Appendix C. Internal Representation of Data Types

This appendix contains the detailed internal representations of the PDS standard data types listed in Table 3.2 of the Data Type Definitions chapter of this document.

Chapter Contents

Appendix C. Internal Representation of Data Types .............................................................C-1

C.1 MSB_INTEGER.......................................................................................................C-2
C.2 MSB_UNSIGNED_INTEGER.................................................................................C-4
C.3 LSB_INTEGER........................................................................................................C-6
C.4 LSB_UNSIGNED_INTEGER..................................................................................C-8
C.5 IEEE_REAL...........................................................................................................C-10
C.6 IEEE_COMPLEX ..................................................................................................C-13
C.7 PC_REAL ..............................................................................................................C-14
C.8 PC_COMPLEX......................................................................................................C-17
C.9 VAX_REAL, VAXG_REAL..................................................................................C-18
C.10 VAX_COMPLEX, VAXG_COMPLEX .................................................................C-22
C.11 MSB_BIT_STRING...............................................................................................C-23
C.12 LSB_BIT_STRING.............................................................................................C-25
C.1 MSB_INTEGER

Aliases: INTEGER
          MAC_INTEGER
          SUN_INTEGER

This section describes the signed integers stored in Most Significant Byte first (MSB) order. In this section the following definitions apply:

\[ b_0 \rightarrow b_3 \] Arrangement of bytes as they appear when read from a file (e.g., read \( b_0 \) first, then \( b_1 \), \( b_2 \), and \( b_3 \))

\[ i\text{-sign} \] Integer sign bit (bit 7 in the highest-order byte

\[ i_0 \rightarrow i_3 \] Arrangement of bytes in the integer, from lowest order to highest order.

The bits within each byte are interpreted from right to left (e.g., lowest value = bit 0, highest value = bit 7) in the following way:

4-byte integers:
- In \( i_0 \), bits 0-7 represent \( 2^{0} \) through \( 2^{7} \)
- In \( i_1 \), bits 0-7 represent \( 2^{8} \) through \( 2^{15} \)
- In \( i_2 \), bits 0-7 represent \( 2^{16} \) through \( 2^{23} \)
- In \( i_3 \), bits 0-6 represent \( 2^{24} \) through \( 2^{30} \)

2-byte integers:
- In \( i_0 \), bits 0-7 represent \( 2^{0} \) through \( 2^{7} \)
- In \( i_1 \), bits 0-6 represent \( 2^{8} \) through \( 2^{14} \)

1-byte integers:
- In \( i_0 \), bits 0-6 represent \( 2^{0} \) through \( 2^{6} \)

Negative integers are represented in two’s complement.

C.1.1 MSB 4-byte Integer

![Diagram of MSB 4-byte Integer]

\[ \begin{array}{cccc}
    i_3 & i_2 & i_1 & i_0 \\
    76543210 & 76543210 & 76543210 & 76543210 \\
    b_0 & b_1 & b_2 & b_3
\end{array} \]
C.1.2 MSB 2-byte Integer

C.1.3 MSB 1-byte Integer
C.2 MSB_UNSIGNED_INTEGER

Aliases: UNSIGNED_INTEGER
        MAC_UNSIGNED_INTEGER
        SUN_UNSIGNED_INTEGER

This section describes unsigned integers stored in Most Significant Byte first (MSB) format. In this section the following definitions apply:

\[ b_0 - b_3 \]
Arrangement of bytes as they appear when read from a file (e.g., read \( b_0 \) first, then \( b_1, b_2 \) and \( b_3 \))

\[ i_0 - i_3 \]
Arrangement of bytes in the integer, from lowest order to highest order. The bits within each byte are interpreted from right to left (e.g., lowest value = bit 0, highest value = bit 7), in the following way:

4-bytes:
- In \( i_0 \), bits 0-7 represent \( 2^{0} \) through \( 2^{7} \)
- In \( i_1 \), bits 0-7 represent \( 2^{8} \) through \( 2^{15} \)
- In \( i_2 \), bits 0-7 represent \( 2^{16} \) through \( 2^{23} \)
- In \( i_3 \), bits 0-7 represent \( 2^{24} \) through \( 2^{31} \)

2-bytes:
- In \( i_0 \), bits 0-7 represent \( 2^{0} \) through \( 2^{7} \)
- In \( i_1 \), bits 0-7 represent \( 2^{8} \) through \( 2^{15} \)

1-byte:
- In \( i_0 \), bits 0-7 represent \( 2^{0} \) through \( 2^{7} \)

C.2.1 MSB 4-byte Unsigned Integers

<table>
<thead>
<tr>
<th>( i_3 )</th>
<th>( i_2 )</th>
<th>( i_1 )</th>
<th>( i_0 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>76543210</td>
<td>76543210</td>
<td>76543210</td>
<td>76543210</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>( b_0 )</th>
<th>( b_1 )</th>
<th>( b_2 )</th>
<th>( b_3 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>76543210</td>
<td>76543210</td>
<td>76543210</td>
<td>76543210</td>
</tr>
</tbody>
</table>
C.2.2 MSB 2-byte Unsigned Integers

\[
\begin{array}{cc}
i_1 & i_0 \\
76543210 & 76543210 \\
b_0 & b_1
\end{array}
\]

C.2.3 MSB 1-byte Unsigned Integers

\[
\begin{array}{c}
i_0 \\
76543210 \\
b_0
\end{array}
\]
C.3 LSB_INTEGER

Aliases: PC_INTEGER
        VAX_INTEGER

This section describes signed integers stored in Least Significant Byte first (LSB) order. In this section the following definitions apply:

\[ b_0 - b_3 \] Arrangement of bytes as they appear when reading a file (e.g., read byte b0 first, then b1, b2 and b3)

\[ i\text{-sign} \] Integer sign bit – bit 7 in the highest order byte

\[ i_0 - i_3 \] Arrangement of bytes in the integer, from lowest order to highest order. The bits within each byte are interpreted from right to left (e.g., lowest value = bit 0, highest value = bit 7), in the following way:

4-bytes:
In \( i_0 \), bits 0-7 represent \( 2^{0} \) through \( 2^{7} \)
In \( i_1 \), bits 0-7 represent \( 2^{8} \) through \( 2^{15} \)
In \( i_2 \), bits 0-7 represent \( 2^{16} \) through \( 2^{23} \)
In \( i_3 \), bits 0-6 represent \( 2^{24} \) through \( 2^{30} \)

2-bytes:
In \( i_0 \), bits 0-7 represent \( 2^{0} \) through \( 2^{7} \)
In \( i_1 \), bits 0-6 represent \( 2^{8} \) through \( 2^{14} \)

1-byte:
In \( i_0 \), bits 0-6 represent \( 2^{0} \) through \( 2^{6} \)

All negative values are represented in two’s complement.

C.3.1 LSB 4-byte Integers

\[
\begin{array}{cccc}
\text{i-sign} & \text{i0} & \text{i1} & \text{i2} \\
76543210 & 76543210 & 76543210 & 76543210 \\
\text{b0} & \text{b1} & \text{b2} & \text{b3}
\end{array}
\]
C.3.2 LSB 2-byte Integers

\[\text{i-sign} \quad \downarrow \quad \text{i0} \quad \downarrow \quad \text{i1} \]

\[
\begin{array}{c|c}
\text{76543210} & \text{76543210} \\
\hline
\text{b0} & \text{b1}
\end{array}
\]

C.3.3 LSB 1-byte Integers

\[\text{i-sign} \quad \downarrow \quad \text{i0} \]

\[
\begin{array}{c}
\text{76543210} \\
\hline
\text{b0}
\end{array}
\]
C.4  **LSB_UNSIGNED_INTEGER**

**Aliases:** PC_UNSIGNED_INTEGER  
VAX_UNSIGNED_INTEGER

This section describes unsigned integers stored in Least Significant Byte first (LSB) format. In this section the following definitions apply:

\[ b_0 – b_3 \] Arrangement of bytes as they appear when reading a file (e.g., read byte b0 first, then b1, b2 and b3)

\[ i_0 – i_3 \] Arrangement of bytes in the integer, from lowest order to highest order. The bits within each byte are interpreted from right to left (e.g., lowest value = bit 0, highest value = bit 7), in the following way:

**4-bytes:**
- In \( i_0 \), bits 0-7 represent \( 2^{**0} \) through \( 2^{**7} \)
- In \( i_1 \), bits 0-7 represent \( 2^{**8} \) through \( 2^{**15} \)
- In \( i_2 \), bits 0-7 represent \( 2^{**16} \) through \( 2^{**23} \)
- In \( i_3 \), bits 0-7 represent \( 2^{**24} \) through \( 2^{**31} \)

**2-bytes:**
- In \( i_0 \), bits 0-7 represent \( 2^{**0} \) through \( 2^{**7} \)
- In \( i_1 \), bits 0-7 represent \( 2^{**8} \) through \( 2^{**15} \)

**1-byte:**
- In \( i_0 \), bits 0-7 represent \( 2^{**0} \) through \( 2^{**7} \)

### C.4.1  **LSB 4-byte Unsigned Integers**

<table>
<thead>
<tr>
<th>( i_0 )</th>
<th>( i_1 )</th>
<th>( i_2 )</th>
<th>( i_3 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>76543210</td>
<td>76543210</td>
<td>76543210</td>
<td>76543210</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>( b_0 )</th>
<th>( b_1 )</th>
<th>( b_2 )</th>
<th>( b_3 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>76543210</td>
<td>76543210</td>
<td>76543210</td>
<td>76543210</td>
</tr>
</tbody>
</table>
C.4.2 LSB 2-byte Unsigned Integers

<table>
<thead>
<tr>
<th>i0</th>
<th>i1</th>
</tr>
</thead>
<tbody>
<tr>
<td>76543210</td>
<td>76543210</td>
</tr>
<tr>
<td>b0</td>
<td>b1</td>
</tr>
</tbody>
</table>

C.4.3 LSB 1-byte Unsigned Integers

<table>
<thead>
<tr>
<th>i0</th>
</tr>
</thead>
<tbody>
<tr>
<td>76543210</td>
</tr>
<tr>
<td>b0</td>
</tr>
</tbody>
</table>
C.5 IEEE_REAL

Aliases: FLOAT
REAL
MAC_REAL
SUN_REAL

This section describes the internal format of IEEE-format floating-point numbers. In this section the following definitions apply:

\( b0 – b9 \) Arrangement of bytes as they appear when read from a file (e.g., read \( b0 \) first, then \( b1, b2, b3 \), etc.)

\( m \)-sign Mantissa sign bit

\( int \)-bit In 10-byte real format only, the integer part of the mantissa, assumed to be “1” in other formats, is explicitly indicated by this bit

\( e0 – e1 \) Arrangement of the portions of the bytes that make up the exponent, from lowest order to highest order. The bits within each byte are interpreted from right to left (e.g., lowest value = rightmost bit in the exponent part of the byte, highest value = leftmost bit in the exponent part of the byte) in the following way:

10-bytes (temporary):
- In \( e0 \), bits 0-7 represent \( 2^{0} \) through \( 2^{7} \)
- In \( e1 \), bits 0-6 represent \( 2^{8} \) through \( 2^{14} \)

Exponent bias = 16383

8-bytes (double precision):
- In \( e0 \), bits 4-7 represent \( 2^{0} \) through \( 2^{3} \)
- In \( e1 \), bits 0-6 represent \( 2^{4} \) through \( 2^{10} \)

Exponent bias = 1023

4-bytes (single precision):
- In \( e0 \), bit 7 represent \( 2^{0} \)
- In \( e1 \), bits 0-6 represent \( 2^{1} \) through \( 2^{7} \)

Exponent bias = 127
Arrangement of the portions of the bytes that make up the mantissa, from highest order fractions to the lowest order fraction. The order of the bits within each byte progresses from left to right, with each bit representing a fractional power of two, in the following way:

10-bytes (temporary):
- In m0, bits 6-0 represent $1/2^{**1}$ through $1/2^{**7}$
- In m1, bits 7-0 represent $1/2^{**8}$ through $1/2^{**15}$
- In m2, bits 7-0 represent $1/2^{**16}$ through $1/2^{**23}$
- In m3, bits 7-0 represent $1/2^{**24}$ through $1/2^{**31}$
- In m4, bits 7-0 represent $1/2^{**32}$ through $1/2^{**39}$
- In m5, bits 7-0 represent $1/2^{**40}$ through $1/2^{**47}$
- In m6, bits 7-0 represent $1/2^{**48}$ through $1/2^{**55}$
- In m7, bits 7-0 represent $1/2^{**56}$ through $1/2^{**63}$

8-bytes (double precision):
- In m0, bits 3-0 represent $1/2^{**1}$ through $1/2^{**4}$
- In m1, bits 7-0 represent $1/2^{**5}$ through $1/2^{**12}$
- In m2, bits 7-0 represent $1/2^{**13}$ through $1/2^{**20}$
- In m3, bits 7-0 represent $1/2^{**21}$ through $1/2^{**28}$
- In m4, bits 7-0 represent $1/2^{**29}$ through $1/2^{**36}$
- In m5, bits 7-0 represent $1/2^{**37}$ through $1/2^{**44}$
- In m6, bits 7-0 represent $1/2^{**45}$ through $1/2^{**52}$

4-bytes (single precision):
- In m0, bits 6-0 represent $1/2^{**1}$ through $1/2^{**7}$
- In m1, bits 7-0 represent $1/2^{**8}$ through $1/2^{**15}$
- In m2, bits 7-0 represent $1/2^{**16}$ through $1/2^{**23}$

The following representations all follow this format:

\[
mantissa \times 2^{(exponent – bias)}
\]

Note that the integer part ("1.") is implicit in all formats except the 10-byte (temporary) real format, as described above. In all cases the exponent is stored as an unsigned, biased integer (that is, the stored exponent value – bias value = true exponent).
C.5.1  IEEE 10-byte (Temporary) Real Numbers

```
m-sign  int-bit (always 1)
\downarrow  e1   e0  \downarrow  m0   m1   m2
76543210 76543210 76543210 76543210 76543210
b0   b1   b2   b3   b4

m3  m4  m5  m6  m7
76543210 76543210 76543210 76543210 76543210
b5   b6   b7   b8   b9
```
C.6 IEEE_COMPLEX

Aliases: COMPLEX
       MAC_COMPLEX
       SUN_COMPLEX

IEEE complex numbers consist of two IEEE_REAL format numbers of the same precision, contiguous in memory. The first number represents the real part and the second the imaginary part of the complex value.

For more information on using IEEE_REAL formats, see Section C.5.
C.7 PC_REAL

Aliases: None

This section describes the internal storage format corresponding to the PC_REAL data type. In this section the following definitions apply:

- **b0 – b9**: Arrangement of bytes as they appear when read from a file (e.g., read b0 first, then b1, b2 and b3)
- **m-sign**: Mantissa sign bit
- **int-bit**: In 10-byte real format only, the integer part of the mantissa, assumed to be “1” in other formats, is explicitly indicated by this bit.
- **e0 – e1**: Arrangement of the portions of the bytes that make up the exponent, from lowest order to highest order. The bits within each byte are interpreted from right to left (e.g., lowest value = rightmost bit in the exponent part of the byte, highest value = leftmost bit in the exponent part of the byte) in the following way:

  10-bytes (temporary):
  - In e0, bits 0-7 represent $2^{*}0$ through $2^{*}7$
  - In e1, bits 0-6 represent $2^{*}8$ through $2^{*}14$
  
  Exponent bias = 16383

  8-bytes (double precision):
  - In e0, bits 4-7 represent $2^{*}0$ through $2^{*}3$
  - In e1, bits 0-6 represent $2^{*}4$ through $2^{*}10$
  
  Exponent bias = 1023

  4-bytes (single precision):
  - In e0, bit 7 represent $2^{*}0$
  - In e1, bits 0-6 represent $2^{*}1$ through $2^{*}7$
  
  Exponent bias = 127

- **m0 – m7**: Arrangement of the portions of the bytes that make up the mantissa, from highest order fractions to the lowest order fraction. The order of the bits within each byte progresses from left to right, with each bit representing a fractional power of two, in the following way:

  10-bytes (temporary):
In m0, bits 6-0 represent $1/2^{2}$ through $1/2^{7}$
In m1, bits 7-0 represent $1/2^{8}$ through $1/2^{15}$
In m2, bits 7-0 represent $1/2^{16}$ through $1/2^{23}$
In m3, bits 7-0 represent $1/2^{24}$ through $1/2^{31}$
In m4, bits 7-0 represent $1/2^{32}$ through $1/2^{39}$
In m5, bits 7-0 represent $1/2^{40}$ through $1/2^{47}$
In m6, bits 7-0 represent $1/2^{48}$ through $1/2^{55}$
In m7, bits 7-0 represent $1/2^{56}$ through $1/2^{63}$

8-bytes (double precision):
In m0, bits 3-0 represent $1/2^{1}$ through $1/2^{4}$
In m1, bits 7-0 represent $1/2^{5}$ through $1/2^{12}$
In m2, bits 7-0 represent $1/2^{13}$ through $1/2^{20}$
In m3, bits 7-0 represent $1/2^{21}$ through $1/2^{28}$
In m4, bits 7-0 represent $1/2^{29}$ through $1/2^{36}$
In m5, bits 7-0 represent $1/2^{37}$ through $1/2^{44}$
In m6, bits 7-0 represent $1/2^{45}$ through $1/2^{52}$

4-bytes (single precision):
In m0, bits 6-0 represent $1/2^{1}$ through $1/2^{7}$
In m1, bits 7-0 represent $1/2^{8}$ through $1/2^{15}$
In m2, bits 7-0 represent $1/2^{16}$ through $1/2^{23}$

The following representations all follow this format:

$$1.\text{mantissa} \times 2^{(\text{exponent} - \text{bias})}$$

Note that the integer part (“1.”) is implicit in all formats except the 10-byte (temporary) real format, as described above. In all cases the exponent is stored as an unsigned, biased integer (that is, the stored exponent value – bias value = true exponent).
C.7.1  PC 10-byte (Temporary) Real Numbers

m7  m6  m5  m4  m3
---  ---  ---  ---  ---
76543210 76543210 76543210 76543210 76543210

b0  b1  b2  b3  b4

m2  m1  m0  e0  e1
---  ---  ---  ---  ---
76543210 76543210 76543210 76543210 76543210

b5  b6  b7  b8  b9

C.7.2  PC 8-byte (Double Precision) Real Numbers

m6  m5  m4  m3  m2  m1  e0  m0  e1
---  ---  ---  ---  ---  ---  ---  ---  ---
76543210 76543210 76543210 76543210 76543210 76543210 76543210 76543210 76543210

b0  b1  b2  b3  b4  b5  b6  b7

C.7.3  PC 4-byte (Single Precision) Real Numbers

m2  m1  m0  e1
---  ---  ---  ---
76543210 76543210 76543210 76543210

b0  b1  b2  b3
C.8 PC_COMPLEX

Aliases: None

PC complex numbers consist of two PC_REAL format numbers of the same precision, contiguous in memory. The first number represents the real part and the second the imaginary part of the complex value.

For more information on using PC_REAL formats, see Section C.7.
C.9 VAX_REAL, VAXG_REAL

Aliases: VAX_DOUBLE for VAX_REAL only.
No aliases for VAXG_REAL

This section describes the internal format corresponding to the VAX_REAL and VAXG_REAL data types. In this section the following definitions apply:

\[
b0 – b15 \quad \text{Arrangement of bytes as they appear when read from a file (e.g., read b0 first, then b1, b2 and b3)}
\]

\[
m-sign \quad \text{Mantissa sign bit}
\]

\[
e0 – e1 \quad \text{Arrangement of the portions of the bytes that make up the exponent, from lowest order to highest order. The bits within each byte are interpreted from right to left (e.g., lowest value = rightmost bit in the exponent part of the byte, highest value = leftmost bit in the exponent part of the byte) in the following way:}
\]

16-bytes (H-type, quad precision):
- In e0, bits 0-7 represent \(2^0\) through \(2^7\)
- In e1, bits 0-6 represent \(2^8\) through \(2^{14}\)

Exponent bias = 16385

8-bytes (G-type, double precision):
- In e0, bits 4-7 represent \(2^0\) through \(2^3\)
- In e1, bits 0-6 represent \(2^4\) through \(2^{10}\)

Exponent bias = 1025

8-bytes (D-type, double precision):
- In e0, bit 7 represents \(2^0\)
- In e1, bits 0-6 represent \(2^1\) through \(2^7\)

Exponent bias = 129

4-bytes (F-type, single precision):
- In e0, bit 7 represent \(2^0\)
- In e1, bits 0-6 represent \(2^1\) through \(2^7\)

Exponent bias = 129
Arrangement of the portions of the bytes that make up the mantissa, from highest order fractions to the lowest order fraction. The order of the bits within each byte progresses from left to right, with each bit representing a fractional power of two, in the following way:

16-bytes (H-type, quad precision):
- In m0, bits 7-0 represent $1/2^{**1}$ through $1/2^{**8}$
- In m1, bits 7-0 represent $1/2^{**9}$ through $1/2^{**16}$
- In m2, bits 7-0 represent $1/2^{**17}$ through $1/2^{**24}$
- In m3, bits 7-0 represent $1/2^{**25}$ through $1/2^{**32}$
- In m4, bits 7-0 represent $1/2^{**33}$ through $1/2^{**40}$
- In m5, bits 7-0 represent $1/2^{**41}$ through $1/2^{**48}$
- In m6, bits 7-0 represent $1/2^{**49}$ through $1/2^{**56}$
- In m7, bits 7-0 represent $1/2^{**57}$ through $1/2^{**64}$
- In m8, bits 7-0 represent $1/2^{**65}$ through $1/2^{**72}$
- In m9, bits 7-0 represent $1/2^{**73}$ through $1/2^{**80}$
- In m10, bits 7-0 represent $1/2^{**81}$ through $1/2^{**88}$
- In m11, bits 7-0 represent $1/2^{**89}$ through $1/2^{**96}$
- In m12, bits 7-0 represent $1/2^{**97}$ through $1/2^{**104}$
- In m13, bits 7-0 represent $1/2^{**105}$ through $1/2^{**112}$

8-bytes (G-type, double precision):
- In m0, bits 3-0 represent $1/2^{**1}$ through $1/2^{**4}$
- In m1, bits 7-0 represent $1/2^{**5}$ through $1/2^{**12}$
- In m2, bits 7-0 represent $1/2^{**13}$ through $1/2^{**20}$
- In m3, bits 7-0 represent $1/2^{**21}$ through $1/2^{**28}$
- In m4, bits 7-0 represent $1/2^{**29}$ through $1/2^{**36}$
- In m5, bits 7-0 represent $1/2^{**37}$ through $1/2^{**44}$
- In m6, bits 7-0 represent $1/2^{**45}$ through $1/2^{**52}$

8-bytes (D-type, double precision):
- In m0, bits 6-0 represent $1/2^{**1}$ through $1/2^{**7}$
- In m1, bits 7-0 represent $1/2^{**8}$ through $1/2^{**15}$
- In m2, bits 7-0 represent $1/2^{**16}$ through $1/2^{**23}$
- In m3, bits 7-0 represent $1/2^{**24}$ through $1/2^{**31}$
- In m4, bits 7-0 represent $1/2^{**32}$ through $1/2^{**39}$
- In m5, bits 7-0 represent $1/2^{**40}$ through $1/2^{**47}$
- In m6, bits 7-0 represent $1/2^{**48}$ through $1/2^{**55}$

4-bytes (F-type, single precision):
- In m0, bits 6-0 represent $1/2^{**1}$ through $1/2^{**7}$
- In m1, bits 7-0 represent $1/2^{**8}$ through $1/2^{**15}$
- In m2, bits 7-0 represent $1/2^{**16}$ through $1/2^{**23}$
The following representations all follow this format:

\[ \text{mantissa} \times 2^{(\text{exponent} - \text{bias})} \]

Note that the integer part ("1.") is implicit in all formats except the 10-byte (temporary) real format, as described above. In all cases the exponent is stored as an unsigned, biased integer (that is, the stored exponent value – bias value = true exponent).

C.9.1 VAX 16-byte H-type (Quad Precision) Real Numbers

![VAX 16-byte H-type Real Number Format]

C.9.2 VAX 8-byte G-type (Double Precision) Real Numbers

![VAX 8-byte G-type Real Number Format]
C.9.3 VAX 8-byte D-type (Double Precision) Real Numbers

```
  e0-bit  m-sign
    ↓  m0  ↓  e1  m2  m1  m4  m3  m6  m5
  76543210 76543210 76543210 76543210 76543210 76543210 76543210 76543210
  b0  b1  b2  b3  b4  b5  b6  b7
```

C.9.4 VAX 4-byte F-type (Single Precision) Real Numbers

```
  e0-bit  m-sign
    ↓  m0  ↓  e1  m2  m1
  76543210 76543210 76543210 76543210
  b0  b1  b2  b3
```
C.10 VAX_COMPLEX, VAXG_COMPLEX

Aliases: None

VAX complex numbers consist of two VAX_REAL (or VAXG_REAL) format numbers of the same precision, contiguous in memory. The first number represents the real part and the second the imaginary part of the complex value.

For more information on using VAX_REAL formats, see Section C.9.
C.11 MSB_BIT_STRING

Aliases: None

This section describes the storage format for bit strings stored in Most Significant Byte first (MSB) format. In this section the following definitions apply:

\[ b_0 \rightarrow b_3 \]  
Arrangement of bytes as they appear when read from a file (e.g., read \( b_0 \) first, then \( b_1 \), \( b_2 \) and \( b_3 \))

The bits within a byte are numbered from left to right, as shown below:

```
76543210
|    |    |    |    |
|    | 1  |    | 8  |
```

Note that in the case of MSB bit strings, no byte-swapping is required. That is, the physical storage order of the bytes is identical to the logical order.

C.11.1 MSB \( n \)-byte Bit Strings

```
<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1-8</td>
<td>9-16</td>
<td>17-24</td>
<td>25-32</td>
</tr>
<tr>
<td>76543210</td>
<td>76543210</td>
<td>76543210</td>
<td>76543210</td>
</tr>
<tr>
<td>b0</td>
<td>b1</td>
<td>b2</td>
<td>b3</td>
</tr>
</tbody>
</table>
```

\[ (nx8) - (nx8) \]

The bits within a byte are numbered from left to right, as shown above.

C.11.2 MSB 2-byte Bit String

```
<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1-8</td>
<td>9-16</td>
</tr>
<tr>
<td>76543210</td>
<td>76543210</td>
</tr>
<tr>
<td>b0</td>
<td>b1</td>
</tr>
</tbody>
</table>
```
C.11.3 MSB 1-byte Bit String

```
bits 1-8

76543210

b0
```
C.12 LSB_BIT_STRING

Aliases: VAX_BIT_STRING

This section describes the structure of bit strings stored in Least Significant Byte first (LSB) order. In this section, the following definitions apply:

\[ b_0 – b_3 \] Arrangement of bytes as they appear when read from a file (e.g., read \( b_0 \) first, then \( b_1, b_2 \) and \( b_3 \))

The bits within a byte are numbered from left to right, as shown below:

```
    76543210
   ↑   ↑
 bit 1 bit 8
```

Note that for LSB bit strings byte-swapping is required to convert the storage order of bytes to the logical order.

C.12.1 LSB 4-byte Bit String

Physical order (as read from the file):

```
<table>
<thead>
<tr>
<th>bits</th>
<th>bits</th>
<th>bits</th>
<th>bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>25-32</td>
<td>17-24</td>
<td>9-16</td>
<td>1-8</td>
</tr>
<tr>
<td>76543210</td>
<td>76543210</td>
<td>76543210</td>
<td>76543210</td>
</tr>
</tbody>
</table>
```

<table>
<thead>
<tr>
<th>b0</th>
<th>b1</th>
<th>b2</th>
<th>b3</th>
</tr>
</thead>
</table>

Logical order (after byte-swapping):

```
<table>
<thead>
<tr>
<th>bits</th>
<th>bits</th>
<th>bits</th>
<th>bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-8</td>
<td>9-16</td>
<td>17-24</td>
<td>25-32</td>
</tr>
<tr>
<td>76543210</td>
<td>76543210</td>
<td>76543210</td>
<td>76543210</td>
</tr>
</tbody>
</table>
```

<table>
<thead>
<tr>
<th>b3</th>
<th>b2</th>
<th>b1</th>
<th>b0</th>
</tr>
</thead>
</table>
C.12.2 LSB 2-byte Bit String

Physical order (as read from the file):

<table>
<thead>
<tr>
<th>bits</th>
<th>bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>9-16</td>
<td>1-8</td>
</tr>
<tr>
<td>76543210</td>
<td>76543210</td>
</tr>
</tbody>
</table>

Logical order (after byte-swapping):

<table>
<thead>
<tr>
<th>bits</th>
<th>bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-8</td>
<td>9-16</td>
</tr>
<tr>
<td>76543210</td>
<td>76543210</td>
</tr>
</tbody>
</table>

C.12.3 LSB 1-byte Bit String

<table>
<thead>
<tr>
<th>bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-8</td>
</tr>
<tr>
<td>76543210</td>
</tr>
</tbody>
</table>

Note that in this degenerate case no byte-swapping is required.
binary storage formats, C-1

, C-1
data representation
  internal, C-1
IEEE_COMPLEX
  storage format, C-13
IEEE_REAL
  storage format, C-10
least significant byte first (LSB) storage format, C-6, C-8, C-25
LSB_BIT_STRING
  storage format, C-25
LSB_INTEGER
  storage format, C-6
LSB_UNSIGNED_INTEGER
  storage format, C-8
most significant byte first (MSB) storage format, C-2, C-4, C-23
MSB_BIT_STRING
  storage format, C-23
MSB_INTEGER
  storage format, C-2
MSB_UNSIGNED_INTEGER
  storage format, C-4
PC_COMPLEX
  storage format, C-17
PC_REAL
  storage format, C-14
, C-1

, C-1

, C-1

, C-1

, C-1

, C-1
VAX_COMPLEX
  storage format, C-22
VAX_REAL
storage format, C-18
VAXG_COMPLEX
  storage format, C-22
VAXG_REAL
  storage format, C-18