The changes to the Mesa Language for Mesa 14.0 are:

New options for machine-dependent record declarations.

Changes that result from allocating global frames from outside the main data space (MDS). The Mesa 14.0 compiler can produce code in which global frames are allocated in the MDS if the 'o switch (meaning old code) is used. With this switch, these changes do not apply.

3 Common constructed data types

3.3.7 Machine dependent records

The syntax for machine-dependent record declarations has been extended to facilitate writing portable Mesa code. One can now specify:

The size of the unit for the word part of a field specification (e.g. the unit for 2 in (2:0..15)).

Whether bit 0 of a word refers to the most significant bit or the least significant bit

The syntax is:

```
dependentRec    ::=
    BitNumOption UnitOption MACHINE DEPENDENT RECORD |
    UnitOption BitNumOption MACHINE DEPENDENT RECORD |
    BitNumOption MACHINE DEPENDENT RECORD |
    UnitOption MACHINE DEPENDENT RECORD |
    MACHINE DEPENDENT RECORD | empty

BitNumOption    ::= MSBIT | LSBIT | NATIVE
UnitOption      ::= WORD8 | WORD16 | WORD32
```

Examples:

```
T1: TYPE = MSBIT WORD16 MACHINE DEPENDENT RECORD [...];
T2: TYPE = WORD8 NATIVE MACHINE DEPENDENT RECORD [...];
T3: TYPE = WORD32 MACHINE DEPENDENT RECORD [...];
```

The **UnitOption** defines the unit size for the word part of a field specification. The**BitNumOption** (MSBIT/LSBIT) controls how the bits are numbered within a word of the size specified in the **UnitOption**. For example, if the **UnitOption** is WORD8 then bit 2 in an MSBIT declaration becomes bit 5 if the declaration is

rewritten as an LSBIT declaration. If the **UnitOption** is WORD32, however, then bit 2 in an MSBIT declaration becomes bit 29 if the declaration is rewritten as an LSBIT declaration. NATIVE means whichever bit numbering is native to the target architecture (MSBIT for PrincOps).

If the **BitNumOption** or the **UnitOption** is omitted, then the defaults are NATIVE WORD16, which are backwards compatible.

Restrictions:

With this definition of MSBIT/LSBIT certain restrictions apply when the **BitNumOption** is not the native (default) bit numbering for the machine (e.g., not MSBIT on PrincOps machines). These restrictions are in addition to the restrictions regarding record size and field alignment.

When using the non-native **BitNumOption**, it is possible to declare records in which the bit numbers for a field are all contiguous, but the field would actually be split into two parts. Such declarations are illegal. The following declaration, for example, is NOT legal for PrincOps.

```
T: TYPE = LSBIT WORD8 MACHINE DEPENDENT RECORD [
    n(0:0..14): NATURAL,
    b(1:7..7): BOOLEAN];
```

Field *n* of this type would occupy the entire first byte of the record, and the least significant 7 bits of the second byte, thus leaving the most significant bit of the second byte as a gap in the field (filled in by field *b*), as shown in Figure 3-1.

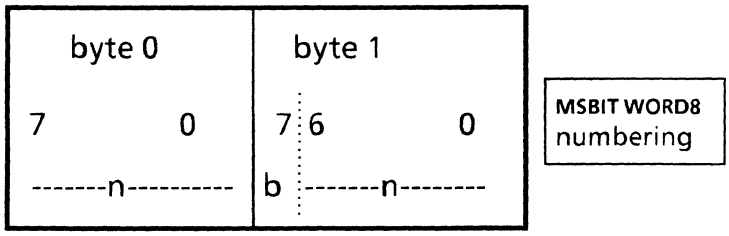

Figure 3-1

This declaration is legal, however, on a machine in which LSBIT is the native bit numbering.

The following similar declaration, using WORD16, is legal even on machines whose native bit numbering is MSBIT.

```
T: TYPE = LSBIT WORD16 MACHINE DEPENDENT RECORD [
    n(0:0..14): NATURAL,
    b(0:15..15): BOOLEAN];
```

In this case, the bit numbering is reversed (from the native bit number) for the entire 16-bit word, as shown in figure 3.2.

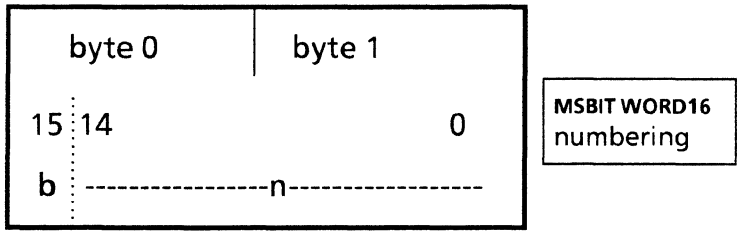

Figure 3-2

The general rule for insuring that a field is not split is:

> If the non-native bit numbering option is used, and n is the number of bits specified in the unit option (i.e. WORDn), then:
>
> 1) a field smaller than an n-bit word can not cross an n-bit word boundary.
>
> 2) a field occupying more than an n-bit word must begin at bit zero of an n-bit word and its size must be a multiple of n bits.

A declaration containing a split field will generate the error message:

> *<field name>* is a split field

To avoid gaps in a record, when the non-native **BitNumOption** is used, the total length of the record must be a multiple of the length specified in the **UnitNumOption**. For example, on a machine whose native bit numbering is **MSBIT** (e.g,. PrincOps), the following declaration has a gap.

```
T: TYPE = LSBIT WORD16 MACHINE DEPENDENT RECORD [
  c1(0:0..7), c2(0:8..15), c3(1:0..7): CHARACTER];
```

When rewritten with **MSBIT** notation, the gap becomes more evident.

```
T: TYPE = MSBIT WORD16 MACHINE DEPENDENT RECORD [
  c1(0:8..15), c2(0:0..7), c3(1:8..15): CHARACTER];
```

For variant records, if there is a tag field then it must be the first field in the variant part of the record. When using the non-native **BitNumOption**, extra care must be taken to insure this requirement is met. In the following declaration, the tag field is not the first field in the variant part of the record, but rather is between the two fields of each variant if the target machine is PrincOps.

```
T: TYPE = LSBIT WORD16 MACHINE DEPENDENT RECORD [
variant(0:0..31): SELECT tag(0:0..0) FROM
  a1 => [n(0:1..15): NATURAL, i(1:0..15): INTEGER],
```

```
    a2 = > [n(0:1..15): NATURAL, c(1:0..15): CARDINAL],
ENDCASE];
```

3.4 The types POINTER and LONG POINTER

Since Global frames do not reside in the main data space (MDS), short pointers can not be used to indirectly access global variables.

3.4.2 String literals and string expressions

The example at the bottom of page 3-34 is no longer legal. When applied to a global variable, the @ operator returns a long pointer type. In this example, the variables *pointer1*, and *pointer2* must be decalred as **LONG POINTER TO INTEGER**.

6 Other data types and storage management

6.1.1 String literals and string expressions

A string literal allocated in the global frame can be assigned only to a **LONG STRING**. Similarly, a global string literal can not be passed as an actual parameter to a procedure whose formal parameter is a short string.

6.1.2 Declaring strings

A string that points to a **StringBody** allocated in a global frame must be a **LONG STRING**. The following string initialization:

```
s: STRING ← [30];
```

is allowable only inside a procedure, not in the mainline code (unless it is within a nested block in the mainline code).

6.2.1 Array descriptor types

A descriptor for an array in the global frame must be a **LONG DESCRIPTOR**. When applied to a global array, the **DESCRIPTOR** operator can only yield a **LONG DESCRIPTOR** type.

I GENERAL TOOLS

4 Executive

4.2.2 Command interface

CacheAddress maintains the network address cache that is used with the AddressTranslation interface. CacheAddress allows one to create, list, load, store, and manage the network address cache.

The command syntax is:

```
>CacheAddress  command/arg command/arg
```

Command/Arg

SetSize/n	address cache size is set to n.
GetSize	returns address cache size.
Flush	flushes the content of the cache, size remains the same.
List	lists contents of the cache.
Certify/entry	certifies entry in clearinghouse, entry may contain '*
Load/file	loads contents of file into cache.
Store/file	stores contents of cache into file.
Statistics	lists local statistics.

```
CacheAddress SetSize/20 List Store/foo.cache
```

Creating cache files

To set up your machine to use CacheAddress, do the following:

1. Type into the Executive:

   ```
   >CacheAddress SetSize/20
   ```

   This will set the size of the cache to 20.

2. Run for a day with this cache. The first time you lookup a machine address, it will be placed into the cache. To see the cache at any point, type into the Executive:

   ```
   >CacheAddress List
   ```

3. After running CacheAddress for awhile, create a cache file by the command:

```
>CacheAddress Store/<>Address.cache
```

This will place the current contents of the cache into the file Address.cache.

4. At this point, place into your user.cm InitialCommand: section:

```
[System]
InitialCommand: ... ; CacheAddress SetSize/20
load/Address.cache; ...
```

Certifying your address cache

At some point your address cache may become invalid because an address in the clearinghouse has changed. To validate your entire current address cache, type:

```
>CacheAddress Certify
```

This will cause all entries in your cache to be validated. If you wish only to certify a single entry (Goofy), use :

```
>CacheAddress Certify/"Goofy:OSBU North:Xerox"
```

OR

```
>CacheAddress Certify/Goofy
```

Patterns can also be used to certify entries. '* will match zero or more of any letter, and '# will match any single character. Make sure to quote the asterisk in the Executive, otherwise it will match files on your disk.

```
>CacheAddress Certify/G'*
```
(this will certify all name starting with 'G')

If you keep your address cache stored in a file, you will want to update you cache file after doing a Certify. Example:

```
>CacheAddress Store/address.cache
```

13 Compare

Compare will now correctly compare files on a remote NS File Server.

III SYSTEM BUILDING TOOLS

19 Compiler

With Pilot 14.0, global frames are not allocated from the main data space (MDS). This architectural change causes some small semantic changes to the Mesa language. These language changes are documented in the Mesa Language Manual Change Summary.

19.2.2 Switches

The Mesa 14.0 compiler now has an /o switch, which causes the compiler to generate code for pre-Mesa 14.0. When this switch is used, the compiler will create a module with a global frame in the MDS. The restrictions noted in the Mesa Language Change Summary for Mesa 14.0 do not apply if this switch is used.

The default for the /o switch is false.

21 MakeBoot

With Pilot 14.0, MakeBoot changed to reflect the new architecture. In addition, MakeBoot now uses the runtime loader in Pilot to load the input object files. Because of the differences in the Pilot runtime loader and the old MakeBoot loader, there are some new restrictions.

The new restriction with MakeBoot is all input object files must be bound with their code. With the pre 14.0 MakeBoot, the object files could be bound without code (the /-c switch in the Binder), and MakeBoot would search the disk for the object file that contains the code. Since the Pilot runtime loader does not have this feature, it is necessary to bind object files with the code included.

21.2.2 New switches

A previously undocumented switch for MakeBoot:

/u Utility Pilot bootfile: the resulting bootfile is a Utility Pilot client.

A new switch for MakeBoot:

/c Code Links: use code links when possible when the object files are loaded. The default is **FALSE**, and frame

links are used. Frame links are preferable for modules that have global frames outside the **MDS**.

21.2.3 Parameter file

The parameter file can contain the following new entries:

GFT: *number*;

allow *number* of entries in the global frame table. **GFT** is the maximum number of modules that can be loaded with the resulting bootfile. This number include the MakeBoot loaded modules and the runtime loaded modules.

GFTBASE: *number*;

set the base of the global frame table at page *number*.

LOCALFRAMEPAGES: number;

sets the size of the the local frame heap to number pages. The default value for **LOCALFRAMEPAGES** is 50.

24 Debugger

See the XDE User's Guide - New chapter - Debugger portion of this change summary.

28 Performance tools

The performance tools have changed to use Sword interfaces instead of CoPilot interfaces. The performance tools may only be used with an outload or remote debugging session; "same world" performance tools are not available.

28.1.5 Getting started

The debugging session must be created before running the performance tool. If another outload or remote debugging session is started after the performance tool is run, it should be started with the -s switch (see the section on DebugUsefulDefs in the new XUG Debugger chapter). If the debugging session ends, the performance tool should be deactivated.