrl
      read(5,1200) string
      write(6,1300)
      read(5,1400) ybirth,mbirth,dbirth
      today=int(calender)
      ytoday=today[0:7]
      dtoday=today[7:9]
      mtoday=0
      do 50 i=1,12
         mtoday=mtoday + 1
         if (dtoday .le. daym(mtoday)) goto 60
         dtoday=dtoday - daym(mtoday)
50    continue
60    age=ytoday - ybirth
      if (mtoday - mbirth) 100,80,200
80    if (dtoday - ybirth) 100,200,200
100   age=ytoday - ybirth - 1
200   write(6,1500) string,age
      stop
1100  format(1a1,'. May I have your name? ')
1200  format(a12)
1300  format(%320C,"and your birthday (yymmdd):")
1400  format(3I2)
1500  format(" ",a12,", I believe your age is ",I2)
      end
```

Running the Migration Aid on the Example Program

The following command runs the migration aid on the FORTRAN 66/V source file INPUT66 and produces the HP FORTRAN 77/V source file OUTPUT77:

```
:CONVERT INPUT66,OUTPUT77,FTNCMDS.PUB.SYS
```

While the migration aid runs, it sends the following information to the terminal (user responses are highlighted):

```
FORTRAN MIGRATION AID (C) 1987/A.00.02

++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
+
+ FORTRAN/V TO FORTRAN 77/V MIGRATION AID
+ ======================================
+ This migration aid reads commands from the COMMAND file and asks
+ you whether to apply them or not in the conversion process. For
+ each command, the migration aid asks:
+
+ Apply the corresponding commands? (expected Y,N,A)
+
+ Answer Y to apply them automatically.
+        N to not apply them.  (Carriage return assumes this option).
+        A to be asked before the changes are done.
+        E to exit the program (no conversion will be done).
+
+
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

 Free format allows you to start your source code anywhere on the
 line; this is no longer allowed in FORTRAN 77.  The next changes
 convert free format to fixed format;

 WARNING! if the text is already in FIXED format, answer "N" to the
 question otherwise wrong modifications will result.

 If you plan to convert many files that already are in FIXED format,
 you might find it convenient to delete this set of commands from the
 file FTNCMDS (or better, you create your own copy of the file
 FTNCMDS without them) in order to avoid being asked to apply these
 commands and answering "YES" accidentally.


apply these commands? expected (Y,N,A) N


 The next changes will affect the $INCLUDE and $CONTROL compiler
 directives.  Changes are

  enclose include file with quotes.
  - from CHECK=n  to  CHECK_FORMAL_PARM n
```

```
  - from SEGMENT=sname to SEGMENT "sname"
  - from LIST/NOLIST to LIST on/off
  - from CODE/NOCODE to LIST_CODE on/off
  - from MAP/NOMAP to TABLES on/off
  - from BOUNDS to RANGE on
  - from WARN/NOWARN to WARNINGS on/off
  - from LOCATION/NOLOCATION to CODE_OFFSETS on/off
  - from INIT to INIT ON

apply these commands? expected (Y,N,A) Y

 The next changes will delete any compiler directive in FORTRAN/V that
 has no equivalent in FORTRAN 77.
 They are $EDIT
          $TRACE
          $CONTROL STAT/NOSTAT
          $CONTROL SOURCE/NOSOURCE
          $CONTROL LABEL/NOLABEL
          $CONTROL FIXED
          $CONTROL FREE
          $CONTROL FILE=n-m
          $CONTROL FILE=n
          $CONTROL ERRORS=n
          $CONTROL CROSSREF
          $CONTROL CROSSREF ALL
 If after deletion of these options the control line is left empty
 the line will be removed.

apply these commands? expected (Y,N,A) Y

 The next changes will affect the directive $SET. They will change
 strings of the form $SET Xn ON / OFF to $SET (Xn=.TRUE. / .FALSE.)

 They will also change conditional compile expressions of the form
 $IF Xn=ON/OFF to $IF (Xn/.not.Xn)

apply these commands? expected (Y,N,A) Y

 The next changes will affect octal constants of the form %nn
 followed by a J or L or alone.  They will be converted to nnB.  Note
 that numbers of the form %nnC, %nnR, %nnD won't be changed.

apply these commands? expected (Y,N,A) Y

 The next changes will convert numerical ascii expressions to its
 Hollerith equivalent, for example,
   %"c"L ----> 1Hc
   %"cc"L ---> 2Hcc
   %"ccc"L---> 3Hccc
   %"cccc"L--> 4Hcccc
```

```
 apply these commands? expected (Y,N,A) Y

 The next changes will affect character constants of the form %nnC.
 They will be converted to CHAR(nn) except if they are found in the
 same line as a FORMAT statement.  Note that numbers of the form
 %nn,%nnJ,%nnL,%nnR,%nnD won't be changed.

apply these commands? expected (Y,N,A) Y

 The next changes will affect the condition code intrinsic,
 alternative return locations, and parameters passed by value in the
 following way,
 -Occurrences of .CC. will be converted to CCODE().
 -Alternative return locations '$nn' will be converted to '*nn'
  locations.
 -The backslash (\) of parameters passed by value will be deleted.
 CAUTION: in FORTRAN 77/V, the actual mode of passing variables
 should be specified through the ALIAS statement.

apply these commands? expected (Y,N,A) Y

 The next changes will convert ACCEPT and DISPLAY statements
 to READ * and PRINT * respectively.

apply these commands? expected (Y,N,A) Y

 The next commands will affect default INTEGER and LOGICAL variables.
 INTEGER and LOGICAL variables will be converted to INTEGER*2 and
 LOGICAL*2, respectively.
 Note that this command is intended to preserve the 16-bit length
 variables since in MPE/iX the default is 32-bit (or INTEGER*4). If
 you use the directive $SHORT, you might not want to apply this
 command.  Note as well that in MPE/iX, the 32-bit variables are
 handled more efficiently than the 16-bit ones.

apply these commands? expected (Y,N,A) Y

 The next commands will change CHARACTER declarations to the new
 syntax.  They will also enclose the object of PARAMETER statements
 between parentheses Note that the changed lines may still need
 manual changes.

apply these commands? expected (Y,N,A) Y

 The next commands will put parentheses after the parameterless
 intrinsics.  For example,

 100    C=CLOCK    will be changed to
 100    C=CLOCL()
```

```
 The intrinsics affected are
 -CALENDAR           CAUSEBREAK      CLOCK
 -DEBUG              FATHER          FREELOCRIN
 -GETJCW             GETORIGIN       GETPRIVMODE
 -GETUSERMODE        PROCTIME        RESETDUMP
 -TERMINATE          TIMER

apply these commands? expected (Y,N,A) Y
 The next commands will change all the following intrinsic functions
 to their corresponding name in FORTRAN 77.

  IABS....HABS     JABS....IABS      JINT....INT        JDINT...IDINT
  JMOD....MOD      MOD.....HMOD      AJMAX0..AMAX0      JMAX0...MAX0
  JMAX1...MAX1     AJMIN0..AMIN0     JMIN0...MIN0       JMIN1...MIN1
  JFIX....IFIX     ISIGN...HSIGN     JSIGN...ISIGN      IDIM....HDIM
  JDIM....IDIM     FLOATJ..FLOAT

apply these commands? expected (Y,N,A) Y

 The next two sets of commands affect substrings and partial-word
 designators which have the same syntax in FORTRAN/V but differ in
 FORTRAN 77.  Don't answer "Y" to both of them but alternatively, if
 you answer "Y" to one of them, answer "N" to the other or use Ask
 mode ("A").

 Please refer to the Migration Guide for further explanations.

 EXPRESSIONS OF THE FORM VAR[e1:e2]. Part 1.
 ===========================================
 The next commands will treat strings of the form VAR[m:n] as
 substrings, and will change them to VAR(m:m+n-1) (note that n does
 not represent the number of characters anymore but rather the last).
 If the expression does not contain n (i.e. VAR[m]), it will be
 changed to the corresponding VAR(m:) in FORTRAN 77.

apply these commands? expected (Y,N,A) A

 EXPRESSIONS OF THE FORM VAR[e1:e2]. Part 2.
 ===========================================
 The next changes will convert constructs of the form
 VARi[e1:e2]=VARj[e3:e4] to a call to the bit intrinsic MVBITS (move
 bits), and expressions of the form VARi=VARj[e3:e4] to a call to
 IBITS (extract bits).

apply these commands? expected (Y,N,A) A

 The next commands will change some cumbersome math generated by the
 preceding commands, such as 15-15, 16-0, and 1-1.

apply these commands? expected (Y,N,A) Y
```

```
 The next change will replace the STR function with an internal WRITE

  VAR1 = STR(VAR2, num) --> WRITE (VAR1, Inum) VAR2

apply these commands? expected (Y,N,A) Y

 Deletion of END=label in WRITE statements

apply these commands? expected (Y,N,A) Y

 NOW THE CONVERSION WILL START.

old line:
$control uslinit
new line:
$OPTION uslinit                                                  MIGF77
C$control uslinit                                                MIGF66
$OPTION uslinit                                                  MIGF77
      program test
      system intrinsic dateline,calendar
old line:
      parameter prompt = "today is...."
new line:
      PARAMETER ( prompt = "today is....")                       MIGF77
C      parameter prompt = "today is...."                         MIGF66
      PARAMETER ( prompt = "today is....")                       MIGF77
      character cctrl,datebuf*27
      character*12 string
old line:
      integer today,age,daym(12)
new line:
      INTEGER*2 today,age,daym(12)                               MIGF77
C      integer today,age,daym(12)                                MIGF66
      INTEGER*2 today,age,daym(12)                               MIGF77
old line:
      integer dbirth, mbirth, ybirth
new line:
      INTEGER*2 dbirth, mbirth, ybirth                           MIGF77
C      integer dbirth, mbirth, ybirth                            MIGF66
      INTEGER*2 dbirth, mbirth, ybirth                           MIGF77
old line:
      integer dtoday, mtoday, ytoday
new line:
      INTEGER*2 dtoday, mtoday, ytoday                           MIGF77
C      integer dtoday, mtoday, ytoday                            MIGF66
      INTEGER*2 dtoday, mtoday, ytoday                           MIGF77
C     calendar  jan,feb,mar,apr,may,jun,jul,aug,sep,oct,nov,dec
      data daym/ 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31/
      call dateline(datebuf)
```

```
old line:
      PRINT *, prompt,datebuf[1:17]
ASK option was requested
      PRINT *, prompt,datebuf(1:1+17-1)
apply it? expected (Y,N) Y
old line:
      display prompt,datebuf[1:17]
new line:
      PRINT *, prompt,datebuf(1:17)                                  MIGF77
C      display prompt,datebuf[1:17]                                  MIGF66
      PRINT *, prompt,datebuf(1:17)                                  MIGF77
old line:
      cctrl=%320C
new line:
      cctrl=CHAR(320B)                                               MIGF77
C      cctrl=%320C                                                   MIGF66
      cctrl=CHAR(320B)                                               MIGF77
      write(6,1100) cctrl
      read(5,1200) string
      write(6,1300)
      read(5,1400) ybirth,mbirth,dbirth
old line:
      today=int(calendar)
new line:
      today=int(CALENDAR())                                          MIGF77
C      today=int(calendar)                                           MIGF66
      today=int(CALENDAR())                                          MIGF77
old line:
      ytoday=today[0:7]
ASK option was requested
      ytoday=today(0:0+7-1)
apply it? expected (Y,N) N
old line:
      ytoday=today[0:7]
ASK option was requested
      ytoday=IBITS(today,-0+15-7+1,7)
apply it? expected (Y,N) Y
old line:
      ytoday=today[0:7]
new line:
      ytoday=IBITS(today,+15-7+1,7)                                  MIGF77
C      ytoday=today[0:7]                                             MIGF66
      ytoday=IBITS(today,+15-7+1,7)                                  MIGF77
old line:
      dtoday=today[7:9]
ASK option was requested
      dtoday=today(7:7+9-1)
apply it? expected (Y,N) N
old line:
      dtoday=today[7:9]
```

```
ASK option was requested
      dtoday=IBITS(today,-7+15-9+1,9)
apply it? expected (Y,N) Y
old line:
      dtoday=today[7:9]
new line:
      dtoday=IBITS(today,+8-9+1,9)                                    MIGF77
C      dtoday=today[7:9]                                              MIGF66
      dtoday=IBITS(today,+8-9+1,9)                                    MIGF77
      mtoday=0
      do 50 i=1,12
         mtoday=mtoday+1
         if ( dtoday .le. daym(mtoday)) go to 60
         dtoday=dtoday-daym(mtoday)
50    continue
60    age=ytoday-ybirth
      if ( mtoday-mbirth) 100,80,200
80    if ( dtoday-dbirth) 100,200,200
100   age=ytoday-ybirth-1
200   write(6,1500) string,age
      stop
1100  format(1a1,'. May I have your name? ')
1200  format(a12)
1300  format(%320C,"and your birthday (yymmdd):")
1400  format(3I2)
1500  format(" ",a12,", I believe your age is ",I2)
      end

10 lines have been changed.(21)
38 lines of code found
1 comment lines found
```

The Converted Program

Here is the converted program, OUTPUT77:

```
C$control uslinit
$OPTION uslinit
      program test
      system intrinsic dateline,calendar
C      parameter prompt = "today is...."
      PARAMETER ( prompt = "today is....")
      character cctrl,datebuf*27
      character*12 string
C      integer today,age,daym(12)
      INTEGER*2 today,age,daym(12)
C      integer dbirth, mbirth, ybirth
      INTEGER*2 dbirth, mbirth, ybirth
C      integer dtoday, mtoday, ytoday
      INTEGER*2 dtoday, mtoday, ytoday
C     calendar  jan,feb,mar,apr,may,jun,jul,aug,sep,oct,nov,dec
      data daym/ 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31/
      call dateline(datebuf)
```

```
C      display prompt,datebuf[1:17]
      PRINT *, prompt,datebuf(1:17)
C      cctrl=%320C
      cctrl=CHAR(320B)
      write(6,1100) cctrl
      read(5,1200) string
      write(6,1300)
      read(5,1400) ybirth,mbirth,dbirth
C      today=int(calendar)
      today=int(CALENDAR())
C      ytoday=today[0:7]
      ytoday=IBITS(today,+15-7+1,7)
C      dtoday=today[7:9]
      dtoday=IBITS(today,+8-9+1,9)
      mtoday=0
      do 50 i=1,12
         mtoday=mtoday+1
         if ( dtoday .le. daym(mtoday)) go to 60
         dtoday=dtoday-daym(mtoday)
50    continue
60    age=ytoday-ybirth
      if ( mtoday-mbirth) 100,80,200
80    if ( dtoday-dbirth) 100,200,200
100   age=ytoday-ybirth-1
200   write(6,1500) string,age
      stop
1100  format(1a1,'. May I have your name? ')
1200  format(a12)
1300  format(%320C,"and your birthday (yymmdd):")
1400  format(3I2)
1500  format(" ",a12,", I believe your age is ",I2)
      end
```

Compilation of the Converted Program

When OUTPUT77 is compiled by the HP FORTRAN 77/V compiler, it produces the following listing:

```
 PAGE    1  HEWLETT-PACKARD   HP32116A.00.08
HP FORTRAN 77 (C) HEWLETT-PACKARD CO. 1987 MON, MAY 18, 1987, 12:55 PM


    0     1         C$control uslinit
    0     2         $OPTION uslinit
    1     3               program test
    2     4               system intrinsic dateline,calendar
    2     5         C      parameter prompt = "today is...."
                           ^
 **** WARNING #  1  WARNING: THIS SYSTEM-SPECIFIC FEATURE IS NOT PART
OF HP STANDARD FORTRAN 77 (830)
    3     6               PARAMETER ( prompt = "today is....")
                                            ^
 **** ERROR # 1   INCONSISTENT PARAMETER TYPE (153)
    4     7               character cctrl,datebuf*27
    5     8               character*10 string(10)
    5     9         C      integer today,age,daym(12)
    6    10               INTEGER*2 today,age,daym(12)
    6    11         C      integer dbirth, mbirth, ybirth
    7    12               INTEGER*2 dbirth, mbirth, ybirth
    7    13         C      integer dtoday, mtoday, ytoday
    8    14               INTEGER*2 dtoday, mtoday, ytoday
    8    15         C  calendar jan,feb,mar,apr,may,jun,jul,aug,sep,oct,nov,dec
    9    16               data daym/31,28,31,30,31,30,31,31,30,31,30,31/
   10    17               call dateline(datebuf)
   10    18         C      display prompt,datebuf[1:17]
                             ^
**** WARNING #  2   WARNING: THIS SYSTEM-SPECIFIC FEATURE IS NOT
PART OF HP STANDARD FORTRAN 77 (830)
   11    19               PRINT *, prompt,datebuf(1:17)
   11    20         C      cctrl=%320C
   12    21               cctrl=CHAR(320B)
   13    22               write(6,1100) cctrl
   14    23               read(5,1200) string
   15    24               write(6,1300)
   16    25               read(5,1400) ybirth,mbirth,dbirth
   16    26         C      today=int(calendar)
   17    27               today=int(CALENDAR())
   17    28         C      ytoday=today[0:7]
                                 ^
**** WARNING # 3 WARNING: THIS SYSTEM-SPECIFIC FEATURE IS NOT PART OF
HP STANDARD FORTRAN 77 (830)
   18    29               ytoday=IBITS(today,+15-7+1,7)
   18    30         C      dtoday=today[7:9]
   19    31               dtoday=IBITS(today,+8-9+1,9)
   20    32               mtoday=0
   21    33               do 50 i=1,12
   22    34      1           mtoday=mtoday+1
   23    35      1           if ( dtoday .le. daym(mtoday)) go to 60
   24    36      1           dtoday=dtoday-daym(mtoday
```

```
  25    37      1  50    continue
  26    38         60    age=ytoday-ybirth
  27    39               if ( mtoday-mbirth) 100,80,200
  28    40         80    if ( dtoday-dbirth) 100,200,200
  29    41         100   age=ytoday-ybirth-1
  30    42         200   write(6,1500) string,age
  31    43               stop
  32    44         1100  format(1a1,'. May I have your name? ')
  33    45         1200  format(a12)

PAGE    2


  34    46         1300  format(%320C,"and your birthday (yymmdd)
  35    47         1400  format(3I2)
  36    48         1500  format(" ",a12,", I believe your age is ",I2)
  37    49               end

  NUMBER OF ERRORS =     1   NUMBER OF WARNINGS =    3
  PROCESSOR TIME   0: 0: 2   ELAPSED TIME      0: 0:11
  NUMBER OF LINES =     49
```

Recompilation of the Converted Program

An error occurred because the migration aid could not explicitly type the constant in the PARAMETER statement. We can correct the problem by editing the converted program to do the explicit typing. Next we delete the lines containing MIGF66 and add the $STANDARD_LEVEL SYSTEM compiler directive to eliminate the warnings. Here is the result of recompiling with these corrections. A sample run of the program follows.

```
 PAGE    1  HEWLETT-PACKARD   HP32116A.00.08
HP FORTRAN 77 (C) HEWLETT-PACKARD CO. 1987 MON, MAY 18, 1987, 12:58 PM


0    1      $standard_level system
0    2      $OPTION uslinit
1    3            program test
2    4            system intrinsic dateline,calendar
3    5            Character*12 prompt
4    6            PARAMETER ( prompt = "today is)
5    7            character cctrl,datebuf*27
6    8            character*12 string
7    9            INTEGER*2 today,age,daym(12)
8   10            INTEGER*2 dbirth, mbirth, ybirth
9   11            INTEGER*2 dtoday, mtoday, ytoday
9   12      C     calendar  jan,feb,mar,apr,may,jun,jul,aug,sep,oct,nov,dec
10  13            data daym/ 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31/
11  14            call dateline(datebuf)
12  15            PRINT *, prompt,datebuf(1:17)
13  16            cctrl=CHAR(320B)
14  17            write(6,1100) cctrl
```

```
15  18            read(5,1200) string
16  19            write(6,1300)
17  20            read(5,1400) ybirth,mbirth,dbirth
18  21            today=int(CALENDAR())
19  22            ytoday=IBITS(today,+15-7+1,7)
20  23            dtoday=IBITS(today,+8-9+1,9)
21  24            mtoday=0
22  25            do 50 i=1,12
23  26   1           mtoday=mtoday+1
24  27   1           if ( dtoday .le. daym(mtoday)) go to 60
25  28   1           dtoday=dtoday-daym(mtoday)
26  29   1  50    continue
27  30      60    age=ytoday-ybirth
28  31            if ( mtoday-mbirth) 100,80,200
29  32      80    if ( dtoday-dbirth) 100,200,200
30  33      100   age=ytoday-ybirth-1
31  34      200   write(6,1500) string,age
32  35            stop
33  36      1100  format(1a1,'. May I have your name? ')
34  37      1200  format(a12)
35  38      1300  format(%320C,"and your birthday (yymmdd) ")
36  39      1400  format(3I2)
37  40      1500  format(" ",a12,", I believe your age is ",I2)
38  41            end

NUMBER OF ERRORS =     0   NUMBER OF WARNINGS =    0
PROCESSOR TIME   0: 0: 3   ELAPSED TIME      0: 0:14
NUMBER OF LINES =     41

END OF PROGRAM

END OF PREPARE


today is....MON, MAY 18, 1987. May I have your name? Wendy Carlos
and your birthday (yymmdd):540806

Wendy Carlos, I believe your age is 32

END OF PROGRAM
```

6

Customizing the Command File

This section describes how the commands of the command file work. You do not need to read this section unless you plan to create new command files or add more commands to the existing file.

Since the conversion instructions reside in ASCII files, the instructions can be customized to your specifications. You can add new commands to the existing file or create new command files that suit your particular needs. The supplied command file is `FTNCMDS.PUB.SYS`. If you modify the command file, we recommend copying the file and making the modifications in the copy. You can then specify the new file as the third parameter to the `UDC CONVERT`.

If you use a text editor to create a new commands file, be sure that no sequence numbers are stored in the file. The migration aid treats all the usable length of the line as a command.

Command Syntax

The command file uses these commands:

`R`	Perform the replacement, at most once per line.
`N`	Perform the replacement, but don't do any more replacements on this line.
`G`	Perform the replacement everywhere it occurs on the line.
`D`	If the expression is found, delete the line.
`J`	If the search expression is not found, skip the next commands until another group of commands is found.
`Y`	If the search expression is not found, don't do any more replacement on this line.
`S`	If the line is found, perform the replacement and set an internal flag.
`T`	If the expression is found and the flag is already set, perform the replacement and clear the flag.
`|`	Continuation line; up to five continuation lines are allowed.
`*`	This character at the start of a line indicates a comment.

The syntax of the `R`, `N`, `G`, `S`, and `T` commands is

```
command/search expression//replace expression/
```

Example

```
R/boy//girl/
```

This replaces the first occurrence of `boy` (upper- or lowercase) in each line with `girl`.

The slashes in the example are the search string delimiters. Two slashes are used between the search and replacement strings. If one is omitted, the search is performed but no replacement done.

The syntax for the `D`, `J`, and `Y` commands is

```
command/delete expression/
```

Example

```
D/boy/
```

If the string `boy` (upper- or lowercase) is found, the line is deleted.

The set of commands used in search expressions differ from those used in replacement expressions. In particular, some of the same command characters have different meanings depending on whether they are part of the search expression field or the replace expression field. The following sections explain these differences.

Search String Commands

Search string commands are those that appear on the left side of the command.

The following characters are interpreted as commands if they precede a search string, unless they are themselves preceded by a backslash character (\).

Position Expressions

^	search string starts at beginning of line
$	search string ends at end of line
^*nnn*	search string starts at column *nnn*
$*nnn*	search string ends at (*last column* - *nnn*)
<*nnn*	search string starts in any column before *nnn*
>*nnn*	search string starts in any column after *nnn*
\	don't interpret the next character as a command

Tag Fields

`{`*expn*`}`	"Remember" the text matched by the expression *expn* for use in the replacement string.
`a&b`	Match a string delimited by evenly nested occurrences of `a` and `b` (which can be any two characters), and also "remember" the contents for use in the replacement string.

Example

```
R/(&)//[wow]/
```

This command replaces any set of characters delimited by parentheses with the string `[wow]`. For example, if a source line contained the string `(hi(people))`, the migration aid would replace it with `[wow]`.

Character Classes

`[`*set*`]`	Match any character in *set*, which is defined as ■ specific list of characters (as, for example, `[abde]` or `[ab012&"@]`), or ■ range of characters, the first separated from the last by a hyphen (as, for example, `[a-z]` or `[0-9]`), or ■ mixture of the two.
`[^`*set*`]`	Match any characters except those in *set*, which is as defined above.

Examples

`[a-gxyz]`	is the set `abcdefgxyz`
`[a-zA-Z]`	is the complete set of alphabetic characters
`[^0-9]`	is anything but a digit

Closures

`*`	Match zero or more occurrences of the preceding character or character class (but *do not* match anything else).
`*`*nnn*	Match exactly *nnn* occurrences of preceding character or class.
`+`*nnn*	Match at least *nnn* occurrences of preceding character or class.
`?`*nnn*	Match up to *nnn* occurrences of preceding character or class.
`.`	Match any single character.
`@`	Match any string until the next character is found.

Examples

The following command converts all occurrences of the string `ACCEPT` to the string `READ *,`.

```
G/{@}ACCEPT//&1READ \*,/
```

The following command converts only the first occurrence of the string `ACCEPT` to the string `READ *,`.

```
R/{@}ACCEPT//&1READ \*,/
```

Replacement String Commands

Replacement string commands are those that appear on the right side of the `R`, `N`, `G`, and `S` commands.

Tag Fields

`&`*n*	Put tag field *n* into the replacement string at the current position. If *n* is not specified, use the entire search string.
`>`*n*	Substitute the *n*th tag field matched and shift it to uppercase.
`<`*n*	Substitute the *n*th tag field matched and shift it to lowercase.

Fill Commands

`*`*x*	Change the fill character to *x* (the default is the space character).
`^`*n*	Fill with fill character up to column *n*.
`$`*n*	Move the rest of the replacement string so that it terminates at column *n*.

Examples

`R/^2[a-zA-Z]//^7&/`	Moves everything found starting at column 2 of the current line to column 7 if an alphabetic character is found in column 2. Then fills columns 2 through 6 with spaces.
`R/^boy//girl/`	Replaces `boy` only if it occurs at the beginning of a line.
`R/b.y//girl/`	Replaces `boy`, `bay`, `buy`, `bny`, etc.
`R/^6BEGIN//^9BEGIN/`	Moves `BEGIN`, if found starting in column 6, to column 9, but does not affect `BEGIN` found anywhere else in the line.

Example Command File

Here is an example of a command file, with comments explaining what it does.

```
* This file converts free format to fixed format FORTRAN.
*
* If the string "$CONTROL FREE" is found, delete that line:
D/$CONTROL FREE/
* If a line begins with a series of alphanumeric characters,
* followed by a space, delete them (or, if the line begins with just
* a space, delete the space).  This removes blank spaces in column 1
* of the FORTRAN 66/V line numbers and deletes the
* "sequence fields," which have no counterpart
* in HP FORTRAN 77/V.
R/^[a-zA-Z0-9]* ///
* If a line begins with a #, replace it with a C.
N/^\#//C/
* If a line begins with one or more digits followed by a space,
* move everything that follows the space to the right of
* column 7.  This moves all statements preceded by statement
* numbers from whatever column they are in to column 7.
R/^[0-9]+ //&^7/
* If a line begins with anything other than a digit or a dollar
* sign, move that to the right of column 7.  This moves all
* statements, except compiler options and statements with
* statement numbers, to column 7.
R/^[^0-9$]//^7&/
* The T and S commands are used in this order to "remember" that
* an & was found on the previous line and to put it on the next line.
* This produces the FORTRAN 66/V construct of an ampersand at the end
* of a line indicating a continuation line to conform to the HP FORTRAN 77/V
* construct of a character in column 6 indicating a continuation line.
T/^6 //\&/
S/\&$///
```

7

Introduction to Part II

This part of the migration guide describes migration of HP FORTRAN 77/V programs to the MPE/iX operating system.

Compatibility Mode Versus Native Mode

HP FORTRAN 77/V programs can be run on MPE/iX in *compatibility mode* or *native mode*. As discussed in Chapter 1, FORTRAN 77/V object code can be directly transferred to an MPE/iX based system with no changes to the source code; these programs can then be run on an MPE/iX system under compatibility mode. Although it is a fast way to move programs from one system to the other, migrating to compatibility mode does not take full advantage of the higher performance of the 900 Series HP 3000 computer and the MPE/iX operating system.

An alternate method is to convert the FORTRAN 77/V source code to HP FORTRAN 77/iX source code, compile it with the HP FORTRAN 77/iX compiler, and run the compiled code on the MPE/iX system in native mode. Programs run in native mode are usually much faster and more efficient than those run in compatibility mode.

Migrating to compatibility mode was explained in Chapter 1 ("Introduction"). Migrating to native mode is explained in the rest of this manual.

Factors Affecting Migration

Many HP FORTRAN 77/V programs require no more than recompilation by the HP FORTRAN 77/iX compiler to run correctly, since the version of FORTRAN 77 accepted by the HP FORTRAN 77/iX compiler is a superset of that accepted by the HP FORTRAN 77/V compiler.

HP FORTRAN 77/V programs containing certain features may need some modification to be compiled by the HP FORTRAN 77/iX compiler and to run correctly after compilation. These features are described in Chapter 8 ("Differences").

If a program has any of the features described in Chapter 8, you may need to make some changes to the program before compiling it with the HP FORTRAN 77/iX compiler. These changes are described in Chapter 9 ("Source Program Conversion").

Changes may also be required to HP FORTRAN 77/V data files before they can be used by an HP FORTRAN 77/iX program. These changes are described in Chapter 10 ("Data File Conversion").

8

Differences

An HP FORTRAN 77/iX program differs in a few respects from an HP FORTRAN 77/V program, mainly because of differences in machine architecture.

Changed Features

Some features are implemented differently in HP FORTRAN 77/V and HP FORTRAN 77/iX. These are explained below.

Word Size

The main difference affecting conversion from HP FORTRAN 77/V to HP FORTRAN 77/iX is word size. The machine word size on computers running the MPE V operating system is 16 bits, while that on computers running MPE/iX is 32 bits. Because of the 32-bit word size, programs declaring data items to be `INTEGER*2` run slower on HP FORTRAN 77/iX than programs declaring them to be `INTEGER*4`. (A feature that improves performance in HP FORTRAN 77/V may decrease it in HP FORTRAN 77/iX.)

Note

Data items declared as INTEGER default to 32 bits in both FORTRAN 77/V and HP FORTRAN 77/iX (unless the `SHORT` compiler directive is used, which makes `INTEGER` data items 16 bits).

Floating-Point Data

HP FORTRAN 77/iX represents data items of types `REAL`, `DOUBLE PRECISION`, `COMPLEX`, and `DOUBLE COMPLEX` with the IEEE floating-point standard. HP FORTRAN 77/V uses a proprietary HP 3000 floating-point format for these items.

Uninitialized Variables

The MPE/iX Link Editor does not initialize stack space for all variables as the Segmenter does on MPE V. Therefore, uninitialized variables that do not cause problems on MPE V may cause programs to abort on MPE/iX based systems. To avoid this, ensure that all variables are properly initialized.

Alignment

The default alignment of data items larger than 16 bits is different in HP FORTRAN 77/iX and HP FORTRAN 77/V. The term *alignment* here refers to a data item's position in memory relative to the adjacent word boundaries. For example, in both versions of HP FORTRAN 77 an `INTEGER*4` variable is aligned by default on a word boundary in both versions of HP FORTRAN 77. Therefore, the alignment is on a 16-bit boundary in HP FORTRAN 77/V and on a 32-bit boundary in HP FORTRAN 77/iX (see Figure 8-1). The compiler accomplishes the alignment by leaving "holes" (unallocated memory locations) where necessary in the allocated memory space of a program. Consider the following declarations:

```
INTEGER*4    i4
CHARACTER*5  ch5
INTEGER*2    i2
REAL*8       d8
```

Figure 8-1 compares how these data items might be allocated in memory in HP FORTRAN 77/V and in HP FORTRAN 77/iX:

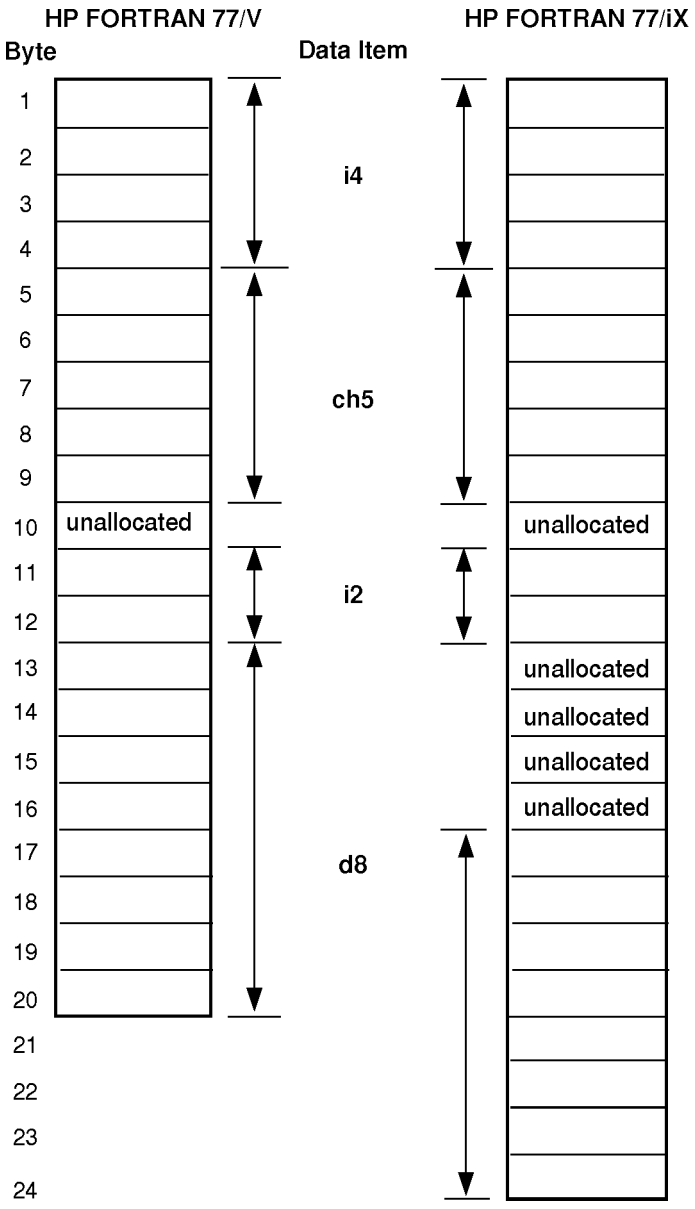


Figure 8-1. Data Alignment Comparison

In Figure 8-1, both versions of HP FORTRAN 77 assign the first hole in the same position after the character variable, because the following data item, which is declared as an `INTEGER*2`, is 16-bit aligned in both.

HP FORTRAN 77/iX assigns a second hole to align the double precision variable `D8` on a double-word (64-bit) boundary. In HP FORTRAN 77/V, on the other hand, double precision items need only be word (16-bit) aligned. In the absence of an `EQUIVALENCE` statement, the compiler leaves these holes and allocates memory to align each variable optimally.

An `EQUIVALENCE` statement can attempt to force illegal alignments. For example:

```
CHARACTER ch(12)
INTEGER*4 i, j
EQUIVALENCE (ch(1), i), (ch(6), j)
```

is illegal in both FORTRAN 77/V and HP FORTRAN 77/iX, because it attempts to align `j` on an 8-bit boundary (by forcing an equivalence between `j` and the sixth element of `ch`).

However, the following:

```
CHARACTER ch(12)
INTEGER*4 i, j
EQUIVALENCE (ch(1), i), (ch(7), j)
```

is legal in HP FORTRAN 77/V, but not in HP FORTRAN 77/iX, because it forces `j` to be aligned on a 16-bit boundary (by forcing an equivalence between `j` and the seventh element of `ch`), instead of on the 32-bit boundary dictated by HP FORTRAN 77/iX. The compiler responds to illegal alignment requests with the error message

```
ILLEGAL OR INCONSISTENT EQUIVALENCE STATEMENT
```

However, HP FORTRAN 77/iX allows the above use of `EQUIVALENCE` if the `HP3000_16 ON` or `HP3000_16 ALIGNMENT` compiler directive is used.

The change in alignment requirements affects items that are equivalenced differently from MPE/iX defaults. The following table shows all nine data types and their alignment requirements in HP FORTRAN 77/V and HP FORTRAN 77/iX:

Table 8-1.
Data Alignment on HP FORTRAN 77/V and HP FORTRAN 77/iX

Data Type	HP FORTRAN 77/V	HP FORTRAN 77/iX
`REAL*8`	16 bits	64 bits
`COMPLEX*16`	16 bits	64 bits
`COMPLEX*8`	16 bits	32 bits
`REAL*4`	16 bits	32 bits
`INTEGER*4`	16 bits	32 bits
`LOGICAL*4`	16 bits	32 bits
`INTEGER*2`	16 bits	16 bits
`LOGICAL*2`	16 bits	16 bits
`Character`	8 bits	8 bits
`Common`	16 bits	64 bits

Loading or storing data items that are not aligned on MPE/iX default boundaries as shown in this table requires the compiler to generate special code sequences to adjust for the differences in alignment. The `HP3000_16 ALIGNMENT` directive turns on generation of these sequences and allows equivalences that would otherwise be illegal in MPE/iX.

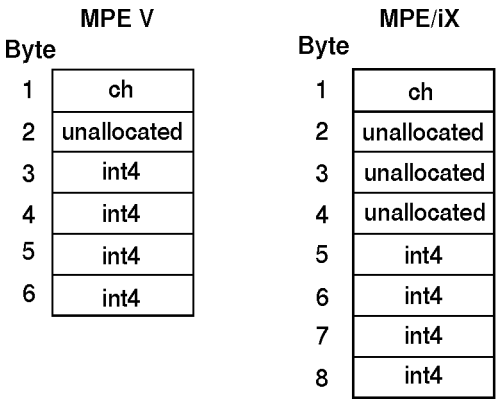
Common Blocks

Alignment also becomes an issue when common blocks are defined differently between program units. The compiler is required to allocate the items in a common block in the order of their occurrence in the `COMMON` statement. Holes are inserted in the allocated common area to align each variable according to its declared data type. If the type declarations for the variables in a given common block vary between program units, the holes may vary in size, according to the data types of the surrounding variables. Therefore, what was a size match for two definitions on MPE V may match a variable with a hole on MPE/iX. If the type definitions vary between the program units of a common block, the program may not be portable between machines with different alignment rules. Fortunately, this is not a common practice.

Here is an example showing alignment differences for common block variables in two program units. The declarations

```
SUBROUTINE sub1
COMMON /blk1/ch,int4
CHARACTER ch
INTEGER*4 int4
```

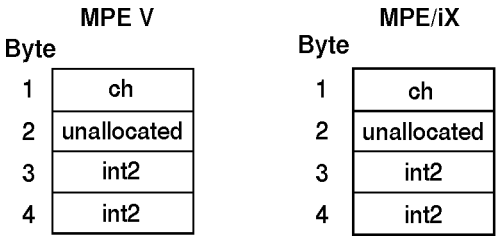
produce the allocations shown below on MPE V and MPE/iX:



The declarations

```
SUBROUTINE sub2
COMMON /blk1/ch,int2
CHARACTER ch
INTEGER*2 int2
```

produce the allocations shown below on MPE V and MPE/iX:



In the allocations above, note that for the common block `blk1`, `int2` overlays `int4` on MPE V but on MPE/iX `int2` overlays two unallocated bytes. Using the `HP3000_16 ALIGNMENT` option would resolve this alignment problem.

SYSTEM INTRINSIC Statement

Programs using the `SYSTEM INTRINSIC` statement to call MPE/iX intrinsic functions that are incompatible with their MPE V counterparts (`MYCOMMAND`, `CREATEPROCESS`, etc.) must modify the calls to those intrinsics. The changed intrinsics generally store procedure labels or addresses in 16-bit variables; these items are 32 bits in MPE/iX. Refer to the *MPE/iX Intrinsics Reference Manual* and the *Application Migration Guide* when using these intrinsics.

SEGMENT and LOCALITY Directives

The `LOCALITY` directive of FORTRAN 77/iX serves as a synonym for the `SEGMENT` directive of FORTRAN 77/V. The `SEGMENT` directive is also retained in FORTRAN 77/iX for compatibility with FORTRAN 77/V.

See "LOCALITY Directive" later in this chapter for more information.

SYSINTR Directive

The MPE V directive `SPLINTR` has been replaced in MPE/iX by `SYSINTR`.

Overlapping Character Substring Moves

FORTRAN 77/V allows overlapping character substring moves to have a ripple effect; FORTRAN 77/iX does not allow this ripple effect. For example, the following code has a different result on FORTRAN 77/V and FORTRAN 77/iX:

```
character ch*10
ch(1:1) = '*'
ch(2:10) = ch(1:9)
```

On FORTRAN 77/V, the character string `ch` is filled with asterisks (`*`). On FORTRAN 77/iX, the first and second positions contain asterisks and the remainder of the string is undefined.

If your FORTRAN 77/V program takes advantage of the ripple effect of overlapping character substring moves, use the `STRING_MOVE` option with the `HP3000_16` directive. (See the section "HP3000_16 Directive" later in this chapter for more information.) If the `STRING_MOVE` option is used and there are overlapping character substrings, the string is moved one character at a time. If `STRING_MOVE` is not used, a fast move is done. Therefore, to increase performance, do not use the `STRING_MOVE` option if your program does not take advantage of the ripple effect of character substring moves.

Removed Limitations

The MPE V operating system limits stack space to 65,535 bytes. The virtual memory mechanism of the MPE/iX operating system increases stack space to more than 2^{30} bytes.

USLINIT Directive

The `USLINIT` compiler directive is used in HP FORTRAN 77/V to empty the user subprogram library (`USL`) before placing any object code in it. The `USL` is a specially formatted file that contains a relocatable binary module for the main procedure, a relocatable binary module for each subprogram, a directory to record information about each of the relocatable binary modules in that library, and information about the program's data stack. The MPE V-based system compilers actively manage user subprogram libraries and write directly to these files.

The HP FORTRAN 77/iX compiler creates a new file containing a single object module for each compilation. It never appends to an existing `USL` file as a compiler on an MPE V-based system does. Furthermore, the need for a `USL` vanishes since an object module can function as an independent file. Therefore, the `USLINIT` directive does not exist in HP FORTRAN 77/iX.

You can find more information about user subprogram libraries in the *Link Editor/iX Reference Manual*.

MORECOM Directive

The `MORECOM` directive is used in HP FORTRAN 77/V to allow you to allocate more room in a data segment for common variables. The MPE/iX system automatically allocates a much larger area for common variables, so the `MORECOM` directive is not needed in HP FORTRAN 77/iX. See the *Link Editor/iX Reference Manual* for more information on data space.

New Features

Many features not available in HP FORTRAN 77/V are added in HP FORTRAN 77/iX. These are outlined below. For complete information about any of these features, see the *HP FORTRAN 77/iX Reference* manual.

HP3000_16 Directive

The HP3000_16 compiler directive specifies various options for compatibility with HP FORTRAN 77/V. These are the options:

`$HP3000_16 REALS`	Specifies MPE V format for floating-point data.
`$HP3000_16 ALIGNMENT`	Specifies alignment of noncharacter data on 16-bit boundaries.
`$HP3000_16 STRING_MOVE`	Specifies that overlapping character strings are moved.
`$HP3000_16 ON`	Specifies all the above options.
`$HP3000_16`	Same as `$HP3000_16 ON`.
`$HP3000_16 OFF`	Specifies none of the above options.

These options apply to an entire program unit and may not be changed within a program unit. For best results, compile entire programs with a consistent list of HP3000_16 options either by placing the directive before any other statements at the beginning of each source file, or by passing the option to the compiler through the INFO string.

OPTIMIZE Directive

The OPTIMIZE directive specifies optimization of object code. The directive provides these levels of code optimization:

`$OPTIMIZE OFF`	Level 0 optimization (does no optimizing). This is the default.
`$OPTIMIZE ON`	Same as `$OPTIMIZE LEVEL2`.
`$OPTIMIZE LEVEL1`	Optimizes only within each basic block.
`$OPTIMIZE LEVEL2`	Level 2 optimization with the following ASSUME settings:
	`ASSUME_NO_PARAMETER_O`
	`ASSUME_PARM_TYPES_MAT`
	`ASSUME_NO_EXTERNAL_PA`
	`ASSUME_NO_SIDE_EFFECT`

	See below for `ASSUME` descriptions.
`$OPTIMIZE LEVEL2_MIN`	Level 2 optimization with all the `ASSUME` settings at `OFF`.
`$OPTIMIZE LEVEL2_MAX`	Level 2 optimization with all the `ASSUME` settings at `ON`.

`$OPTIMIZE ASSUME_NO_PARAMETER_OVERLAPS`	No actual parameters passed to a subprogram overlap each other.
`$OPTIMIZE ASSUME_NO_SIDE_EFFECTS`	The current subprogram changes only local variables. It does not change any variables in `COMMON`, nor does it change parameters.
`$OPTIMIZE ASSUME_PARM_TYPES_MATCHED`	All of the actual parameters passed were the type expected by this subprogram.
`$OPTIMIZE ASSUME_NO_EXTERNAL_PARMS`	None of the parameters passed to the current subprogram are from an external space, that is, different from the user's own data space. Parameters can come from another space if they come from operating system space or if they are in a space shared by other users.

For further details about the `OPTIMIZE` directive and its options, see the *HP FORTRAN 77/iX Programmer's Guide*.

SYMDEBUG Directive

The MPE/iX operating system provides symbolic debugging for HP FORTRAN 77/iX programs. If the `SYMDEBUG` directive is used, the compiler inserts debugging information in the object file. Symbolic debugging is not compatible with code optimization; therefore, do not use both `$SYMDEBUG` and `$OPTIMIZE`.

LOCALITY Directive

The `LOCALITY` directive allows you to group procedures together in virtual memory to maximize throughput and minimize system overhead. `LOCALITY` is similar to the `SEGMENT` directive of HP FORTRAN 77/V. However, unlike `SEGMENT`, which is required in HP FORTRAN 77/V when large programs must be broken into segments, the `LOCALITY` directive is not required in HP FORTRAN 77/iX. HP FORTRAN 77/iX handles memory management transparently.

EXTERNAL_ALIAS and LITERAL_ALIAS Directives

The `EXTERNAL_ALIAS` and `LITERAL_ALIAS` directives allow you to redefine the names of functions and subroutines within a program unit.

UPPERCASE and LOWERCASE Directives

The `UPPERCASE` and `LOWERCASE` directives allow you to turn on or off shifting to uppercase or lowercase in FORTRAN 77 external names.

ON Statement and INTEGER*2 Conditions

In MPE/iX, `ON` statements specifying trap handling for `INTEGER*2` conditions are enabled only when the `$CHECK_OVERFLOW INTEGER_2` option (which is the default) is on. This restriction also applies to trap handling for `INTEGER` conditions if the `$SHORT` option is on.

9

Source Program Conversion

Certain changes may be required in an HP FORTRAN 77/V program before it can be compiled by the HP FORTRAN 77/iX compiler and executed properly on the MPE/iX system. These changes mainly involve data format.

The quickest way to run your program in native mode is to use the `HP3000_16` directive as explained below. However, to take full advantage of the performance increases of MPE/iX, you should eventually convert your data files to IEEE format and use HP Precision Architecture alignment (which is the default).

Using MPE V Binary Data Files or TurboIMAGE Databases

If your program accesses binary data files or databases, use one of the following directives:

`$HP3000_16 REALS`	Allows access to floating-point or double precision data in MPE V format but assumes MPE/iX data alignment. Use this directive when accessing binary data files that were created on an MPE V system and that contain floating-point values.
`$HP3000_16 ALIGNMENT`	Aligns noncharacter data on 16-bit boundaries. Use this directive if your program uses a TurboIMAGE database but none of the data items are floating-point or double precision.

Programs Packing Data Items with EQUIVALENCE

Programs that use `EQUIVALENCE` statements to pack data items relative to each other must either remove the packing dependency or use the `$HP3000_16` directive with the `ALIGNMENT` option.

Implied Equivalence

If a program implies equivalence by defining common blocks differently among program units, either use the `HP3000_16` directive with the `ALIGNMENT` option or revise the program. Note that the compiler cannot identify and flag this type of equivalence the way it does the explicit use of illegal equivalences.

Integers and Logicals

Integers and logicals can remain as originally defined or be converted to 32-bit items for improved performance. If you convert 16-bit integers and logicals to 32-bit items, check that equivalences, common variables, and parameter lists match correctly.

Using the Same Source Code

If you wish to compile the same source code using both the HP FORTRAN 77/V and HP FORTRAN 77/iX compilers, you can use the IF directive to specify conditional compilation. This will switch the declarations of items that significantly affect performance.

Example

```
$SET (MPE_IX = .TRUE., MPE_V = .FALSE.)
$IF (MPE_IX)
      INTEGER*4 i,j,k
      LOGICAL*4 tested
$ENDIF
$IF (MPE_V)
$SHORT
      INTEGER*2 i,j,k
      LOGICAL*2 tested
$ENDIF
```

`SEGMENT` directives can be left in the source code as synonyms for the `LOCALITY` directive of HP FORTRAN 77/iX. See "LOCALITY Directive" in Chapter 8 for further information.

10

Data File Conversion

ASCII files need no conversion. Binary (unformatted) files are also compatible except for floating-point items (of type `REAL`, `DOUBLE PRECISION`, `COMPLEX`, or `DOUBLE COMPLEX`). Binary files containing reals in the MPE V format can be used without any conversion if the program is compiled specifying `HP3000_16 REALS` or `HP3000_16 ON`. If you want full native mode floating-point performance, you should convert the binary files to IEEE floating-point format, as explained below.

Converting Binary Files to IEEE Format

Since only you know the format of your program's data files, you should write a short program (or two) to convert files. The easiest way is to write two short programs, one to read in the unformatted file and write it out as an ASCII (formatted) file, the other to reverse the process on the MPE/iX operating system in native mode.

The `FORMAT` statement used should specify more than the actual precision of the variables used. For example, use format descriptor `E14.8` for single precision and `E24.18` for double precision. The first program can be run either on the MPE V operating system or on MPE/iX. If on MPE/iX, run it either in compatibility mode or in native mode with `HP3000_16 REALS` turned on. Running the conversion program in native mode with `HP3000_16 REALS` can introduce a small amount of conversion error and some loss of precision because of the differences between IEEE and HP 3000 floating-point formats. However, the total error introduced should not exceed half of one decimal digit.

Here is an HP FORTRAN 77/V program that converts a data file to HP FORTRAN 77/iX format:

```
C  HP FORTRAN 77/V program to convert a direct access binary data
C     file containing floating-point data items.

      PROGRAM convert
      REAL x, y, z
      INTEGER*2 i, j
      INTEGER*4 i4, recnum
      DOUBLE PRECISION dp

      OPEN (12, FILE='mydata', ACCESS='DIRECT', FORM='UNFORMATTED',
     >      RECL=28, STATUS='OLD')
```

```
      OPEN (15, FILE='newdata', ACCESS='SEQUENTIAL', FORM='FORMATTED',
     >      STATUS='NEW')

      recnum = 1

C  Main loop reading and writing records until past the end of file,
C     which is an error on a direct access file.

10    CONTINUE             ! Do until end of file.
      READ (12, REC=recnum, ERR=99) i, x, i4, dp, j, y, z
      recnum = recnum + 1
      WRITE (15, 100) i, x, i4, dp, j, y, z ! Same I/O list as READ.
100   FORMAT (I7, E14.8, I11, E24.18, I7, 2E14.8)
      GOTO 10

C  Exit from loop, file finished.

99    CONTINUE
      STOP 'Now, :STORE off file "newdata" for transfer to MPE XL'
      END
```

Here is an HP FORTRAN 77/iX program that converts an HP FORTRAN 77/V data file:

```
C  HP FORTRAN 77/iX program to convert direct access binary data
C     file containing floating-point data items.

      PROGRAM convert
      REAL x, y, z
      INTEGER*2 i, j
      INTEGER*4 i4, recnum
      DOUBLE PRECISION dp

      OPEN (12, FILE='mydata', ACCESS='DIRECT', FORM='UNFORMATTED',
     >      RECL=28, STATUS='OLD') ! Native mode copy of original file.
      OPEN (15, FILE='newdata', ACCESS='SEQUENTIAL', FORM='FORMATTED',
     >      STATUS='NEW')        ! Input file restored from MPE V system.

      recnum = 1

C  Main loop reading and writing records until past the end of file.

10    CONTINUE                 ! Do until end of file.
      READ (15, FMT=100, ERR=99) i, x, i4, dp, j, y, z
      WRITE (12, REC=recnum) i, x, i4, dp, j, y, z ! Same I/O list as READ.
      recnum = recnum + 1
100   FORMAT (I7, E13.8, I11, E23.18, I7, 2E13.8)
      GOTO 10

C  Exit from loop, file finished.
```

```
99    CONTINUE
      WRITE (6, *) 'FILE "mydata" CREATED WITH', RECNUM - 1, ' RECORDS.'
      CLOSE (15, STATUS='DELETE')       ! Purge file used for transfer.
      END
```

Another way is to use the HPFPCONVERT system intrinsic. A program can read in the FORTRAN 77/V floating-point data, pass the data to HPFPCONVERT for translation to IEEE format, and then write out the data in native mode. This process can be performed completely in native mode as long as no operations are performed on the FORTRAN 77/V format floating-point data other than passing it to HPFPCONVERT. HPFPCONVERT is described in detail in the *MPE/iX Intrinsics Reference Manual*.

HPFPCONVERT loses only the actual difference in precision of the two floating-point formats (two bits for double precision, none in single).

Note

Some single precision values that are legal on MPE V may not be translated to IEEE single precision floating-point. The untranslatable values are those of a magnitude greater than 1^{38} or less than 10^{-43}.

11

Conversion Checklist

Here is a quick check. If your program does not use any of the features listed in this chapter, you need only recompile to run it in full native mode. If your program has any of these characteristics, it may need modification. Here are some points to check:

EQUIVALENCE Statement or Redefined Common Blocks

HP FORTRAN 77/iX has different alignment requirements from HP FORTRAN 77/V. In most programs, the compiler automatically allocates the alignment. However, your HP FORTRAN 77/V program might force a specific alignment by the use of `EQUIVALENCE` or implicitly expect a particular alignment by using different definitions of the same common block among program units. Typically, such explicit or implicit equivalences are used for three reasons: to change the data type of a variable, to pack variables into a logical record to be passed to system routines or routines coded in another language, or to save space.

An equivalence that changes the data type of a variable is safe and presents no problem in migration from HP FORTRAN 77/V to HP FORTRAN 77/iX, provided that the equivalenced variables or arrays are the same size in memory (for example, when equivalencing a `REAL*4` and an `INTEGER*4`). Multiple equivalences to an array, however, are likely to produce a dependency on the alignment convention of the compiler and generate error messages.

The type of equivalence that creates a packed record is commonly used to call database intrinsics and other system intrinsics. The compiler flags this as an error if an alignment is specified that conflicts with the requirements of HP FORTRAN 77/iX.

These are some solutions to alignment problems:

- If your program has alignment problems, specify `HP3000_16 ALIGNMENT`; this causes the compiler to use HP FORTRAN 77/V's alignment rules and to generate code to handle the resulting variables correctly. Specify the `ALIGNMENT` option before the first statement in your compilation file. This will cause the options to take effect in every program unit (mixing alignments generally produces undesirable results).
- If you call FORTRAN subprograms from other languages (for example, to receive elements of Pascal packed records and any

COBOL compatible type) that may align variables other than on their regular boundaries, specify `HP3000_16 ALIGNMENT`.

Note

Specifying `HP3000_16 ALIGNMENT` causes a slight reduction in performance, so do not use it unnecessarily.

- If your program uses equivalence to overlay one logical set of data items with another (planning for them to be used separately, for data space savings), consider modifying it not to do this. The practice is unnecessary in MPE/iX (with its large data space) and could introduce defects during software maintenance on such a fragile construct.

SYSTEM INTRINSIC Statement

Calling MPE/iX intrinsics in HP FORTRAN 77/iX is usually identical to calling MPE V intrinsics in HP FORTRAN 77/V.

- If your program uses system intrinsics that have been redefined in MPE/iX (such as `MYCOMMAND` or `CREATEPROCESS`), make the appropriate changes in calling methods. The usual reason an intrinsic is redefined is that the MPE V version uses an address stored in a 16-bit variable or array element. Addresses are at least 32 bits in MPE/iX. Because the FORTRAN 77/iX compiler does not know which intrinsics have been redefined, it does not produce a warning when one is used. See the *MPE/iX Intrinsics Reference Manual*, the *Programmer's Skills Migration Guide*, and the *Application Migration Guide* for the system intrinsics that are changed on MPE/iX. Migration tools such as the Object Code Scanner and the Runtime Event Logging Tool will also assist in identifying changed intrinsics.

Binary Files with HP 3000 Floating-Point Data

- If your program reads or writes HP 3000 floating-point data in binary form, you must specify `HP3000_16 REALS` in every program unit.

The performance of IEEE floating-point is usually at least 100 times faster than HP 3000 floating-point. If your application uses floating-point extensively, do the following:

- Convert such data files to native mode, as described in Chapter 10.
- Convert compatibility mode databases to native mode databases.
- Modify each program to take advantage of MPE/iX alignment conventions, as described in the first section of this chapter.
- Remove the `HP3000_16 ON` option if used.

Index

D