Technical Report

CMU/SEI-87-TR-019
ESD-TR-87-170

The Use of Representation Clauses and Implementation-Dependent Features in Ada:

IVA. Qualitative Results for Ada/M(44)
Version 1.6

B. Craig Meyers
Andrea L. Cappellini

July 1987

Technical Report

CMU/SEI-87-TR-19
ESD/TR-87-170
July 1987

The Use of Representation Clauses and Implementation-Dependent Features in Ada: IVA. Qualitative Results for Ada/M(44)

Version 1.6

B. Craig Meyers
Andrea L. Cappellini

Ada Embedded Systems Testbed Project

Approved for public release.
Distribution unlimited.

JPO approval signature on file

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213

This report was prepared for the SEI Joint Program Office HQ ESC/AXS

5 Eglin Street

Hanscom AFB, MA 01731-2116

The ideas and findings in this report should not be construed as an official DoD position. It is published in the interest of scientific and technical information exchange.

FOR THE COMMANDER

(signature on file)

Thomas R. Miller, Lt Col, USAF, SEI Joint Program Office

This work is sponsored by the U.S. Department of Defense.

This work was created in the performance of Federal Government Contract Number F19628-95-C-0003 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center. The Government of the United States has a royalty-free government-purpose license to use, duplicate, or disclose the work, in whole or in part and in any manner, and to have or permit others to do so, for government purposes pursuant to the copyright license under the clause at 52.227-7013.

This document is available through Research Access, Inc. / 800 Vinial Street / Pittsburgh, PA 15212. Phone: 1-800-685-6510. FAX: (412) 321-2994. RAI also maintains a World Wide Web home page at http://www.rai.com

Copies of this document are available through the National Technical Information Service (NTIS). For information on ordering, please contact NTIS directly: National Technical Information Service / U.S. Department of Commerce / Springfield, VA 22161. Phone: (703) 487-4600.

This document is also available through the Defense Technical Information Center (DTIC). DTIC provides acess to and transfer of scientific and technical information for DoD personnel, DoD contractors and potential con tractors, and other U.S. Government agency personnel and their contractors. To obtain a copy, please contact DTIC directly: Defense Technical Information Center / 8725 John J. Kingman Road / Suite 0944 / Ft. Belvoir, VA 22060-6218. Phone: 1-800-225-3842 or 703-767-8222.

The Use of Representation Clauses and Implementation-Dependent Features in Ada: IVA. Qualitative Results for Ada/M(44) Version 1.6

Abstract: This report, one in a series, provides a qualitative assessment of the support of representation clauses and implementation-dependent features in Ada provided by the Ada/M(44) compiler, Version 1.6. The evaluation questions that were presented in a previous report of this series form the basis of the qualitative assessment. A subjective evaluation of the support provided for these features is also presented.

1. Introduction

The Ada language was developed as a general purpose language applicable to the development and maintenance of mission-critical systems for the Department of Defense (DoD). In the development of the language, a need to allow the language to interact with the underlying machine architecture was recognized. This coupling is discussed in Chapter 13 of the *Reference Manual for the Ada Programming Language* [1].

A frequent characteristic of mission-critical systems is that they employ "packed" data structures. Furthermore, within such packed structures there may be nonstandard data representations. For example, an integer type may have a length of 12 bits, or some fixed-point type may have arbitrarily scaled precision.

Such packed data structures are defined in Ada through the use of representation clauses. The use of these clauses is highly machine dependent. That is, some compilers may provide only limited support, while others may provide a full set of capabilities. Since representation clauses are implementation dependent, their use may affect the portability of any code which uses these features.

This report is one of a series of reports related to the use of representation clauses and implementation-dependent features in Ada. The first report in the series, reference [2], provides an overview of the use of representation clauses and implementation-dependent features and contains a series of case study examples. A second report, reference [3], formulated a list of questions applicable to the evaluation of a particular compiler from the perspective of representation clauses and implementation-dependent features. This is followed by a discussion of experimental procedures and methodologies, in reference [4]. Finally, in reference [5], a qualitative assessment of the support provided for representation clauses and implementation-dependent features for the VAX Ada compiler was performed.

The purpose of this report is to provide a qualitative assessment of the support of representation clauses and implementation-dependent features in Ada provided by the Ada/M(44) compiler. Thus, the questions raised in reference [3] are answered here; the emphasis is principally qualitative.

The decision to assess the Ada/M(44) compiler was motivated by the following reasons. First, the development of this compiler is funded by the Navy. This development is coupled to the Ada Language System (ALS) which was originally funded by the Army. Second, the Ada/M(44) targets to the AN/UYK-44, which is a Navy standard processor. Thus, the results reported here are expected to be of interest to applications that use the Ada/M(44) compiler.

This report is organized in the following manner: In Chapter 2 we present a discussion of the qualitative results applicable to the use of representation clauses and implementation-dependent features for the Ada/M(44) compiler. As in reference [5], a generalized subjective assessment based on the detailed results is also given. In Chapter 3, the relation between the results obtained and the examples presented in reference [2] is discussed. This discussion illustrates the use of qualitative results, such as reported here, for application-specific problems. A brief summary appears in Chapter 4, followed by a list of the applicable references. The actual results, which are answers to specific questions, are provided in Appendix I.

This report has been prepared by the Ada Embedded Systems Testbed Project at the Software Engineering Institute (SEI). The SEI is a federally funded research and development center (FFRDC) sponsored by the Department of Defense (DoD) and established and operated by Carnegie Mellon University. This report was prepared while the authors were on sabbatical leave at the SEI.

2. Discussion

A basic goal of reference [3] was to define a set of questions relevant to the assessment of a particular compiler from the perspective of representation clauses and implementation-dependent features. The questions appearing in reference [3] are of a qualitative nature, as well as quantitative. Reference [5] shows the application of that set of questions for the purpose of qualitatively assessing the support provided for representation clauses and implementation-dependent features for the VAX Ada compiler.

It is the purpose of this Section to provide a response to the questions posed in reference [3] for the Ada/M(44) compiler in the same manner as was presented in reference [5] for VAX Ada. As the emphasis here is toward a qualitative level of assessment, the reported results emphasize this aspect of evaluation. In other words, the focus in this report is on whether a particular aspect of representation clauses and implementation-dependent features is supported, as well as the amount of support. Questions which are principally quantitative in nature may be evaluated according to the methodology described in reference [4] and will not be addressed here. Thus, no attempt to evaluate compiler performance is made in this report. In spite of this, it is believed that dissemination of these results is warranted and are of interest to the application development community.

The responses to the questions have been obtained from consideration of the applicable documentation, as well as examination of selected generated code. The particular version of the Ada/M(44) compiler which was used for this assessment was Version 1.6. The questions and answers are presented in Appendix I. In the responses, we have attempted to cite the document that provided the required information. The particular documents are cited in the following manner:

1. PSEH = *Ada/M(44) Program Support Environment (PSE) Handbook* [6]
2. RTEH = *Ada/M(44) Run-Time Environment Handbook* [7]
3. DN = *Ada/M(44) Delivery Notes ADAM* [8]
4. TD = *AN/UYK-44 Technical Description* [9]

Some of the questions appearing in Appendix I can only be answered by detailed assessment, and such questions have been so indicated. The detailed assessment requires quantitative procedures such as those described in reference [4]. For our purposes, the term quantitative is used in a broad sense in that it refers to performance issues and measurements and also to detailed examination of generated code to extract information about the internals of the compiler.

Although the responses presented in Appendix I are cast in qualitative terms, it is quite evident that there is a considerable amount of information available. It is recognized that many users will desire an overview of the support provided for representation clauses and implementation-dependent features. Thus, in reference [5], a set of subjective criteria was discussed from which support of the major aspects of representation clauses and implementation-dependent features may be assessed. The set of subjective criteria is the following:

1. **Full Support**: The compiler provides support for the language feature, subject to natural limitations imposed on the hardware implementation.

2. **Support with Minor Limitations**: The compiler provides support for the language feature subject to only minor limitations. This implies, then, that the support provided should be satisfactory for many applications.
3. **Support with Major Limitations**: The compiler provides support for the language feature but there are major limitations. This should be understood to mean that use of the indicated language feature in many applications would be difficult.
4. **Unsupported**: The compiler provides no support for the language feature.

We stress that the criteria listed above are subjective in nature. There may be cases where a particular category is supported with only minor limitations. It may be, however, that the minor limitations could cause a serious problem for some applications. In spite of the caveat about the subjective nature of the judgments to be presented, nevertheless, we believe they have merit.

The results presented are basically organized in correspondence with the categories appearing in Appendix I. One category not present in Appendix I yet deemed sufficiently important enough to include in this subjective evaluation is that of support facilities. We are referring specifically to documentation and debugging facilities. Below, each category title is given, followed by the subjective rating. The rating is then followed by explanatory information where it is deemed especially relevant.

1. **Pragma OPTIMIZE**: Full Support.
2. **Data Types Supported**: Supported with Minor Limitations. The lack of support for floating-point types with precision greater than six (decimal) digits may cause problems for some applications.
3. **Pragma PACK**: Full Support.
4. **Length Clauses**: Supported with Minor Limitations. It is required that fixed-point types be represented according to a constant size. This could present problems for certain applications.
5. **Enumeration Representation Clauses**: Full Support.
6. **Record Representation Clauses**: Supported with Minor Limitations. Where a record contains a component of a fixed-point type, restrictions on the size of the component may be a problem due to the limitations imposed by the compiler.
7. **Address Clauses**: Supported with Major Limitations. Address clauses are not supported for objects.
8. **Data Conversion and Assignment**: Full Support.
9. **Representation Attributes**: Full Support. Ada/M(44) provides an additional attribute which allows the user to access physical addresses and may be helpful for some systems.
10. **Pragma INTERFACE**: Supported with Major Limitations. Only subprograms written in assembler may be specified.

11. **Support Facilities**: Supported with Major Limitations. No symbolic debugger is currently available for the AN/UYK-44 target. Additionally, the documentation does not provide as much information as a user would perhaps require. For example, in the Ada/M(44) references [6] through [9], there is no Appendix F. Appendix F lists the implementation-dependent characteristics. As another example, the *Reference Manual for the Ada Programming Language* states that whether or not a record component can overlap a storage boundary is implementation-defined. The Ada/M(44) references do not specify whether or not storage boundary overlaps are allowed.

From the preceding summary, it is evident that the support for representation clauses and implementation-dependent features by Ada/M(44) has considerable variation. In some cases, there is more support than required by the language, such as representation attributes. In other cases, notably address clauses, there is only minimal support provided or, as with length clauses, the support provided has restrictions as to details of implementations. The preceding illustrates two points. First, the results presented here serve to illustrate the compiler-dependent aspects of support provided for representation clauses and implementation-dependent features. A second point – and of possibly greater significance to application developers – is that the selection and use of a particular compiler must be made with extreme care.

It must be stressed that the results presented in this report are of a qualitative nature. As such, they describe the support for a particular aspect of representation clauses and implementation-dependent features. The issue of support for a language feature is clearly different from either the performance of the compiler or the assessment of the effect of use of the feature. Issues of the latter type are principally quantitative in nature.

3. Relation to Examples in Volume I

In the preceding chapter, we presented a synopsis of the support provided by the Ada/M(44) compiler for representation clauses and implementation-dependent features. The synopsis abstracted some general results from the detailed answers.

It may be well, however, to consider the manner in which a report such as this would be used by application developers. This is clearly an important issue and is now demonstrated. Thus, in the first volume of this series, reference [2], a number of case study examples were presented. Those examples were drawn from the mission-critical systems community and contain certain characteristics representative of problems typically encountered.

In the following, a brief statement of the problem for each example in reference [2] is presented. The implementation of the particular example is then considered, based on the Ada/M(44) compiler which is targeted for the AN/UYK-44. The ability to affect a solution to the stated problem is presented. The discussion is based on the results presented in Appendix I. That is, we provide specific references to the questions (and answers) deemed relevant for the example under consideration. It is believed that such a procedure illustrates the manner in which a report such as this may be used by application developers.

In the following paragraphs, references are made to the questions and answers appearing in Appendix I. Questions and answers are referenced by category letter followed by the number of the question. For example, "F.3" refers to question 3 under category F, which is Record Representation Clauses.

In Section 5.2 of the first report in this series, reference [2], an introductory example was given illustrating the use of representation clauses. The example was that of a message header that appears in every message used for communications between a shipboard Inertial Navigation System (INS) and some external computer (EC). This example illustrates the length, enumeration, and record clauses, with data of type integer and enumeration. The assumed value of SYSTEM.STORAGE_UNIT for the examples is equal to 8 and the default value for Ada/M(44) is 16. Thus, this implementation would be invalid for Ada/M(44). Though, as stated in J.4, pragma STORAGE_UNIT is supported and can be used to change Ada/M(44) SYSTEM.STORAGE_UNIT to 8 for these examples. With the inclusion of pragma STORAGE_UNIT, this example would be legal for Ada/M(44) since the requirements for this example are within the following restrictions:

1. The size specified in a length clause for an integer type must not exceed 16 bits (as stated in D.2).
2. The storage specified in a component clause must be enough to hold any value of the component type (as noted in F.3).
3. Record components are allowed to overlap storage unit boundaries (see F.4).

A note should be made that as stated in A.2, Ada/M(44) ordering of bits and storage units is right to left, and these examples use left to right ordering. Thus, when examining the actual contents of memory, the results will be different from that expected. If right to left ordering is required, say by some external system, then these examples will not be valid for Ada/M(44).

The second example, given in Section 5.3 of reference [2], is that of a Test Message which is used to test the communications interface between the INS and EC. This example contains a message header (implemented in the previous example) and integer test data. As in the last example, the requirements here are within the restrictions listed in F.3 that the storage specified in the component clause must be enough to hold any value of the component type, and in F.4 that storage unit boundary overlaps are allowed. Also, the implementation of this example, and some of the examples to follow, use a predefined type INTEGER which is assumed to be 32 bits. Ada/M(44) predefined type INTEGER is 16 bits, as seen in B.1. Note also in B.1 that there is another pre-defined type, LONG_INTEGER, which is 32 bits. Therefore, with the inclusion of pragma STORAGE_UNIT and replacing type INTEGER with LONG_INTEGER, this implementation would be valid for Ada/M(44).

The next example, in Section 5.4 of reference [2], is slightly more complicated than the previous examples in that it contains both fixed- and floating-point data. A navigation message is the example, and it is used to send data such as ownship latitude, longitude, and speed to an EC. This example contains a requirement that fixed-point data be represented in less than 32 bits. Ada/M(44) represents fixed-point types in 32 bits as discussed in B.2. Based on this and the restriction listed in F.3 that a component clause must specify enough storage to hold any value of the component type, this implementation is illegal for Ada/M(44). To implement this example using Ada/M(44), a conversion routine must be considered that converts the actual data field in the message to the default Ada/M(44) 32-bit fixed-point representation. This will be the representation from which calculations are performed.

The fourth example, Section 5.5 of reference [2], illustrates an implementation of analog conversion. In this example, ship heading, roll, and speed data are DMA mapped to particular addresses and must be converted into actual values. The requirement that data are DMA mapped was met by the use of address clauses. This example would not be valid on Ada/M(44) since, as stated in G.1, address clauses for objects are not supported.

In Section 5.6 of reference [2], an example of a message checksum was given. This function computes the checksum of the navigation message that was implemented in Section 5.4 of reference [2] and discussed above. Recall that the implementation of the navigation message was invalid for Ada/M(44). Thus, this checksum implementation would be invalid since it depends on that message implementation. A general-purpose checksum routine based on the use of the generic function UNCHECKED_CONVERSION and dealing with valid message implementations would be legal for Ada/M(44) since the following restrictions are met:

1. The size specified for an integer type must not exceed 16 bits (as stated in D.2).
2. Ada/M(44) supports UNCHECKED_CONVERSION (reported in H.3).
3. The size of the source and target objects used by an instantiation of UNCHECKED_CONVERSION must be equal (as stated in H.4).

Also, as previously noted, type INTEGER must be replaced with LONG_INTEGER.

The final example given in reference [2], Section 6.4.2, illustrates the use of pragma INTERFACE. The problem in this example is the need to allocate and access some data structure containing data

and status information for an INS gyro. As part of this problem, a conversion must be performed to obtain a 32-bit, fixed-point quantity that has 15 bits of precision from arbitrary fixed-point quantities. This conversion was performed by an assembler routine that is accessed via pragma INTERFACE with representation attributes providing the appropriate parameters to that routine. This example is compilable on Ada/M(44) with the following providing relevant information to verify this:

1. There are no restrictions on the use of 'ADDRESS (see I.1).
2. There are no restrictions on the use of 'FIRST_BIT (see I.4).
3. There are no restrictions on the use of 'LAST_BIT (see I.5).
4. Pragma INTERFACE is supported, though the language name is restricted to MACROM_NORMAL which is the AN/UYK-44 assembler (see J.5).

The preceding has illustrated the use of results presented in this report to assess problems that are representative of mission-critical systems. The purpose in presenting the above discussion, therefore, is to illustrate how reports such as this may be used. It is to be noted that the emphasis in the above discussion has been on application of qualitative results. That is, the emphasis is more toward obtaining a solution to a problem, as opposed to an assessment of how "well" the solution implements the problem requirement.

4. Summary

This report is one of a series dealing with the use of representation clauses and implementation-dependent features in Ada. This report provides a qualitative assessment of the Ada/M(44) compiler, Version Release 1.6. Subjective criteria were established to provide an overall assessment of the support provided by this compiler for representation clauses and implementation-dependent features. In general and based upon those subjective criteria, this compiler provides support with minor limitations for the implementation of representation clauses and implementation-dependent features. Some exceptions, however, have been noted.

This report may be used in conjunction with the results of a detailed experimental assessment of the Ada/M(44) compiler to determine its characteristics for use by specific applications.

References

1. *Reference Manual for the Ada Programming Language*, Department of Defense MIL-STD-1815, 1983.

2. B. Craig Meyers and Andrea L. Cappellini, *The Use of Representation Clauses and Implementation-Dependent Features in Ada. I: Overview*, CMU/SEI-87-TR-14, ESD-TR-87-115, July 1987.

3. B. Craig Meyers and Andrea L. Cappellini, *The Use of Representation Clauses and Implementation-Dependent Features in Ada. IIA: Evaluation Questions*, CMU/SEI-87-TR-15, ESD-TR-87-116, July 1987.

4. B. Craig Meyers and Andrea L. Cappellini, *The Use of Representation Clauses and Implementation-Dependent Features in Ada. IIB: Experimental Procedures*, CMU/SEI-87-TR-18, ESD-TR-87-126, July 1987.

5. B. Craig Meyers and Andrea L. Cappellini, *The Use of Representation Clauses and Implementation-Dependent Features in Ada. IIIA: Qualitative Results of the VAX Ada Compiler*, CMU/SEI-87-TR-17, ESD-TR-87-118, July, 1987.

6. Softech, Inc., *Ada/M(44) Program Support Environment (PSE) Handbook, Final Volume I*, 30 September 1986, Contract N00024-85-C-7037.

7. Softech, Inc., *Ada/M(44) Run-Time Environment (RTE) Handbook, Final*, 30 September 1986, Contract N00024-85-C-7037.

8. Softech, Inc., *Ada/M(44) Delivery Notes*, 3 October 1986, Contract N00024-85-C-7037.

9. Sperry Univac, *AN/UYK-44 Technical Description*, January 1984.

Appendix I: Questions Relevant to the Use of Representation Clauses and Implementation-Dependent Features

A. General:

1. **What is the basic unit of SYSTEM.STORAGE_UNIT? (This is useful when defining record layouts.)**

One word or 16 bits. [PSEH, page 7-7]

2. **What is the ordering of allocation for storage units? Is it left-to-right or right-to-left with respect to each other? How are bits numbered within storage units? Is it left-to-right or right-to-left? Does the numbering always begin with zero? (This is useful when defining record layouts and verifying the actual allocation of record layouts.)**

Bits within storage units are numbered 0 .. 15 starting at the right. [PSEH, page 7-11] The ordering of storage units with respect to each other is right to left.

3. **It is also appropriate to consider the role of the underlying architecture, particularly regarding data conversions from representation clauses to other formats. Does the machine include instructions for inserting and extracting bit-length fields? What are the restrictions on the use of such instructions (for example, what is the maximum field size to which an instruction may be applied)?**

In the current instruction set for AN/UYK-44, there are no instructions that support arbitrary bit length insertion and extraction. Samples of machine code generated for AN/UYK-44 showed that calls were generated to the runtime library to perform general purpose bit operations, e.g. bit move. There are, however, instructions to perform masked substitution of bit fields but these instructions are restricted to corresponding bit fields.

It is interesting to note that the instruction set also supports basic mathematical functions, e.g., square root, sine, arcsine, in hardware.

[TD]

4. **Is pragma OPTIMIZE supported? If so, are there any restrictions on its use?**

Yes. The default argument is SPACE and the OPTIMIZE option must be given to the compiler for this pragma to be in effect. [PSEH, page 7-3]

5. **The use of representation clauses may present unusual problems throughout design and coding. What facilities exist for verifying results when representation clauses are used? We are speaking here of the debugger; thus, are there restrictions on the use of the debugger when representation clauses are used?**

The DEBUG option to the compiler is not supported for Ada/M(44). Thus, currently there are no symbolic debugging facilities with Ada/M(44). [PSEH, page 9-4]

It should be noted that there are other facilities, namely MTASS, for simulation and/or debugging of object code for the AN/UYK-44.

6. **Does the compiler provide a load map that contains sufficient details to identify the location of quantities specified using representation clauses?**

Neither the Ada/M(44) compiler nor linker provides an option for a load map with physical location of program variables. [PSEH, page 9-3 and page 11-5]

7. **Are there any restrictions on representation clauses?**

There are no documented restrictions.

8. **Compiler implementors currently have the option as to what degree, if any, the features in Chapter 13 of the** *Reference Manual for the Ada Programming Language* **will be supported. It is conceivable that upgraded versions of an implementation will enhance the support originally available for such features as representation clauses. How is the documentation upgraded? Is it by release notes or page changes? The manner in which this is accomplished can affect the ease with which documentation can be used.**

The principal documents referenced by us, namely references [6] and [7], are final deliveries on the contract. We are not aware of change pages or release notes to the above documents. It is not known at this time the mechanism by which documentation will be upgraded. A set of delivery notes, reference [8], accompanied the documentation received and listed known problems in the software.

B. Data Types Supported:

1. **What are the basic implementations of integer types?**

Ada/M(44) offers the following two integer types:

- INTEGER with range -2**(15) .. 2**(15)-1
- LONG_INTEGER with range -2**(31) .. 2**(31)-1

[PSEH, page 7-6]

Type INTEGER is stored as a word, which is 16 bits. Type LONG_INTEGER is stored as a pair of words, where the most significant portion of the object is stored at the even word address and the least significant portion of the object is stored at the next odd address. [PSEH, page 8-2]

2. **What are the basic implementations of fixed-point types?**

Each fixed-point type is stored as a right justified integer within 32 bits and with an implicit scale factor. [PSEH, page 8-3]

3. **What are the basic implementations of floating-point types?**

Ada/M(44) offers one floating-point type:

FLOAT with precision of 6 digits in the range
-7.237005E75 .. 7.237005E75.

[PSEH, page 7-6]

Type FLOAT is stored as two longword-aligned addresses. Longword-aligned means that the least significant bit of the first word of the object must be bit 0 of an even memory address. [PSEH, page 8-3]

4. ***Does the compiler provide predefined unsigned data types? If not, is it permissible for a user to define these types? For example, is the following legal:***

```
type Unsigned_Small_Int is range 0 .. 7;
for Unsigned_Small_Int'SIZE use 3;
```

Ada/M(44) does not provide any unsigned data types though the example above compiles without error. Thus, it is permissible to define unsigned integer types that are less than or equal to 16 bits since the largest value allowed for the size specified in a length clause for an integer is 16. (See D.2).

C. Pragma PACK:

1. **Does the compiler support the use of pragma PACK?**

Yes. [PSEH, page 7-3]

2. **What restrictions are placed on the use of pragma PACK? For example, are there certain types that may or may not be packed?**

There are no documented restrictions on the use of this pragma. [PSEH, page 7-3]

When pragma PACK is in effect, the compiler does not override the default size of an integer or enumeration type, unless a length clause is given. If a length clause is given, the smaller size of either the default size or the size specified in the clause is used. [PSEH, page 8-2]

D. Length Clauses:

1. **Does the compiler support the use of length clauses? What are the restrictions on their use?**

Yes. [PSEH, page 7-8] There are no documented restrictions.

2. **Are there restrictions on the use of the SIZE attribute designator in a length clause?**

The following restrictions exist:

- *For integer types* specified with range L .. R the size, n, specified must be such that:
  - $2 \leq n \leq 16$ where $R \leq 2^{n-1}-1$ and $L \geq -2^{n-1}$
  - $1 < n \leq 15$ where $R \leq 2^{n-1}$ and $L \geq 0$

  By default, a size 16 is used when $R \leq 2^{15}-1$ and $L \geq -2^{15}$; otherwise, a size of 32 is used.
- *For fixed-point types*, the size specified can only be 32.
- *For floating-point types*, the size specified can only be 32.
- *For enumeration types*, the size, n, specified must be such that:
  - $2 \leq n \leq 16$ where 'FIRST and 'LAST both fall within the range -2^{15} .. $2^{15}-1$
  - $1 < n \leq 15$ where 'FIRST ≥ 0 and 'LAST $\leq 2^{('SIZE)}-1$

  The default size is 16.
- *For arrays and records*, the size specified must be less than or equal to $2^{31}-1$. [PSEH, 7-11]

[PSEH, page 7-8]

3. **Are there restrictions on the use of the STORAGE_SIZE attribute designator in a length clause?**

There are no documented restrictions.

4. **Are there restrictions on the use of the SMALL attribute designator in a length clause?**

There are no documented restrictions.

5. **When using a SIZE attribute designator in a length clause the** *Reference Manual for the Ada Programming Language* **states that the value of the expression specifies an upper bound for the number of bits to be allocated. The presence of a range constraint or the use of a**

predefined type implicitly defines the maximum number of bits required to allocate objects. If extra bits are specified in the length clause, are these extra bits allocated by the compiler?

With the available documentation and the absence of a facility that provides a load map, we were unable to resolve the issue raised above. Due to this fact, detailed experimentation is required for resolution of this issue.

6. **Suppose a type, with associated length clause, has been specified storage where the number of bits is not sufficient to store the specified range of values. For example, suppose an integer type with range 10 .. 13 is defined, and three bits of storage are allocated for that type. Is an error generated for this case? If no error is generated by the compiler, how is a case such as this treated?**

The following was compiled and an error stating "not sufficient storage" was received:

```
type Int is range 10 .. 13;
for Int'SIZE use 3;
```

7. **What impact does the length clause have on the packing algorithm of composite types?**

Detailed experimentation is required for resolution of this.

8. **What is the effect of pragma OPTIMIZE (TIME) on storage allocation when length clauses are used?**

Detailed experimentation is required for resolution of this.

9. **What is the effect of pragma OPTIMIZE (SPACE) on storage allocation when length clauses are used?**

Detailed experimentation is required for resolution of this.

10. **What is the effect of pragma PACK on storage allocation when length clauses are used?**

Detailed experimentation is required for resolution of this.

E. Enumeration Representation Clauses:

1. **Does the compiler support the use of enumeration representation clauses? What are the restrictions on their use?**

Yes. [PSEH, page 7-8] There are no documented restrictions. Preliminary tests show negative code values are allowed.

2. **Consider an enumeration type and associated enumeration representation clause where the enumerated values specified are not contiguous integers, such as:**

```
type Name is (Name_1, Name_2, Name_3, Name_4);
for Name use
  ( Name_1 => 1, Name_2 => 5, Name_3 => 12, Name_4 => 163 );
```

The enumeration type may not be efficiently implemented because of the noncontiguous nature of the integers specified in the enumeration representation clause, illustrated above. Hence, how are enumeration types represented internally, particularly in the case where enumeration clauses are specified with noncontiguous values?

Detailed experimentation is required for resolution of this.

3. **What is the effect of pragma PACK on storage allocation when enumeration representation clauses are used?**

Detailed experimentation is required for resolution of this.

4. **What is the effect of pragma OPTIMIZE (TIME) on storage allocation when enumeration representation clauses are used?**

Detailed experimentation is required for resolution of this.

5. **What is the effect of pragma OPTIMIZE (SPACE) on storage allocation when enumeration representation clauses are used?**

Detailed experimentation is required for resolution of this.

F. Record Representation Clauses:

1. **Does the compiler support the use of record representation clauses? What are the restrictions on their use?**

Yes. [PSEH, page 7-8] There are no documented restrictions.

2. **What are the restrictions on the use of the alignment clause in a record representation clause?**

The only values allowed for alignments are 1 and 2 (for word or doubleword alignment). [PSEH, page 7-11]

3. **What are the restrictions on the use of component clauses in a record representation clause?**

A component clause must specify enough bits to hold any value of the type of the component being allocated. Components of the following types cannot be specified with a component clause: access, array, record, and task. [PSEH, page 7-11]

4. **Are there restrictions on the overlap of record components with respect to the basic machine storage unit? For example, if a machine has a SYSTEM.STORAGE_UNIT equal to 16 bits, is it permitted to have components of a record that are larger than this value?**

Preliminary tests showed components can overlap storage unit boundaries. Thus, components can be larger than 16 bits, which is SYSTEM.STORAGE_UNIT.

5. **Consider the case when a record is specified with a record representation clause. Where is a record component placed which has no associated component clause?**

Storage is first allocated to those components that have component clauses. Following this, storage is allocated for the remaining components of the record using a first-fit algorithm. In particular, if there are gaps between the components specified with a component clause, the storage within these gaps is allocated by a first-fit algorithm. Note that the use of the a first-fit algorithm as indicated above minimizes total storage allocated for a record.

[PSEH, page 8-5]

6. **What is the effect of pragma OPTIMIZE (TIME) on storage allocation when record representation clauses are used?**

Detailed experimentation is required for resolution of this.

7. **What is the effect of pragma OPTIMIZE (SPACE) on storage allocation when record representation clauses are used?**

Detailed experimentation is required for resolution of this.

8. **What is the effect of pragma PACK on storage allocation when record representation clauses are used?**

Detailed experimentation is required for resolution of this.

G. Address Clauses:

1. **Does the compiler support the use of address clauses? What are the restrictions on their use?**

Yes. An address clause is only allowed for a single task entry. An address clause is allowed only within a task specification compiled with the EXECUTIVE compiler option. The values allowed for the simple expression in an address clause are the allowable interrupt entry addresses given in Appendix C, of reference [7]. If more than one task entry is equated to the same address, the most recently executed entry permanently overrides any previous entries. [PSEH, page 7-12]

Note that programs with an address clause specified for an object compile and link without error.

2. **What is the type SYSTEM.ADDRESS?**

Type SYSTEM.ADDRESS represents virtual addresses in the range 0 .. SYSTEM.MEMORY_SIZE-1. [PSEH, page 7-7]

3. **What is the effect of pragma OPTIMIZE (TIME) on storage allocation when address clauses are used?**

Detailed experimentation is required for resolution of this.

4. **What is the effect of pragma OPTIMIZE (SPACE) on storage allocation when address clauses are used?**

Detailed experimentation is required for resolution of this.

5. **Does the compiler enforce strong typing in the presence of address clauses? For example, is the following recognized as erroneous by the compiler:**

```
type T_1 is range 0 .. 100;
O_1 : T_1;
for O_1 use at 16#1000#;

type T_2 is digits 2 range 0.0 .. 100.0;
O_2 : T_2;
for O_2 use at 16#1000#;
```

In spite of the fact that address clauses for objects are not supported, this example (using allowable addresses) does compile without error.

6. **Does the compiler or linker recognize potential conflicts when address clauses are used? That is, suppose an address clause is present that references some address, say X. Assume that the address X is such that it lies within the address space of generated code. How is this case treated by the compiler and/or linker?**

This is not applicable to Ada/M(44) since the only allowable values for the simple expression in an address clause are predefined by Ada/M(44). Thus, such conflicts will not occur.

H. Data Conversion and Assignment:

1. **How is conversion accomplished between values of a type specified by the default representation and a type that is specified with a representation clause? (This refers to the use of a *new* (derived) type that is defined in terms of a representation clause.)**

Detailed experimentation is required for resolution of this.

2. **For conversions between objects of different types, does the compiler produce in-line code or generate a call to a library routine to accomplish the conversion?**

Examination of generated code from preliminary tests showed the following:

- *when converting an integer value to a floating-point value*, the integer value is viewed as a fixed-point value and the AN/UYK-44 instruction, FXC, which converts fixed-point values to floating-point values, is used

- *when converting a floating-point value to an integer value*, a call was made to a run-time library support routine to affect the conversion

The AN/UYK-44 instruction set also includes an instruction, FLC, which converts floating-point values to fixed-point values.

[TD]

3. **Is support of the generic function UNCHECKED_CONVERSION provided?**

Yes. [PSEH, page 7-12]

4. **Are there any restrictions on the use of UNCHECKED_CONVERSION? For example, are there any restrictions on the source and target types for UNCHECKED_CONVERSION? Do they have to be of the same size?**

The size of source and target objects must be the same. [PSEH, page 7-12]

I. Representation Attributes:

1. **What are the restrictions on the use of the 'ADDRESS representation attribute? How does the compiler interpret the use of this attribute?**

There are no documented restrictions.

2. **What are the restrictions on the use of the 'SIZE representation attribute? How does the compiler interpret the use of this attribute?**

There are no documented restrictions.

3. **What are the restrictions on the use of the 'POSITION representation attribute for a record component?**

There are no documented restrictions.

4. **What are the restrictions on the use of the 'FIRST_BIT representation attribute for a record component?**

There are no documented restrictions.

5. **What are the restrictions on the use of the 'LAST_BIT representation attribute for a record component?**

There are no documented restrictions.

6. **What is the effect of pragma OPTIMIZE (TIME) on the values of the representation** attributes?

Detailed experimentation is required for resolution of this.

7. **What is the effect of pragma OPTIMIZE (SPACE) on the values of the representation** attributes?

Detailed experimentation is required for resolution of this.

J. Miscellaneous:

1. **Suppose an object has been allocated storage where the number of bits is not sufficient to store the specified range of values. For example, suppose an object has been allocated three bits of storage, but is specified to be in the range 10 through 13. Is an error generated for this case? If no error is generated by the compiler, how is a case such as this treated?**

The following was compiled and an error stating "not enough storage specified for Rec.B" was received:

```
type Int is range 10 .. 13;

type Rec is
   record
      A : Int;
      B : Int;
   end record;

for Rec use
   record
      A at 0 range 0 .. 3;
      B at 0 range 4 .. 6;
   end record;
```

2. **Does the compiler support the use of pragma SUPPRESS?**

Yes. [PSEH, page 7-4]

3. **What restrictions are placed on the use of pragma SUPPRESS? For example, can every check be suppressed?**

The following restrictions exist:

- suppression of OVERFLOW_CHECK applies only to integer operations
- the pragma only has effect within the compilation unit in which it appears except if ELABORATION_CHECK is suppressed which applied at the declaration of a sub-program or task unit applies to all calls or activations.

[PSEH, page 7-4]

4. **Is pragma STORAGE_UNIT supported? If so, are there any restrictions on the argument?**

Yes. This pragma must appear at the start of the first compilation when creating a library. [PSEH, page 7-3]

5. **Is pragma INTERFACE supported? If so, are there any restrictions on the allowable forms and places of parameters and calls?**

Yes. Pragma INTERFACE (arg1, arg2) is supported where the first argument specifies the language and is restricted to the value MACROM_NORMAL. The second argument specifies the name of the

externally supplied subprogram. Thus, one can only interface to assembler language. [PSEH, page 7-3]

6. **Is pragma SHARED supported? If so, are there any restrictions on its use?**

Yes. There are no documented restrictions. [PSEH, page 7-3]

7. **Are there other implementation-dependent features supported such as pragmas or** attributes?

There is one Ada/M(44) defined attribute relevant to this area, namely PHYSICAL_ADDRESS. This attribute applies to an object and yields the absolute address in physical memory of the object. However, in a test case this attribute was applied to a variable inside a procedure and an error was received. [PSEH, page 7-5]

There is an option to the Ada/M(44) compiler, EXECUTIVE, which allows the user code to run in the executive state of target, and allows access to runtime support library subprograms and data. As stated in G.1, this option must be in effect when using address clauses. [PSEH, page 9-5]

Table of Contents