Chapter 8

Storage Association and Global Data

Control over the location of objects in memory is limited in CM Fortran. However, the EQUIVALENCE statement and the COMMON statement provide some control over the relative order and use of objects in front-end memory. The COMMON statement also is used to make CM and front-end variables *global*, that is, accessible from several program units.

The *storage association* of two or more objects in memory is a full or partial memory sharing relationship among those objects: if two or more objects are storage associated, a change in the value of one object changes the value of all objects with which it is associated. Storage association is established at compile time among variables in common or among variables local to the main program. An EQUIVALENCE statement, a COMMON statement, or an ENTRY statement may cause storage association among front-end variables. Storage association is described in the next section.

Another form of association is *argument association*, which is established at run time between the name of an actual argument in a procedure reference and the name of a corresponding dummy argument. Argument association is described in Section 12.6.3.

Two objects are said to be *associated* if they are storage associated or argument associated.

8.1 Storage Association of Objects

Storage association is the association of two or more objects that occurs when memory allocated for those objects overlaps. A *storage sequence* is a sequence of front-end memory storage units; the *size* of a storage sequence is the number of storage units in the sequence. The front-end variables of a common block form a storage sequence. Storage sequences establish the relative storage locations in front-end memory of variables, array elements, substrings, common blocks, and arguments.

Copyright © 1989–1994 Thinking Machines Corporation

Two storage sequences *S* and *T* are *storage associated* if they share at least one storage unit. If the *i*th storage unit of *S* is the same as the *j*th storage unit of *T*, this causes the (*i*+*k*) th storage unit of *S* to be the same as the (*j*+*k*) th storage unit of *T*, for each integer *k* such that $1 \le i+k \le size(S)$ and $1 \le j+k \le size(T)$.

Two objects are *totally associated* if they have the same storage sequence. Two objects are *partially associated* if they are associated but not totally associated. Partial association may exist only between two character variables of different lengths, or between two numeric variables with storage sequences of different lengths.

A storage unit is either a *numeric storage unit* or a *character storage unit*. The number and kind of front-end storage units occupied by objects of each type in a storage sequence is given below.

INTEGER*4, REAL, and LOGICAL scalars — 1 numeric storage unit

Scalar objects of type integer, single-precision real, and logical each use a single numeric storage unit.

INTEGER*8, DOUBLE PRECISION, COMPLEX — 2 contiguous numeric storage units

The real part of a single-precision complex object occupies the first storage unit, and the imaginary part occupies the second storage unit.

DOUBLE COMPLEX — 4 contiguous numeric storage units

The real part of a double-precision complex object occupies the first two storage units, and the imaginary part occupies the last two storage units.

CHARACTER*N — *N* contiguous character storage units

Each character in an object of type character uses one character storage unit. Character objects occupy a contiguous sequence of character storage units, the first character in the object occupying the first storage unit, the second character occupying the second storage unit, and so on.

Front-end array of size *N* — *N* contiguous storage units

Front-end arrays occupy a sequence of contiguous storage units of the kind appropriate to their type, in array element order.

CM array — no storage sequence

CM arrays have no associated storage sequence.

Objects having different kinds of storage units must not be storage associated.

Version 2.2, October 1994
Copyright © 1989–1994 Thinking Machines Corporation

CM arrays are assigned storage units that are not contiguous, and do not form a storage sequence. The processor layout of an array in CM memory depends upon its shape, the size of the machine in which it is allocated, and the presence of compiler directives. The COMMON statement provides the ability to make one or more CM arrays visible to several program units, but gives no means for storage association of arrays.

8.2 COMMON Statement

The COMMON statement (R8-1) specifies blocks of physical storage, called *common block*s, that may be accessed by any of the program units forming a program. The COMMON statement allows program units to share front-end data or CM array data. Common blocks may be named or unnamed; named common blocks are part of *named common*, and unnamed common blocks are part of *blank common*. COMMON statements cause front-end objects in one program unit to be storage associated with front-end objects in another program unit.

A *common-stmt* is: R8-1
COMMON *common-block-item(s)*

A *common-block-item* is: R8-2
[/ [common-block-*name] /] common-block-object(s)*

A *common-block-object* is: R8-3
variable-*name [(explicit-shape-spec(s))]*

An *explicit-shape-spec* is: R5-3
[lower-bound :] upper-bound

- Only one appearance of a given variable-*name* is permitted in all *common-block-object(s)* lists within a program unit or interface block.
- A *common-block-object* must not be a dummy argument, an automatic array, an array of zero size, or a function name.
- The *upper-bound* (and *lower-bound*, if present) of each *explicit-shape-spec* of a *common-block-object* (R8-3) must be an integer constant expression.
- If any object of a common block is of type character, then all the objects in that common block must be of type character.

A single COMMON statement may declare objects in several different named common blocks as well as in blank common. Each *common-block-item* (R8-2) of a COMMON statement designates one or more objects for allocation either in named common or in blank common. If a *common-block-item* begins with a common-block-*name*, then all objects

Copyright © 1989–1994 Thinking Machines Corporation

named in its *common-block-object(s)* list are allocated in the specified common block. If the common-block-*name* is omitted but the slashes are present, then all objects named in the following *common-block-object(s)* list are allocated in blank common. The first *common-block-item* of a COMMON statement may omit both the common-block-*name* and the bracketing slashes, in which case all objects named in the *common-block-object(s)* list of that item are allocated in blank common.

Any common block name or blank common designation may occur more than once among all the *common-block-item(s)* of all the COMMON statements in a program unit or interface block. The *common-block-object(s)* list following each successive appearance of the same common block name is treated as a continuation of the list for that common block. Similarly, each *common-block-object(s)* list specifying objects in blank common is treated as a continuation of blank common.

A COMMON statement may appear in an interface block. This may be useful for specifying the sizes of automatic arrays.

Examples of COMMON statements:

```
COMMON /BLOCKA/ A, B, D(10,30)      ! named common
COMMON I, J, K                      ! blank common
```

8.2.1 Common Block Storage Sequence

For each common block, the front-end array variables declared in the block form a *common block storage sequence*, as follows:

(1) A storage sequence is formed consisting of the storage sequences of all front-end objects in the common block object lists for the common block. The order of the storage sequence is the same as the order of the appearance of the common block object lists in the program unit.

(2) The storage sequence formed in (1) is extended to include all storage units of any storage sequence associated with it by means of EQUIVALENCE statements (see Section 8.3). The sequence may be extended only by associating other storage sequences with it that extend its length beyond the last storage unit in the sequence (it may not be extended by adding to the front). Objects associated with an object in a given common block are considered to be in that common block.

The *size* of a common block is the size of its common block storage sequence, including any extensions of the sequence resulting from the use of EQUIVALENCE statements.

Version 2.2, October 1994
Copyright © 1989–1994 Thinking Machines Corporation

Within all the program units of a program, all common blocks with the same name share the same storage sequence; similarly, all the blank common blocks declared in different program units share the same storage sequence. This results in the *common association* of front-end objects in different program units.

NOTE: CM array variables appearing in a common block *do not* form a storage sequence, and only a simple form of common association exists among CM array variables appearing in a common block.

8.2.2 Differences between Named and Blank Common Blocks

A blank common block has the same properties as a named common block, with the following exceptions:

- Named common blocks of the same name must be of the same size in all program units of a program in which they appear. Each must declare the same number of storage units of front-end data and the same number of storage units of CM data.
- Blank common blocks may be of different sizes in each of the program units of the program in which they appear. The front-end storage sequence defined by blank common can be of different sizes in different program units, but CM array data declared in blank common must be of the same size in all program units that use blank common.
- An object in a named common block may be initially defined in a block data program unit (Section 9.2), but objects in blank common must not be initially defined.

  Within a block data program unit (Section 9.2), an object in a named common block may be initially defined using a `DATA` statement (Section 9.1) or a type declaration statement that specifies the `DATA` attribute (Section 5.4) and an initial value for the object.

The processor layout and shape of every CM array declared in a common block must be identical in every common block, whether named or blank common.

8.3 EQUIVALENCE Statement

The `EQUIVALENCE` statement (R8-4) specifies the storage association of two or more variables in a program unit. Names of CM arrays must not appear in an `EQUIVALENCE` statement.

Copyright © 1989–1994 Thinking Machines Corporation

An *equivalence-stmt* is: R8-4

EQUIVALENCE *equivalence-set(s)*

An *equivalence-set* is: R8-5

(*equivalence-object [, equivalence-object]...*)

An *equivalence-object* is one of: R8-6

- variable-*name*
- *array-element-designator*
- *substring-designator*

- An *equivalence-object* must not be the name of a dummy argument or an automatic object, an array of zero size, or a CM array; if it is an *array-element-designator*, the parent array must not be a CM array.

- Each subscript or substring range expression in an *equivalence-object* must be an integer constant expression.

An EQUIVALENCE statement specifies that the storage sequences of the objects whose names appear in an *equivalence-set* (R8-5) have the same first storage unit. This causes the storage association of the objects in the *equivalence-set* and may cause storage association of other objects. Objects in the same equivalence set are said to be *equivalenced*.

Using a whole array name as an *equivalence-object* has the same effect as using an array element designator that identifies the first element of the array.

If the objects in an equivalence set do not have the same type or attributes, the EQUIV–ALENCE statement does not cause type conversion, nor does it imply mathematical equivalence. For example, if a scalar and an array are equivalenced, the scalar does not have array properties and the array does not have the properties of a scalar.

An EQUIVALENCE statement causes storage association of objects only within a single program unit, unless one of the equivalenced objects is also in a common block. In this case, the storage sequence of the common block is extended, if necessary, to include all storage units of the storage sequences of all objects in the equivalence set (see Section 8.2.1).

An EQUIVALENCE statement may appear in an interface block. This may be useful for specifying the sizes of automatic arrays.

An example illustrates some of these ideas:

```
REAL A(200), CS(2)
COMPLEX C
COMMON /FOO/ B(100)
EQUIVALENCE ( C, CS )
EQUIVALENCE ( A, B )    ! Extends the storage sequence of FOO
```

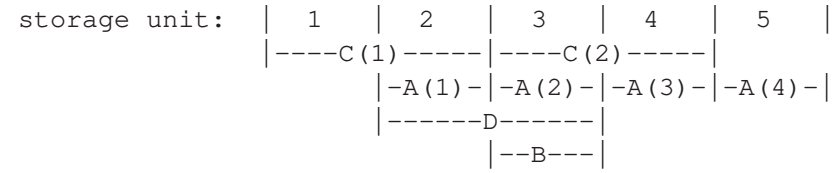

Copyright © 1989–1994 Thinking Machines Corporation

The first EQUIVALENCE statement causes the scalar variable C of type complex to be totally associated with the array CS of type real; CS(1) is the real part of C and CS(2) is the imaginary part. The second EQUIVALENCE statement causes the first 100 elements of array A to occupy the same storage locations as array B. It also causes the common block storage sequence of the named common block FOO to be extended from 100 to 200 numeric storage units.

In the example

```
REAL A(4), B
COMPLEX C(2)
DOUBLE PRECISION D
EQUIVALENCE ( C(2), A(2), B ), ( A, D )
```

the third storage unit of C, the second storage unit of A, the (only) storage unit of B, and the second storage unit of D are specified as the same. The storage sequences may be illustrated as:



A(2) and B are totally associated. The following arrays and array elements are partially associated: A(1) with C(1), A(2) with C(2), A(3) with C(2), B with C(2), D with A(1), D with A(2), B with D, C(1) with D, and C(2) with D. Note that although C(1) and C(2) are each associated with D, C(1) and C(2) are not associated with each other.

8.3.1 Restrictions on EQUIVALENCE Statements

An equivalence statement must not specify that the same storage unit is to occur more than once in a storage sequence. For example, the sequence

```
REAL, DIMENSION(2) :: A
REAL :: B
EQUIVALENCE ( A(1), B ), ( A(2), B )          ! INCORRECT
```

is prohibited, as it would specify the same storage unit for A(1) and A(2).

Copyright © 1989–1994 Thinking Machines Corporation

An EQUIVALENCE statement must not specify that contiguous storage units are to be non-contiguous. For example, the following sequence is prohibited:

```
REAL A(2)
DOUBLE PRECISION D(2)
EQUIVALENCE ( A(1), D(1) ), ( A(2), D(2) )   ! INCORRECT
```

The EQUIVALENCE statement above would require that an additional numeric storage unit be allocated between array elements A(1) and A(2), since the elements of D require twice the number of storage units as the elements of A.

8.3.2 Restrictions on EQUIVALENCE of COMMON Variables

An EQUIVALENCE statement must not cause the storage sequences of two different common blocks to be associated. Equivalence association must not cause a common block storage sequence to be extended by adding storage units preceding the first storage unit of a common block (i.e., the first object specified in a COMMON statement for the common block). For example, the following sequence is not permitted:

```
COMMON /X/ A
REAL B(2)
EQUIVALENCE ( A, B(2) )    ! INCORRECT
```

8.3.3 EQUIVALENCE of Character Objects

Objects of type character may be storage associated through the use of an EQUIVALENCE statement only with other objects of type character. The lengths of the equivalenced character objects are not required to be the same.

An EQUIVALENCE statement associating objects of type character specifies that the storage sequences of the character objects whose names appear in an equivalence set have the same first character storage unit. This causes the storage association of the objects in the equivalence set and may cause storage association of other character objects. Any adjacent

Version 2.2, October 1994
Copyright © 1989–1994 Thinking Machines Corporation

characters in the associated objects may have the same character storage unit and thus may be storage associated. In the example

```
CHARACTER*4 :: A, B
CHARACTER*3 :: C(2)
EQUIVALENCE ( A, C(1) ), ( B, C(2) )
```

the storage association of A, B, and C can be illustrated graphically as:

```
storage unit:    | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
               A |--- --- --- ---|
               B             |--- --- --- ---|
               C |--- --- ---|--- --- ---|
```

A, B, and C are partially associated.

Partial association of character objects occurs when some, but not all, of the storage units of the objects are the same. In the example below, A, B, and C are partially associated.

```
CHARACTER A*4, B*4, C*3
EQUIVALENCE ( A(2:2), B, C )
```


Copyright © 1989–1994 Thinking Machines Corporation