

# REPORT DOCUMENTATION PAGE

Form Approved  
OPM No.

## AD-A251 615



average 1 hour per response, including the time for reviewing instructions, searching existing data sources gathering on. Send comments regarding this burden estimate or any other aspect of this collection of information, including e, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA of Management and Budget, Washington, DC 20503.

RT

3. REPORT TYPE AND DATES

Final: 04 April 1992

4. TITLE AND

Validation Summary Report: Siemens Nixdorf Informationssysteme AG, Ada (SINIX) V4.1, Siemens Nixdorf MX300i under SINIX V5.41 (Host &Target), 92032511.11249

5. FUNDING

6.

IABG-AVF  
Ottobrunn, Federal Republic of Germany

7. PERFORMING ORGANIZATION NAME(S) AND

IABG-AVF, Industrieanlagen-Betriebsgesellschaft  
Dept. SZT/ Einsteinstrasse 20  
D-8012 Ottobrunn  
FEDERAL REPUBLIC OF GERMANY

PERFORMING  
ORGANIZATION

IABG-VSR 105

9. SPONSORING/MONITORING AGENCY NAME(S) AND

Ada Joint Program Office  
United States Department of Defense  
Pentagon, Rm 3E114  
Washington, D.C. 20301-3081

10. SPONSORING/MONITORING  
AGENCY

11. SUPPLEMENTARY

12a. DISTRIBUTION/AVAILABILITY

Approved for public release; distribution unlimited.

12b. DISTRIBUTION

13. (Maximum 200

Siemens Nixdorf Informationssysteme AG, Ada (SINIX) V4.1, Siemens Nixdorf MX300i under SINIX V5.41 (Host &Target), ACVC 1.11.

14. SUBJECT

Ada programming language, Ada Compiler Val. Summary Report, Ada Compiler Val. Capability, Val. Testing, Ada Val. Office, Ada Val. Facility, ANS/MIL-STD-1815A,

15. NUMBER OF

16. PRICE

17. SECURITY  
CLASSIFICATION  
UNCLASSIFIED

18. SECURITY  
UNCLASSIFIED

19. SECURITY  
CLASSIFICATION  
UNCLASSIFIED

20. LIMITATION OF

AVF Control Number: IABG-VSR 105  
04 April, 1992

Ada COMPILER  
VALIDATION SUMMARY REPORT:  
Certificate Number: 920325I1.11249  
Siemens Nixdorf Informationssysteme AG  
Ada (SINIX) V4.1  
Siemens Nixdorf MX300i under SINIX V5.41  
Host and Target



|                    |                                     |
|--------------------|-------------------------------------|
| Accession For      |                                     |
| Dist               | <input checked="" type="checkbox"/> |
| Special            | <input type="checkbox"/>            |
| Availability Codes | <input type="checkbox"/>            |
| By                 |                                     |
| Distribution/      |                                     |
| Availability Codes |                                     |
| Dist               | Avail and/or<br>Special             |
| A-1                |                                     |

Prepared By:  
IABG mbH, Abt. ITE  
Einsteinstr. 20  
W-8012 Ottobrunn  
Germany

92-14404



92 6 01 073

### Certificate Information

The following Ada implementation was tested and determined to pass ACVC 1.11. Testing was completed on 25 March, 1992.

Compiler Name and Version: Ada (SINIX) V4.1

Host Computer System: Siemens Nixdorf MX300i under SINIX Version 5.41

Target Computer System: Same as Host

See section 3.1 for any additional information about the testing environment.

As a result of this validation effort, Validation Certificate 92032511.11249 is awarded to Siemens Nixdorf AG. This certificate expires on 1 June 1993.

This report has been reviewed and is approved.

  
IABG, Abt. ITE  
Michael Tonndorf  
Einsteinstr. 20  
W-8012 Ottobrunn  
Germany

  
Ada Validation Organization  
Director, Computer & Software Engineering Division  
Institute for Defense Analyses  
Alexandria VA 22311

  
Ada Joint Program Office  
Dr. John Solomond, Director  
Department of Defense  
Washington DC 20301

## DECLARATION OF CONFORMANCE

The following declaration of conformance was supplied by the customer.

### Declaration of Conformance

**Customer:** Siemens Nixdorf Informationssysteme AG

**Certificate Awardee:** Siemens Nixdorf Informationssysteme AG

**Ada Validation Facility:** IABG mbH

**ACVC Version:** 1.11

**Ada Implementation:**

**Ada Compiler Name and Version:** Ada (SINIX) V4.1

**Host Computer System:** Siemens Nixdorf MX300i under  
SINIX Version V5.41

**Target Computer System:** Same as Host Computer System

**Declaration:**

I, the undersigned, declare that I have no knowledge of deliberate deviations from the Ada Language Standard ANSI/MIL-STD-1815A ISO 8652-1987 in the implementation listed above.

*CGK/TA4 Käb*

Customer Signature

*Konstanz*

DR. KÄÄB

26. März 1992

Date

## TABLE OF CONTENTS

|            |                                                 |     |
|------------|-------------------------------------------------|-----|
| CHAPTER 1  | INTRODUCTION                                    |     |
| 1.1        | USE OF THIS VALIDATION SUMMARY REPORT . . . . . | 1-1 |
| 1.2        | REFERENCES. . . . .                             | 1-1 |
| 1.3        | ACVC TEST CLASSES . . . . .                     | 1-2 |
| 1.4        | DEFINITION OF TERMS . . . . .                   | 1-2 |
| CHAPTER 2  | IMPLEMENTATION DEPENDENCIES                     |     |
| 2.1        | WITHDRAWN TESTS . . . . .                       | 2-1 |
| 2.2        | INAPPLICABLE TESTS. . . . .                     | 2-1 |
| 2.3        | TEST MODIFICATIONS. . . . .                     | 2-3 |
| CHAPTER 3  | PROCESSING INFORMATION                          |     |
| 3.1        | TESTING ENVIRONMENT . . . . .                   | 3-1 |
| 3.2        | SUMMARY OF TEST RESULTS . . . . .               | 3-1 |
| 3.3        | TEST EXECUTION. . . . .                         | 3-2 |
| APPENDIX A | MACRO PARAMETERS                                |     |
| APPENDIX B | COMPILATION SYSTEM OPTIONS                      |     |
| APPENDIX C | APPENDIX F OF THE Ada STANDARD                  |     |

## CHAPTER 1

### INTRODUCTION

The Ada implementation described above was tested according to the Ada Validation Procedures [Pro90] against the Ada Standard [Ada83] using the current Ada Compiler Validation Capability (ACVC). This Validation Summary Report (VSR) gives an account of the testing of this Ada implementation. For any technical terms used in this report, the reader is referred to [Pro90]. A detailed description of the ACVC may be found in the current ACVC User's Guide [UG89].

#### 1.1 USE OF THIS VALIDATION SUMMARY REPORT

Consistent with the national laws of the originating country, the Ada Certification Body may make full and free public disclosure of this report. In the United States, this is provided in accordance with the "Freedom of Information Act" (5 U.S.C. #552). The results of this validation apply only to the computers, operating systems, and compiler versions identified in this report.

The organizations represented on the signature page of this report do not represent or warrant that all statements set forth in this report are accurate and complete, or that the subject implementation has no nonconformities to the Ada Standard other than those presented. Copies of this report are available to the public from the AVF which performed this validation or from:

National Technical Information Service  
5285 Port Royal Road  
Springfield VA 22161

Questions regarding this report or the validation test results should be directed to the AVF which performed this validation or to:

Ada Validation Organization  
Computer and Software Engineering Division  
Institute for Defense Analyses  
1801 North Beauregard Street  
Alexandria VA 22311-1772

#### 1.2 REFERENCES

- [Ada83] Reference Manual for the Ada Programming Language, ANSI/MIL-STD-1815A, February 1983 and ISO 8652-1987.
- [Pro90] Ada Compiler Validation Procedures, Version 2.1, Ada Joint Program Office, August 1990.
- [UG89] Ada Compiler Validation Capability User's Guide, 21 June 1989.

### 1.3 ACVC TEST CLASSES

Compliance of Ada implementations is tested by means of the ACVC. The ACVC contains a collection of test programs structured into six test classes: A, B, C, D, E, and L. The first letter of a test name identifies the class to which it belongs. Class A, C, D, and E tests are executable. Class B and class L tests are expected to produce errors at compile time and link time, respectively.

The executable tests are written in a self-checking manner and produce a PASSED, FAILED, or NOT APPLICABLE message indicating the result when they are executed. Three Ada library units, the packages REPORT and SPRT13, and the procedure CHECK\_FILE are used for this purpose. The package REPORT also provides a set of identity functions used to defeat some compiler optimizations allowed by the Ada Standard that would circumvent a test objective. The package SPRT13 is used by many tests for Chapter 13 of the Ada Standard. The procedure CHECK\_FILE is used to check the contents of text files written by some of the Class C tests for Chapter 14 of the Ada Standard. The operation of REPORT and CHECK\_FILE is checked by a set of executable tests. If these units are not operating correctly, validation testing is discontinued.

Class B tests check that a compiler detects illegal language usage. Class B tests are not executable. Each test in this class is compiled and the resulting compilation listing is examined to verify that all violations of the Ada Standard are detected. Some of the class B tests contain legal Ada code which must not be flagged illegal by the compiler. This behavior is also verified.

Class L tests check that an Ada implementation correctly detects violation of the Ada Standard involving multiple, separately compiled units. Errors are expected at link time, and execution is attempted.

In some tests of the ACVC, certain macro strings have to be replaced by implementation-specific values -- for example, the largest integer. A list of the values used for this implementation is provided in Appendix A. In addition to these anticipated test modifications, additional changes may be required to remove unforeseen conflicts between the tests and implementation-dependent characteristics. The modifications required for this implementation are described in section 2.3.

For each Ada implementation, a customized test suite is produced by the AVF. This customization consists of making the modifications described in the preceding paragraph, removing withdrawn tests (see section 2.1), and possibly removing some inapplicable tests (see section 2.2 and [UG89]).

In order to pass an ACVC an Ada implementation must process each test of the customized test suite according to the Ada Standard.

### 1.4 DEFINITION OF TERMS

|                                           |                                                                                                                                                                                           |
|-------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Ada Compiler                              | The software and any needed hardware that have to be added to a given host and target computer system to allow transformation of Ada programs into executable form and execution thereof. |
| Ada Compiler Validation Capability (ACVC) | The means for testing compliance of Ada implementations, consisting of the test suite, the support programs, the ACVC user's guide and the template for the validation summary report.    |
| Ada Implementation                        | An Ada compiler with its host computer system and its target computer system.                                                                                                             |
| Ada Joint                                 | The part of the certification body which provides policy and                                                                                                                              |

|                                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
|-------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Program Office (AJPO)               | guidance for the Ada certification system.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| Ada Validation Facility (AVF)       | The part of the certification body which carries out the procedures required to establish the compliance of an Ada implementation.                                                                                                                                                                                                                                                                                                                                                                                                              |
| Ada Validation Organization (AVO)   | The part of the certification body that provides technical guidance for operations of the Ada certification system.                                                                                                                                                                                                                                                                                                                                                                                                                             |
| Compliance of an Ada Implementation | The ability of the implementation to pass an ACVC version.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| Computer System                     | A functional unit, consisting of one or more computers and associated software, that uses common storage for all or part of a program and also for all or part of the data necessary for the execution of the program; executes user-written or user-designated programs; performs user-designated data manipulation, including arithmetic operations and logic operations; and that can execute programs that modify themselves during execution. A computer system may be a stand-alone unit or may consist of several inter-connected units. |
| Conformity                          | Fulfillment by a product, process, or service of all requirements specified.                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| Customer                            | An individual or corporate entity who enters into an agreement with an AVF which specifies the terms and conditions for AVF services (of any kind) to be performed.                                                                                                                                                                                                                                                                                                                                                                             |
| Declaration of Conformance          | A formal statement from a customer assuring that conformity is realized or attainable on the Ada implementation for which validation status is realized.                                                                                                                                                                                                                                                                                                                                                                                        |
| Host Computer System                | A computer system where Ada source programs are transformed into executable form.                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| Inapplicable test                   | A test that contains one or more test objectives found to be irrelevant for the given Ada implementation.                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| ISO                                 | International Organization for Standardization.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| LRM                                 | The Ada standard, or Language Reference Manual, published as ANSI/MIL-STD-1815A-1983 and ISO 8652-1987. Citations from the LRM take the form "<section>.<subsection>:<paragraph>."                                                                                                                                                                                                                                                                                                                                                              |
| Operating System                    | Software that controls the execution of programs and that provides services such as resource allocation, scheduling, input/output control, and data management. Usually, operating systems are predominantly software, but partial or complete hardware implementations are possible.                                                                                                                                                                                                                                                           |
| Target Computer System              | A computer system where the executable form of Ada programs are executed.                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| Validated Ada Compiler              | The compiler of a validated Ada implementation.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| Validated Ada Implementation        | An Ada implementation that has been validated successfully either by AVF testing or by registration [Pro90].                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| Validation                          | The process of checking the conformity of an Ada compiler to                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |

the Ada programming language and of issuing a certificate for this implementation.

Withdrawn  
test

A test found to be incorrect and not used in conformity testing. A test may be incorrect because it has an invalid test objective, fails to meet its test objective, or contains erroneous or illegal use of the Ada programming language.

## CHAPTER 2

### IMPLEMENTATION DEPENDENCIES

#### 2.1 WITHDRAWN TESTS

The following tests have been withdrawn by the AVO. The rationale for withdrawing each test is available from either the AVO or the AVF. The publication date for this list of withdrawn tests is 02 August 1991.

|         |         |         |         |         |         |
|---------|---------|---------|---------|---------|---------|
| E28005C | B28006C | C32203A | C34006D | C35508I | C35508J |
| C35508M | C35508N | C35702A | C35702B | B41308B | C43004A |
| C45114A | C45346A | C45612A | C45612B | C45612C | C45651A |
| C46022A | B49008A | B49008B | A74006A | C74308A | B83022B |
| B83022H | B83025B | B83025D | B83026B | C83026A | C83041A |
| B85001L | C86001F | C94021A | C97116A | C98003B | BA2011A |
| CB7001A | CB7001B | CB7004A | CC1223A | BC1226A | CC1226B |
| BC3009B | BD1B02B | BD1B06A | AD1B08A | BD2A02A | CD2A21E |
| CD2A23E | CD2A32A | CD2A41A | CD2A41E | CD2A87A | CD2B15C |
| BD3006A | BD4008A | CD4022A | CD4022D | CD4024B | CD4024C |
| CD4024D | CD4031A | CD4051D | CD5111A | CD7004C | ED7005D |
| CD7005E | AD7006A | CD7006E | AD7201A | AD7201E | CD7204B |
| AD7206A | BD8002A | BD8004C | CD9005A | CD9005B | CDA201E |
| CE2107I | CE2117A | CE2117B | CE2119B | CE2205B | CE2405A |
| CE3111C | CE3116A | CE3118A | CE3411B | CE3412B | CE3607B |
| CE3607C | CE3607D | CE3812A | CE3814A | CE3902B |         |

#### 2.2 INAPPLICABLE TESTS

A test is inapplicable if it contains test objectives which are irrelevant for a given Ada implementation. Reasons for a test's inapplicability may be supported by documents issued by the ISO and the AJPO known as Ada Commentaries and commonly referenced in the format AI-ddddd. For this implementation, the following tests were determined to be inapplicable for the reasons indicated; references to Ada Commentaries are included as appropriate.

The following 201 tests have floating-point type declarations requiring more digits than `SYSTEM.MAX_DIGITS`:

|                       |                       |
|-----------------------|-----------------------|
| C24113L..Y (14 tests) | C35705L..Y (14 tests) |
| C35706L..Y (14 tests) | C35707L..Y (14 tests) |
| C35708L..Y (14 tests) | C35802L..Z (15 tests) |
| C45241L..Y (14 tests) | C45321L..Y (14 tests) |
| C45421L..Y (14 tests) | C45521L..Z (15 tests) |
| C45524L..Z (15 tests) | C45621L..Z (15 tests) |
| C45641L..Y (14 tests) | C46012L..Z (15 tests) |

C35713B, C45423B, B86001T, and C86006H check for the predefined type `SHORT_FLOAT`; for this implementation, there is no such type.

C35713C, B86001U, and C86006G check for the predefined type `LONG_FLOAT`; for this implementation, there is no such type.

C35713D and B86001Z check for a predefined floating-point type with a name other than `FLOAT`, `LONG_FLOAT`, or `SHORT_FLOAT`; for this implementation, there is no such type.

A35801E checks that `FLOAT'FIRST..FLOAT'LAST` may be used as a range constraint in a floating-point type declaration; for this implementation, that range exceeds the range of safe numbers of the largest predefined floating-point type and must be rejected. (See section 2.3.)

C45423A, C45523A, and C45622A check that the proper exception is raised if `MACHINE_OVERFLOWS` is `TRUE` and the results of various floating-point operations lie outside the range of the base type; for this implementation, `MACHINE_OVERFLOWS` is `FALSE`.

C45531M..P and C45532M..P (8 tests) check fixed-point operations for types that require a `SYSTEM.MAX_MANTISSA` of 47 or greater; for this implementation, `MAX_MANTISSA` is less than 47.

B86001Y uses the name of a predefined fixed-point type other than type `DURATION`; for this implementation, there is no such type.

CA2009C, and CA2009F check whether a generic unit can be instantiated before its body (and any of its subunits) is compiled; this implementation creates a dependence on generic units as allowed by AI-00408 and AI-00506 such that the compilation of the generic unit bodies makes the instantiating units obsolete. (See section 2.3.)

LA3004A..B, EA3004C..D, and CA3004E..F (6 tests) check `pragma INLINE` for procedures and functions; this implementation does not support `pragma INLINE`.

CD1009C checks whether a length clause can specify a non-default size for a floating-point type; this implementation does not support such sizes.

CD2A84A, CD2A84E, CD2A84I..J (2 tests), and CD2A84O use length clauses to specify non-default sizes for access types; this implementation does not support such sizes.

BD8001A, BD8003A, BD8004A..B (2 tests), and AD8011A use machine code insertions; this implementation provides no package `MACHINE_CODE`.

AE2101C and EE2201D..E (2 tests) use instantiations of package `SEQUENTIAL_IO` with unconstrained array types and record types with discriminants without defaults; these instantiations are rejected by this compiler.

AE2101H, EE2401D, and EE2401G use instantiations of package `DIRECT_IO` with unconstrained array types and record types with discriminants without defaults; these instantiations are rejected by this compiler.

The tests listed in the following table check that `USE_ERROR` is raised if the given file operations are not supported for the given combination of mode and access method; this implementation supports these operations.

| Test    | File Operation | Mode                 | File Access Method |
|---------|----------------|----------------------|--------------------|
| CE2102D | CREATE         | IN FILE              | SEQUENTIAL_IO      |
| CE2102E | CREATE         | OUT FILE             | SEQUENTIAL_IO      |
| CE2102F | CREATE         | INO $\bar{U}$ T FILE | DIRECT_IO          |
| CE2102I | CREATE         | IN FILE              | DIRECT_IO          |
| CE2102J | CREATE         | OUT FILE             | DIRECT_IO          |
| CE2102N | OPEN           | IN FILE              | SEQUENTIAL_IO      |
| CE2102O | RESET          | IN FILE              | SEQUENTIAL_IO      |
| CE2102P | OPEN           | OUT FILE             | SEQUENTIAL_IO      |
| CE2102Q | RESET          | OUT FILE             | SEQUENTIAL_IO      |
| CE2102R | OPEN           | INO $\bar{U}$ T FILE | DIRECT_IO          |
| CE2102S | RESET          | INO $\bar{U}$ T FILE | DIRECT_IO          |
| CE2102T | OPEN           | IN FILE              | DIRECT_IO          |
| CE2102U | RESET          | IN FILE              | DIRECT_IO          |
| CE2102V | OPEN           | OUT FILE             | DIRECT_IO          |
| CE2102W | RESET          | OUT FILE             | DIRECT_IO          |
| CE3102E | CREATE         | IN FILE              | TEXT_IO            |
| CE3102F | RESET          | Any Mode             | TEXT_IO            |
| CE3102G | DELETE         | -----                | TEXT_IO            |
| CE3102I | CREATE         | OUT FILE             | TEXT_IO            |
| CE3102J | OPEN           | IN FILE              | TEXT_IO            |
| CE3102K | OPEN           | OUT FILE             | TEXT_IO            |

The following 16 tests check operations on sequential, direct, and text files when multiple internal files are associated with the same external file and one or more are open for writing; USE\_ERROR is raised when this association is attempted.

|            |            |         |            |         |
|------------|------------|---------|------------|---------|
| CE2107B..E | CE2107G..H | CE2107L | CD2110B    | CE2110D |
| CE2111D    | CE2111H    | CE3111B | CE3111D..E | CE3114B |
| CE3115A    |            |         |            |         |

CE2203A checks that WRITE raises USE\_ERROR if the capacity of an external sequential file is exceeded; this implementation cannot restrict file capacity.

CE2403A checks that WRITE raises USE\_ERROR if the capacity of an external direct file is exceeded; this implementation cannot restrict file capacity.

CE3304A checks that SET LINE LENGTH and SET PAGE LENGTH raise USE\_ERROR if they specify an inappropriate value for the external file; there are no inappropriate values for this implementation.

CE3413B checks that PAGE raises LAYOUT ERROR when the value of the page number exceeds COUNT'LAST; for this implementation, the value of COUNT'LAST is greater than 150000, making the checking of this objective impractical.

## 2.3 TEST MODIFICATIONS

Modifications (see Section 1.3) were required for 8 tests.

The following tests were split into two or more tests because this implementation did not report the violations of the Ada Standard in the way expected by the original tests.

|         |         |         |
|---------|---------|---------|
| B22003A | B83033B | B85013D |
|---------|---------|---------|

A35801E was graded inapplicable by Evaluation Modification as directed by the AVO. The compiler rejects the use of the range FLOAT'FIRST..FLOAT'LAST as the range constraint of a floating-point type

declaration because the bounds lie outside of the range of safe numbers (cf. LRM 3.5.7:12).

CA2009C and CA2009F were graded inapplicable by Evaluation Modification as directed by the AVO. These tests contain instantiations of a generic unit prior to the compilation of that unit's body; as allowed by AI-00408 and AI-00506, the compilation of the generic unit bodies makes the compilation unit that contains the instantiations obsolete.

BC3204C and BC3205D were graded passed by Processing Modification as directed by the AVO. These tests check that instantiations of generic units with unconstrained types as generic actual parameters are illegal if the generic bodies contain uses of the types that require a constraint. However, the generic bodies are compiled after the units that contain the instantiations, and this implementation creates a dependence of the instantiating units on the generic units as allowed by AI-00408 and AI-00506 such that the compilation of the generic bodies makes the instantiating units obsolete--no errors are detected. The processing of these tests was modified by compiling the separate files in the following order (to allow re-compilation of obsolete units), and all intended errors were then detected by the compiler:

BC3204C: C0, C1, C2, C3M, C4, C5, C6, C3M

BC3205D: D0, D1M, D2, D1M

## CHAPTER 3

### PROCESSING INFORMATION

#### 3.1 TESTING ENVIRONMENT

The Ada implementation tested in this validation effort is described adequately by the information given in the initial pages of this report.

For technical information about this Ada implementation system, see:

CGK Computer Gesellschaft Konstanz mbH  
TA 4  
Dr. Käab  
Max-Stromeyer-Str. 168  
W-7750 Konstanz  
Tel: +49 7531 87 3910.

For sales information about this Ada implementation system, see:

Siemens Nixdorf Informationssysteme AG  
SP ZES 63  
Klaus Engelke  
Otto-Hahn-Ring 6  
W-8000 München 83  
Tel: +49 89 636 82549.

Testing of this Ada implementation was conducted at the customer's site by a validation team from the AVF.

#### 3.2 SUMMARY OF TEST RESULTS

An Ada Implementation passes a given ACVC version if it processes each test of the customized test suite in accordance with the Ada Programming Language Standard, whether the test is applicable or inapplicable; otherwise, the Ada Implementation fails the ACVC [Pro90].

For all processed tests (inapplicable and applicable), a result was obtained that conforms to the Ada Programming Language Standard.

The list of items below gives the number of ACVC tests in various categories. All tests were processed, except those that were withdrawn because of test errors (item b; see section 2.1), those that require a floating-point precision that exceeds the implementation's maximum precision (item e; see section 2.2), and those that depend on the support of a file system -- if none is supported (item d). All tests passed, except

those that are listed in sections 2.1 and 2.2 (counted in items b and f, below).

|                                                 |      |         |
|-------------------------------------------------|------|---------|
| a) Total Number of Applicable Tests             | 3786 |         |
| b) Total Number of Withdrawn Tests              | 95   |         |
| c) Processed Inapplicable Tests                 | 88   |         |
| d) Non-Processed I/O Tests                      | 0    |         |
| e) Non-Processed Floating-Point Precision Tests | 201  |         |
| f) Total Number of Inapplicable Tests           | 289  | (c+d+e) |
| g) Total Number of Tests for ACVC 1.11          | 4170 | (a+b+f) |

### 3.3 TEST EXECUTION

A magnetic data cartridge containing the customized test suite (see section 1.3) was taken on-site by the validation team for processing. The contents of the magnetic data cartridge were loaded directly onto the host computer.

After the test files were loaded onto the host computer, the full set of tests was processed by the Ada implementation. The tests were compiled, linked, and executed on the computer system, as appropriate.

Testing was performed using command scripts provided by the customer and reviewed by the validation team. See Appendix B for a complete listing of the processing options for this implementation. It also indicates the default options. The options invoked explicitly for compiling during this test were:

|     |                                                                                         |
|-----|-----------------------------------------------------------------------------------------|
| -fE | generate error log file                                                                 |
| -fI | ignore compilation errors and continue generating code within the same compilation file |
| -fQ | (quiet) suppress messages "added to library" and "Generating code for"                  |
| -fw | suppress warnings                                                                       |

The options invoked explicitly for linking during this test were:

|          |                                                                     |
|----------|---------------------------------------------------------------------|
| -s 75000 | amount of stack space reserved for task stacks in the program stack |
|----------|---------------------------------------------------------------------|

The options -fI, -fQ, and -s are not documented in the generic compiler user manual.

Test output, compiler and linker listings, and job logs were captured on magnetic data cartridge and archived at the AVF. The listings examined on-site by the validation team were also archived.

# APPENDIX A MACRO PARAMETERS

This appendix contains the macro parameters used for customizing the ACVC. The meaning and purpose of these parameters are explained in [UG89]. The parameter values are presented in two tables. The first table lists the values that are defined in terms of the maximum input-line length, which is the value for \$MAX\_IN\_LEN--also listed here. These values are expressed here as Ada string aggregates, where "V" represents the maximum input-line length.

| Macro Parameter              | Macro Value                                    |
|------------------------------|------------------------------------------------|
| \$MAX_IN_LEN                 | 240                                            |
| \$BIG_ID1                    | (1..V-1 => 'A', V => '1')                      |
| \$BIG_ID2                    | (1..V-1 => 'A', V => '2')                      |
| \$BIG_ID3                    | (1..V/2 => 'A') & '3' &<br>(1..V-1-V/2 => 'A') |
| \$BIG_ID4                    | (1..V/2 => 'A') & '4' &<br>(1..V-1-V/2 => 'A') |
| \$BIG_INT_LIT                | (1..V-3 => '0') & "298"                        |
| \$BIG_REAL_LIT               | (1..V-5 => '0') & "690.0"                      |
| \$BIG_STRING1                | "" & (1..V/2 => 'A') & ""                      |
| \$BIG_STRING2                | "" & (1..V-1-V/2 => 'A') & '1' & ""            |
| \$BLANKS                     | (1..V-20 => ' ')                               |
| \$MAX_LEN_INT_BASED_LITERAL  | "2:" & (1..V-5 => '0') & "11:"                 |
| \$MAX_LEN_REAL_BASED_LITERAL | "16:" & (1..V-7 => '0') & "F.E:"               |
| \$MAX_STRING_LITERAL         | "" & (1..V-2 => 'A') & ""                      |

The following table lists all of the other macro parameters and their respective values.

| Macro Parameter                       | Macro Value                     |
|---------------------------------------|---------------------------------|
| \$ACC_SIZE                            | 32                              |
| \$ALIGNMENT                           | 4                               |
| \$COUNT_LAST                          | 2147483646                      |
| \$DEFAULT_MEM_SIZE                    | 1024                            |
| \$DEFAULT_STOR_UNIT                   | 8                               |
| \$DEFAULT_SYS_NAME                    | I80486                          |
| \$DELTA_DOC                           | 2#1.0#E-31                      |
| \$ENTRY_ADDRESS                       | 16#0#                           |
| \$ENTRY_ADDRESS1                      | 16#1#                           |
| \$ENTRY_ADDRESS2                      | 16#2#                           |
| \$FIELD_LAST                          | 2_147_483_647                   |
| \$FILE_TERMINATOR                     | ' '                             |
| \$FIXED_NAME                          | NO_SUCH_FIXED_TYPE              |
| \$FLOAT_NAME                          | NO_SUCH_FLOAT_NAME              |
| \$FORM_STRING                         | " "                             |
| \$FORM_STRING2                        | "CANNOT_RESTRICT_FILE_CAPACITY" |
| \$GREATER_THAN_DURATION               | 90_000.0                        |
| \$GREATER_THAN_DURATION BASE LAST     | 10_000_000.0                    |
| \$GREATER_THAN_FLOAT_BASE LAST        | 1.8E+308                        |
| \$GREATER_THAN_FLOAT_SAFE_LARGE       | 1.0E+308                        |
| \$GREATER_THAN_SHORT_FLOAT_SAFE_LARGE | 1.0E+308                        |
| \$HIGH_PRIORITY                       | 20                              |
| \$ILLEGAL_EXTERNAL_FILE NAME1         | /NODIRECTORY/FILENAME1          |
| \$ILLEGAL_EXTERNAL_FILE NAME2         | /NODIRECTORY/FILENAME2          |
| \$INAPPROPRIATE_LINE_LENGTH           | -1                              |
| \$INAPPROPRIATE_PAGE_LENGTH           | -1                              |
| \$INCLUDE_PRAGMA1                     | PRAGMA INCLUDE ("A28006D1.TST") |

|                                 |                                 |
|---------------------------------|---------------------------------|
| \$INCLUDE_PRAGMA2               | PRAGMA INCLUDE ("B28006F1.TST") |
| \$INTEGER_FIRST                 | -2147483648                     |
| \$INTEGER_LAST                  | 2147483647                      |
| \$INTEGER_LAST_PLUS_1           | 2147483648                      |
| \$INTERFACE_LANGUAGE            | C                               |
| \$LESS_THAN_DURATION            | -90_000.0                       |
| \$LESS_THAN_DURATION_BASE_FIRST | -10_000_000.0                   |
| \$LINE_TERMINATOR               | ' '                             |
| \$LOW_PRIORITY                  | 1                               |
| \$MACHINE_CODE_STATEMENT        | NULL;                           |
| \$MACHINE_CODE_TYPE             | INSTRUCTION                     |
| \$MANTISSA_DOC                  | 31                              |
| \$MAX_DIGITS                    | 15                              |
| \$MAX_INT                       | 2147483647                      |
| \$MAX_INT_PLUS_1                | 2147483648                      |
| \$MIN_INT                       | -2147483648                     |
| \$NAME                          | BYTE_INTEGER                    |
| \$NAME_LIST                     | I80486                          |
| \$NEG_BASED_INT                 | 16#FFFFFFFFE#                   |
| \$NEW_MEM_SIZE                  | 1024                            |
| \$NEW_STOR_UNIT                 | 8                               |
| \$NEW_SYS_NAME                  | I80486                          |
| \$PAGE_TERMINATOR               | ASCII.LF & ASCLL.FF             |
| \$RECORD_DEFINITION             | NEW INTEGER                     |
| \$RECORD_NAME                   | INSTRUCTION                     |
| \$TASK_SIZE                     | 32                              |
| \$TASK_STORAGE_SIZE             | 2048                            |
| \$TICK                          | 1.0                             |
| \$VARIABLE_ADDRESS              | FCNDECL.VAR_ADDRESS             |
| \$VARIABLE_ADDRESS1             | FCNDECL.VAR_ADDRESS1            |
| \$VARIABLE_ADDRESS2             | FCNDECL.VAR_ADDRESS2            |

APPENDIX B  
COMPILATION AND LINKER SYSTEM OPTIONS

The compiler and linker options of this Ada implementation, as described in this Appendix, are provided by the customer. Unless specifically noted otherwise, references in this appendix are to compiler documentation and not to this report.

## 16.1 ada

### 16.1.1 Invocation

`ada [option ...] file.ada ...`

### 16.1.2 Description

The `ada` command invokes the Meridian Ada compiler.

A program library must be created using `mklib` or `newlib` in advance of any compilation. The compiler aborts if it is unable to find a program library (either the default, `ada.lib`, in the current working directory or the library name specified with the `-L` option).

Note that the source file has the extension `.ada`. Just about any non-empty file extension is permitted. The ones not allowed include those used by the Meridian Ada compiling system for other purposes such as `.o` for object module files. If an illegal extension is given, the error message "missing or improper file name" is displayed. Some other commonly used source file extensions are:

- `.ads` for package specification source files
- `.adb` for package body source files
- `.sub` for subunit (separate) source files

### 16.1.3 Options

- ED** Generate debugging output. The `-ED` option causes the compiler to generate the appropriate code and data for operation with the Meridian Ada Debugger. For more information on using this option and using the Debugger, see Chapter 9.
- EX** Generate error log file. The `-EX` option causes the compiler to generate a log file containing all the error messages and warning messages produced during compilation. The error log file has the same name as the source file, with the extension `.err`. For example, the error log file for `simple.ada` is `simple.err`. The error log file is placed in the current working directory. In the absence of the `-EX` option, the error log information is sent to the standard output stream.
- EL** Generate exception location information. The `-EL` option causes location information (source file names and line numbers) to be maintained for internal checks. This information is useful for debugging in the event that an "Exception never handled" message appears when an exception propagates out of the main program (see section 3.3). This option causes the code to be somewhat larger. If `-EL` is not used, exceptions that propagate out of the main program will behave in the same way, but no location information will be printed with the "Exception never handled" message.
- EN** Suppress numeric checking. The `-EN` option suppresses two kinds of numeric checks for the entire compilation:
  1. `division_check`
  2. `overflow_check`

These checks are described in section 11.7 of the LRM. Using `-EN` reduces the size of the code. Note that there is a related `ada` option, `-fs` to suppress all checks for a compilation. See also section 3.3.

The `-EN` option must be used in place of pragma `suppress` for the two numeric checks, because presently pragma `suppress` is not supported for `division_check` and `overflow_check`. Pragma

**suppress** works for other checks, as described in section 2.4.2. In the absence of the **-fN** option, the numeric checks are always performed.

- fs** Suppress all checks. The **-fs** option suppresses all automatic checking, including numeric checking. This option is equivalent to using pragma **suppress** on all checks. This option reduces the size of the code, and is good for producing "production quality" code or for benchmarking the compiler. Note that there is a related **ada** option, **-fN** to suppress only certain kinds of numeric checks. See also sections 2.4.2 and 3.3.
- fv** Compile verbosely. The compiler prints the name of each subprogram, package, or generic as it is compiled.
- fw** Suppress warning messages. With this option, the compiler does not print warning messages about ignored pragmas, exceptions that are certain to be raised at run-time, or other potential problems that the compiler is otherwise forbidden to deem as errors by the LRM.
- g** The **-g** option instructs the compiler to run an additional optimization pass. The optimizer removes common sub-expressions, dead code and unnecessary jumps. It also does loop optimizations. This option is different from the **-g** option to **bamp**. The **-g** option to **ada** optimizes the specified unit when it is compiled; no inter-unit optimization is done. The **-g** option to **bamp** analyzes and optimizes the entire program at link time. Note: Even if **-g** is specified for the **ada** command, the **-R** option to **ada** must still be specified for the **-g** option to **bamp** to be effective.
- R** Keep internal form file. This option is used in conjunction with the Optimizer (see Chapter 7 for more information). Without this option, the compiler deletes internal form files following code generation.

#### **-lmodifiers**

Generate listing file. The **-l** option causes the compiler to create a listing. Optional modifiers can be given to affect the listing format. You can use none or any combination of the following modifiers:

|          |                                    |
|----------|------------------------------------|
| <b>c</b> | continuous listing format          |
| <b>p</b> | obey pragma <b>page</b> directives |
| <b>s</b> | use standard output                |
| <b>t</b> | relevant text output only          |

The formats of and options for listings are discussed in section 16.1.7. The default listing file generated has the same name as the source file, with the extension **.lst**. For example, the default listing file produced for **simple.ada** has the name **simple.lst**. The listing file is placed in the current working directory. Note: **-l** also causes an error log file to be produced, as with the **-fE** option.

#### **-L library-name**

Default: **ada.lib**

Use alternate library. The **-L** option specifies an alternative name for the program library.

**Note:** Options beginning with **-f** can be combined, as in "**-fsv**." This is equivalent to specifying the options separately, e.g. "**-fs -fv**." Options beginning with **-l** can be similarly combined or separated, as in "**-lcs**" or "**-lc -ls**" (see section 16.1.7).

### 16.1.4 Compiler Output Files

Files produced by compilations, other than listings and error logs, are:

|                   |                                                            |
|-------------------|------------------------------------------------------------|
| <b>.atz</b> files | interface description files                                |
| <b>.int</b> files | Meridian Internal Form Files                               |
| <b>.gnn</b> files | generic description files; <i>nn</i> is a two-digit number |

## 16.4 bamp

### 16.4.1 Invocation

`bamp [option ...] [main-procedure-name]`

### 16.4.2 Description

The **bamp** (Build Ada Main Program) command creates an executable program given the name of the main subprogram. The *main-procedure-name* given to **bamp** must be a parameterless procedure that has already been compiled.

**Note:** Be careful not to confuse the name of the source file containing the main subprogram (e.g. *simple.ad*) with the actual name of the main subprogram (e.g. *simple*).

If a *main-procedure-name* is not specified on the **bamp** command line, **bamp** links using the last-compiled subprogram that fits the profile for a main subprogram. To determine which subprogram will be used when no main subprogram is given to **bamp**, use the `lslib -t` option. When in doubt, it may be best to specify the main subprogram explicitly.

Note that when no main subprogram is specified, **bamp** selects the most recently compiled subprogram, not the most recently linked subprogram. If several different main subprograms are linked between compiles, still the most recently compiled subprogram is selected if no subprogram is explicitly specified.

The **bamp** program functions as a high-level linker. It works by creating a top-level main program that contains all necessary context clauses and calls to package elaboration procedures. The main program is created as an internal form file on which the code generator is run. Following this code generation pass, all the required object files are linked.

An optional optimization pass can be invoked via the **bamp** command. The details of optimization are discussed in Chapter 7. The **bamp** options relevant to optimization, `-g` and `-G`, are discussed below.

Programs compiled in Debug mode (with the `ada -fD` option) are automatically linked with the Meridian Ada source level debugger.

### 16.4.3 Options

**-A** Aggressively inline. This option instructs the optimizer to aggressively inline subprograms when used in addition to the `-G` option. Typically, this means that subprograms that are only called once are inlined. If only the `-G` option is used, only subprograms for which `pragma inline` has been specified are inlined.

**-c** *compiler-program-name*

Default: As stored in program library.

Use alternate compiler. The `-c` option specifies the complete (non relative) directory path to the Meridian Ada compiler. This option overrides the compiler program name stored in the program library. The `-c` option is intended for use in cross-compiler configurations, although under such circumstances, an appropriate library configuration is normally used instead.

**-f** Suppress main program generation step. The `-f` option suppresses the creation and additional code generation steps for the temporary main program file. The `-f` option can be used when a simple change has been made to the body of a compilation unit. If unit elaboration order is changed, or if the specification of a unit is changed, or if new units are added, then this option should not be used. The

**-f** option saves a few seconds, but places an additional bookkeeping burden on you. The option should be avoided under most circumstances. Note that invoking **bamp** with the **-n** option followed by another invocation of **bamp** with the **-f** option has the same effect as an invocation of **bamp** with neither option (**-n** and **-f** neutralize each other).

- g** Perform global optimization only. The **-g** option causes **bamp** to invoke the global optimizer on your program. Compilation units to be optimized globally must have been compiled with the **ada -R** option.
- G** Perform global and local optimization. The **-G** option causes **bamp** to perform both global and local optimization on your program. This includes performing pragma **inline**. As with the **-g** option, compilation units to be optimized must have been compiled with the **ada -R** option.
- I** Link the program with a version of the tasking run-time which supports pre-emptive task scheduling. This option produces code which handles interrupts more quickly, but has a slight negative impact on performance in general.

**-L** *library-name*

Default: **ada.lib**

Use alternate library. The **-L** option specifies the name of the program library to be consulted by the **bamp** program. This option overrides the default library name.

- n** No link. The **-n** option suppresses actual object file linkage, but creates and performs code generation on the main program file. Note that invoking **bamp** with the **-n** option followed by another invocation of **bamp** with the **-f** option has the same effect as an invocation of **bamp** with neither option. That is, **-n** and **-f** neutralize each other.
- N** No operations. The **-N** option causes the **bamp** command to do a "dry run"; it prints out the actions it takes to generate the executable program, but does not actually perform those actions. The same kind of information is printed by the **-P** option.

**-o** *output-file-name*

Default: **file**

Use alternate executable file output name. The **-o** option specifies the name of the executable program file written by the **bamp** command. This option overrides the default output file name.

- P** Print operations. The **-P** option causes the **bamp** command to print out the actions it takes to generate the executable program as the actions are performed.
- v** Link verbosely. The **-v** option causes the **bamp** command to print out information about what actions it takes in building the main program such as:
  - The name of the program library consulted.
  - The library search order (listed as "uses" of the library units used by the program).
  - The name of the main program file created (as opposed to the main procedure name).
  - The elaboration order.
  - The total program stack size.
  - The name of the executable load module created.
  - The verbose code generation for the main program file.
- W** Suppress warnings. This option allows you to suppress warnings from the optimizer.

## APPENDIX C

### APPENDIX F OF THE Ada STANDARD

The only allowed implementation dependencies correspond to implementation-dependent pragmas, to certain machine-dependent conventions as mentioned in Chapter 13 of the Ada Standard, and to certain allowed restrictions on representation clauses. The implementation-dependent characteristics of this Ada implementation, as described in this Appendix, are provided by the customer. Unless specifically noted otherwise, references in this Appendix are to compiler documentation and not to this report. Implementation-specific portions of the package STANDARD can be found on page 155 of the compiler documentation.

## Appendix F Implementation-Dependent Characteristics

---

This appendix lists implementation-dependent characteristics of Meridian Ada. Note that there are no preceding appendices. This appendix is called *Appendix F* in order to comply with the Reference Manual for the Ada Programming Language\* (LRM) ANSI/MIL-STD-1815A which states that this appendix be named Appendix F.

Implemented Chapter 13 features include length clauses, enumeration representation clauses, record representation clauses, address clauses, interrupts, package **system**, machine code insertions, pragma **interface**, and unchecked programming.

### F.1 Pragmas

The implemented pre-defined pragmas are:

|                  |                                                                                                                                   |
|------------------|-----------------------------------------------------------------------------------------------------------------------------------|
| <b>elaborate</b> | See the LRM section 10.5.                                                                                                         |
| <b>interface</b> | See section F.1.1.                                                                                                                |
| <b>list</b>      | See the LRM Appendix B.                                                                                                           |
| <b>pack</b>      | See section F.1.2.                                                                                                                |
| <b>page</b>      | See the LRM Appendix B.                                                                                                           |
| <b>priority</b>  | See the LRM Appendix B.                                                                                                           |
| <b>suppress</b>  | See section F.1.3.                                                                                                                |
| <b>inline</b>    | See the LRM section 6.3.2. This pragma is not actually effective unless you compile/link your program using the global optimizer. |

The remaining pre-defined pragmas are accepted, but presently ignored:

|                    |                     |                    |
|--------------------|---------------------|--------------------|
| <b>controlled</b>  | <b>optimize</b>     | <b>system_name</b> |
| <b>shared</b>      | <b>storage_unit</b> |                    |
| <b>memory_size</b> |                     |                    |

Named parameter notation for pragmas is not supported.

When illegal parameter forms are encountered at compile time, the compiler issues a warning message rather than an error, as required by the Ada language definition. Refer to the LRM Appendix B for additional information about the pre-defined pragmas.

#### F.1.1 Pragma Interface

The form of pragma **interface** in Meridian Ada is:

```
pragma interface ( language, subprogram [, "link-name" ] );
```

where:

|                   |                                                                                                                                                                                                                                                      |
|-------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>language</b>   | This is the interface language, one of the names <b>assembly</b> , <b>builtin</b> , <b>c</b> , or <b>internal</b> . The names <b>builtin</b> and <b>internal</b> are reserved for use by Meridian compiler maintainers in run-time support packages. |
| <b>subprogram</b> | This is the name of a subprogram to which the pragma <b>interface</b> applies.                                                                                                                                                                       |

---

\*All future references to the Reference Manual for the Ada Programming Language appear as the LRM.

**link-name** This is an optional string literal specifying the name of the non-Ada subprogram corresponding to the Ada subprogram named in the second parameter. If **link-name** is omitted, then **link-name** defaults to the value of **subprogram** translated to lowercase. Depending on the language specified, some automatic modifications may be made to the **link-name** to produce the actual object code symbol name that is generated whenever references are made to the corresponding Ada subprogram.

It is appropriate to use the optional **link-name** parameter to **pragma interface** only when the interface subprogram has a name that does not correspond at all to its Ada identifier or when the interface subprogram name cannot be given using rules for constructing Ada identifiers (e.g. if the name contains a '\$' character).

The characteristics of object code symbols generated for each interface language are:

|                 |                                                                                                                                                                                                                                                                                                                                                                                                            |
|-----------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>assembly</b> | The object code symbol is the same as <b>link-name</b> .                                                                                                                                                                                                                                                                                                                                                   |
| <b>builtin</b>  | The object code symbol is the same as <b>link-name</b> , but prefixed with two underscore characters ("__"). This language interface is reserved for special interfaces defined by Meridian Software Systems, Inc. The <b>builtin</b> interface is presently used to declare certain low-level run-time operations whose names must not conflict with programmer-defined or language system defined names. |
| <b>c</b>        | The object code symbol is the same as <b>link-name</b> , but with one underscore character ('_') prepended. This is the convention used by the C compiler.                                                                                                                                                                                                                                                 |
| <b>internal</b> | No object code symbol is generated for an internal language interface; this language interface is reserved for special interfaces defined by Meridian Software Systems, Inc. The <b>internal</b> interface is presently used to declare certain machine-level bit operations.                                                                                                                              |

No automatic data conversions are performed on parameters of any interface subprograms. It is up to the programmer to ensure that calling conventions match and that any necessary data conversions take place when calling interface subprograms.

A **pragma interface** may appear within the same declarative part as the subprogram to which the **pragma interface** applies, following the subprogram declaration, and prior to the first use of the subprogram. A **pragma interface** that applies to a subprogram declared in a package specification must occur within the same package specification as the subprogram declaration; the **pragma interface** may not appear in the package body in this case. A **pragma interface** declaration for either a private or nonprivate subprogram declaration may appear in the private part of a package specification.

**Pragma interface** for library units is not supported.

Refer to the LRM section 13.9 for additional information about **pragma interface**.

## F.1.2 Pragma Pack

**Pragma pack** is implemented for composite types (records and arrays).

**Pragma pack** is permitted following the composite type declaration to which it applies, provided that the **pragma** occurs within the same declarative part as the composite type declaration, before any objects or components of the composite type are declared.

Note that the declarative part restriction means that the type declaration and accompanying **pragma pack** cannot be split across a package specification and body.

The effect of **pragma pack** is to minimize storage consumption by discrete component types whose ranges permit packing. Use of **pragma pack** does not defeat allocations of alignment storage gaps for some record types. **Pragma pack** does not affect the representations of real types, pre-defined integer types, and access types.

### F.1.3 Pragma Suppress

Pragma `suppress` is implemented as described in the LRM section 11.7, with these differences:

- Presently, `division_check` and `overflow_check` must be suppressed via a compiler flag, `-ZN`; pragma `suppress` is ignored for these two numeric checks.
- The optional "`ON =>`" parameter name notation for pragma `suppress` is ignored.
- The optional second parameter to pragma `suppress` is ignored; the pragma always applies to the entire scope in which it appears.

### F.2 Attributes

All attributes described in the LRM Appendix A are supported.

### F.3 Standard Types

Additional standard types are defined in Meridian Ada:

- `byte_integer`
- `short_integer`
- `long_integer`

The standard numeric types are defined as:

```

type byte_integer is range -128 .. 127;
type short_integer is range -32768 .. 32767;
type integer is range -2147483648 .. 2147483647;
type long_integer is range -2147483648 .. 2147483647;
type float is digits 15
  range -1.79769313486231E+308 .. 1.79769313486231E+308;
type duration is delta 0.0001 range -86400.0000 .. 86400.0000;
```

### F.4 Package System

The specification of package `system` is:

```

package system is
  type address is new integer;
  type name is (180486);
  system_name : constant name := 180486;
  storage_unit : constant := 8;
  memory_size : constant := 1024;
  -- System-Dependent Named Numbers
  min_int : constant := -2147483648;
  max_int : constant := 2147483647;
  max_digits : constant := 15;
  max_mantissa : constant := 31;
  fine_delta : constant := 2.0 ** (-31);
  tick : constant := 1.0;
```

### — Other System-Dependent Declarations

`subtype priority is integer range 1 .. 20;`

The value of `system.memory_size` is presently meaningless.

## F.5 Restrictions on Representation Clauses

### F.5.1 Length Clauses

A size specification (`t'size`) is rejected if fewer bits are specified than can accommodate the type. The minimum size of a composite type may be subject to application of `pragma pack`. It is permitted to specify precise sizes for unsigned integer ranges, e.g. 8 for the range 0 .. 255. However, because of requirements imposed by the Ada language definition, a full 32-bit range of unsigned values, i.e. 0 ..  $(2^{32}) - 1$ , cannot be defined, even using a size specification.

The specification of collection size (`t'storage_size`) is evaluated at run-time when the scope of the type to which the length clause applies is entered, and is therefore subject to rejection (via `storage_error`) based on available storage at the time the allocation is made. A collection may include storage used for run-time administration of the collection, and therefore should not be expected to accommodate a specific number of objects. Furthermore, certain classes of objects such as unconstrained discriminant array components of records may be allocated outside a given collection, so a collection may accommodate more objects than might be expected.

The specification of storage for a task activation (`t'storage_size`) is evaluated at run-time when a task to which the length clause applies is activated, and is therefore subject to rejection (via `storage_error`) based on available storage at the time the allocation is made. Storage reserved for a task activation is separate from storage needed for any collections defined within a task body.

The specification of `small` for a fixed point type (`t'small`) is subject only to restrictions defined in the LRM section 13.2.

### F.5.2 Enumeration Representation Clauses

The internal code for the literal of an enumeration type named in an enumeration representation clause must be in the range of `standard.integer`.

The value of an internal code may be obtained by applying an appropriate instantiation of `unchecked_conversion` to an integer type.

### F.5.3 Record Representation Clauses

The storage unit offset (the `at static_simple_expression` part) is given in terms of 8-bit storage units and must be even.

A bit position (the `range` part) applied to a discrete type component may be in the range 0 .. 15, with 0 being the least significant bit of a component. A range specification may not specify a size smaller than can accommodate the component. A range specification for a component not accommodating bit packing may have a higher upper bound as appropriate (e.g. 0 .. 31 for a discriminant `string` component). Refer to the internal data representation of a given component in determining the component size and assigning offsets.

Components of discrete types for which bit positions are specified may not straddle 16-bit word boundaries.

The value of an alignment clause (the optional `at mod` part) must evaluate to 1, 2, 4, or 8, and may not be smaller than the highest alignment required by any component of the record.

### F.5.4 Address Clauses

An address clause may be supplied for an object (whether constant or variable) or a task entry, but not for a subprogram, package, or task unit. The meaning of an address clause supplied for a task entry is given in section F.5.5.

An address expression for an object is a 32-bit memory address of type `system.address`.

### F.5.5 Interrupts

A task entry's address clause can be used to associate the entry with a UNIX signal. Values in the range 0 . . 31 are meaningful, and represent the signals corresponding to those values.

An interrupt entry may not have any parameters.

### F.5.6 Change of Representation

There are no restrictions for changes of representation effected by means of type conversion.

## F.6 Implementation-Dependent Components

No names are generated by the implementation to denote implementation-dependent components.

### F.7 Unchecked Conversions

There are no restrictions on the use of `unchecked_conversion`. Conversions between objects whose sizes do not conform may result in storage areas with undefined values.

## F.8 Input-Output Packages

A summary of the implementation-dependent input-output characteristics is:

- In calls to `open` and `create`, the `form` parameter must be the empty string (the default value).
- More than one internal file can be associated with a single external file for reading only. For writing, only one internal file may be associated with an external file; Do not use `reset` to get around this rule.
- Temporary sequential and direct files are given names. Temporary files are deleted when they are closed.
- File I/O is buffered; text files associated with terminal devices are line-buffered.
- The packages `sequential_io` and `direct_io` cannot be instantiated with unconstrained composite types or record types with discriminants without defaults.

### F.9 Source Line and Identifier Lengths

Source lines and identifiers in Ada source programs are presently limited to 200 characters in length.