Chapter 2

Storage Mapping

This chapter contains two sections:

- "Alignment, Size, and Value Ranges" describes how the Fortran compiler implements size and value ranges for various data types as well as how data alignment occurs under normal conditions.
- "Access of Misaligned Data" describes two methods of accessing misaligned data.

Alignment, Size, and Value Ranges

Table 2−1contains information about various data types.

Table 2−1 Size, Alignment, and Value Ranges of Data Types

Type	Synonym	Size	Alignment	Value Range
BYTE	INTEGER*1	8 bits	Byte	-128…127
INTEGER*2		16 bits	Half word[a]	-32,768…32, 767
INTEGER	INTEGER*4[b]	32 bits	Word[c]	$-2^{31}…2^{31}-1$
LOGICAL*1		8 bits	Byte	0…1
LOGICAL*2		16 bits	Half word[a]	0…1
LOGICAL	LOGICAL*4[d]	32 bits	Word[c]	0…1
REAL	REAL*4	32 bits	Word[c]	See the first note below
DOUBLE PRECISION	REAL*8	64 bits	Double word[e]	See the first note below
COMPLEX	COMPLEX*8	64 bits	Word[c]	
DOUBLE COMPLEX		128 bits	Double word[e]	
CHARACTER		8 bits	Byte	-128…127

[a] Byte boundary divisible by two.

[b] When **-i2** option is used, type **INTEGER** would be equivalent to **INTEGER*2**.

[c] Byte boundary divisible by four.

[d] When **-i2** option is used, type **LOGICAL** would be equivalent to **LOGICAL*2**.

[e] Byte boundary divisible by eight.

f

The following notes provide details on some of the items in Table 2−1

- Table 2−2lists the approximate valid ranges for **REAL** and **DOUBLE**.

Table 2−2 Valid Ranges for REAL and DOUBLE Data Types

Range	REAL	DOUBLE
Maximum	$3.40282356 * 10^{38}$	$1.7976931348623158 * 10^{308}$
Minimum normalized	$1.17549424 * 10^{-38}$	$2.2250738585072012 * 10^{-308}$

Minimum denormalized	$1.40129846 * 10^{-46}$	$2.2250738585072012 * 10^{-308}$

Note: When the compiler encounters a **REAL*16** declaration, it issues a warning message. **REAL*16** items are allocated 16 bytes of storage per element, but only the first 8 bytes of each element are used. Those 8 bytes are interpreted according to the format for **REAL*8** floating numbers.

- When the compiler encounters a **REAL*16** constant in a source program, the compiler issues a warning message. The constant is treated as a double precision (**REAL*8**) constant. **REAL*16** constants have the same form as double precision constants, except the exponent indicator is **Q** instead of **D**.
- Table 2−1states that **DOUBLE PRECISION** variables always align on a double−word boundary. However, Fortran permits these variables to align on a word boundary if a **COMMON** statement or equivalencing requires it.
- Forcing **INTEGER**, **LOCICAL**, **REAL**, and **COMPLEX** variables to align on a halfword boundary is not allowed, except as permitted by the **-align8**, **-align16**, and **-align32** command line options. See Chapter 1, "Compiling, Linking, and Running Programs.".
- A **COMPLEX** data item is an ordered pair of real numbers; a double−complex data item is an ordered pair of double−precision numbers. In each case, the first number represents the real part and the second represents the imaginary part.
- **LOGICAL** data items denote only the logical values TRUE and FALSE (written as **.TRUE.** or **.FALSE.**). However, to provide VMS compatibility, **LOGICAL*1** variables can be assigned all values in the range -128 to 127.
- You must explicitly declare an array in a **DIMENSION** declaration or in a data type declaration. To support dimension, the compiler
  - allows up to seven dimensions
  - assigns a default of 1 to the lower bound if a lower bound is not explicitly declared in the **DIMENSION** statement
  - creates an array the size of its element type times the number of elements
  - stores arrays in column−major mode
- The following rules apply to shared blocks of data set up by the **COMMON** statements:
  - The compiler assigns data items in the same sequence as they appear in the common statements defining the block. Data items will be padded according to the alignment switches or the default compiler. See "Access of Misaligned Data" for more information.
  - You can allocate both character and noncharacter data in the same common block.
  - When a common block appears in multiple program units, the compiler allocates the same size for that block in each unit, even though the size required may differ (due to varying element names, types, and ordering sequences) from unit to unit. The size allocated corresponds to the maximum size required by the block among all the program units except when a common block

is defined by using **DATA** statements, which initialize one or more of the common block variables. In this case the common block is allocated the same size as when it is defined.

Access of Misaligned Data

The Fortran compiler allows misalignment of data if specified by the use of special options.

As discussed in the previous section, the architecture of the IRIS−4D series assumes a particular alignment of data. ANSI standard Fortran 77 cannot violate the rules governing this alignment. Common extensions to the dialect, particularly small integer types, allowing intermixing of character and non−character data in **COMMON** and **EQUIVALENCE** statements and mismatching the types of formal and actual parameters across a subroutine interface, provide many opportunities for misalignment to occur.

Code using the extensions that compiled and executed correctly on other systems with less stringent alignment requirements may fail during compilation or execution on the IRIS−4D. This section describes a set of options to the Fortran compilation system that allow the compilation and execution of programs whose data may be misaligned. Be forewarned that the execution of programs that use these options will be significantly slower than the execution of a program with aligned data.

This section describes the two methods that can be used to create an executable object file that accesses misaligned data.

Accessing Small Amounts of Misaligned Data

Use the first method if the number of instances of misaligned data access is small or to provide information on the occurrence of such accesses so that misalignment problems can be corrected at the source level.

This method catches and corrects bus errors due to misaligned accesses. This ties the extent of program degradation to the frequency of these accesses. This method also includes capabilities for producing a report of these accesses to enable their correction.

To use this method, keep the Fortran front end from padding data to force alignment by compiling your program with one of two options to *f77.*

- Use the **-align8** option if your program expects no restrictions on alignment.
- Use the **-align16** option if your program expects to be run on a machine that requires half−word alignment.

You must also use the misalignment trap handler. This requires minor source code changes to initialize the handler and the addition of the handler binary to the link step (see the *fixade*(3f) man page).

Accessing Misaligned Data Without Modifying Source

Use the second method for programs with widespread misalignment or whose source may not be modified.

In this method, a set of special instructions is substituted by the IRIS−4D assembler for data accesses whose alignment cannot be guaranteed. The generation of these more forgiving instructions may be opted for each source file.

You can invoke this method by specifying of one of the alignment options (**-align8**, **-align16**) to *f77* when compiling any source file that references misaligned data (see the *f77*(1) man page). If your program passes misaligned data to system libraries, you might also need to link it with the trap handler. See the *fixade*(3f) man page for more information.