bcc	title	prefix/class-number.revision
	FACILITIES FOR ALLOCATION IN SPL	SPLAL/W-16

checked	authors	approval date	revision date
L. Peter Deutsch	Butler W. Lampson		
checked		classification	
[illegible]	[illegible]	Working Paper	
approved		distribution	pages
[illegible]		Company Private	

ABSTRACT and CONTENTS

Additions to SPL are defined which permit the location of code, arrays, scalars, common blocks and local environments in core to be specified. The default allocation strategies are described.

SPL provides facilities for fixing the core location of the various objects it constructs. This is done with a variety of constructs mostly involving the use of '=' (for equivalencing) in declaration statements.

<u>Code and Common Blocks</u>

The normal allocation strategy is as follows. Core is allocated upward from G in the following main regions:

1) writeable global storage, starting at G. Here are put variables, both scalar and array, which appear in common blocks, and fixed local environments. Some of the first 128 words may also be used for field descriptors and common array descriptors. The first few words, of course, are used for objects whose location is fixed by the hardware, like the pop transfer vector descriptor.
2) read-only storage. Here are put the fixed contents of common blocks (i.e. initialized objects), function descriptors, and code for function blocks.
3) the stack and allocatable storage. Common arrays and fixed local environments may also be put at the beginning of this area if area 1 would otherwise become so large that the function descriptors could not be kept within 16K of G. Alternatively, the area may be appended to area 1 if the total size of the two is somewhat less than 16K.

Space for code is allocated to functions in the order in which they appear in the source. Space for scalar objects in common blocks is allocated in the order in which they appear; arrays and dope are allocated at the compiler's discretion unless something is specified explicitly.

The syntax of function and common statements is augmented as follows (cf SPL/M-1.1 p. 13,17).

```
    function:statement  =  ['FIXED'] ftype ('FUNCTION' [equiv]
                                 / 'ENTRY')
                           identifier '(' [declare:clause:list
                                 ')' [equiv]
                           [function:location]
    common:statement    =  'COMMON' identifier [equiv]
    equiv               =  '=' expr
```

The expression must evaluate to an integer

The second <equiv> in the function:statement tells where to start the code for the function. This location must be beyond the location where the function would normally be allocated; otherwise there is an error comment and the declaration is ignored. Allocation of later functions continues from the end of this one.

The first <equiv> is allowed only for a FIXED function and tells where to put the local environment. The <equiv> in a common:statement tells where to put the stuff in the common block; it causes everything in the block to be allocated there, whether read-only or not.

Magic symbols are provided to enable the programmer to position things in relation to other things:

LAST'CODE has as value the next location after the space in read-only storage occupied by the immediately preceding block.

LAST'GLOBAL does the same for writeable global storage

Thus the programmer can allocate a sequence of common blocks at 4000B with

```
    COMMON A = 4000B; ...
    COMMON B = LAST'GLOBAL;...
            .
            .
            .
```

Scalars

An equivalence (SPL/M-1.1 p.6) can be used to fix the location of a scalar or an array descriptor by writing an integer-valued expression for the object of the equivalence. Thus

```
    DECLARE A = 40B, B[30] = 41B;
```

allocates A at 40 and the descriptor for the array B at 41 and 42. The array itself is allocated according to the default rules.

Field descriptors are normally allocated in the first 128 words of the global environment by the compiler if there is room. This allocation can be suppressed and the field allocated in the function like any other constant by prefixing FIXED to [SIGNED] FIELD in the declaration.

Arrays

The syntax of <dimension> (SPL/M-1.1 p. 3) is extended as follows:

```
    dimension = '[' expr $(','expr) [':'[expr][','expr]] ']'
```

The last optional expression tells where to put the first word of the array. Also, the expressions in the list of subscripts bounds may be null in a formal parameter; in this case the list serves simply to establish the dimensionality of the array.