AD-A208 831

(2)

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE	READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER | 2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) 88121511.09159 Ada Compiler Validation Summary Report: KRUPP ATLAS ELEKTRONIK GMBH, KRUPP ATLAS ELEKTRONIK GMBH Ada Compiler VMME 1.8 A, VAX 11/785 under VMS 4.5 (host) to KRUPP ATLAS ELEKTRONIK GMBH MPR 2300 (target)	5. TYPE OF REPORT & PERIOD COVERED 1 Dec. 1988 - 1 Dec. 1989 6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) IABG, Ottobrunn, Federal Republic of Germany.	8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION AND ADDRESS IABG, Ottobrunn, Federal Republic of Germany.	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Ada Joint Program Office United States Department of Defense Washington, DC 20301-3081	12. REPORT DATE 13. NUMBER OF PAGES
14. MONITORING AGENCY NAME & ADDRESS (If different from Controlling Office) IABG, Ottobrunn, Federal Republic of Germany.	15. SECURITY CLASS (of this report) UNCLASSIFIED 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE N/A

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

UNCLASSIFIED

18. SUPPLEMENTARY NOTES

19. KEYWORDS (Continue on reverse side if necessary and identify by block number)

Ada Programming language, Ada Compiler Validation Summary Report, Ada Compiler Validation Capability, ACVC, Validation Testing, Ada Validation Office, AVO, Ada Validation Facility, AVF, ANSI/MIL-STD-1815A, Ada Joint Program Office, AJPO

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

KRUPP ATLAS ELEKTRONIK GMBH Ada Compiler VVME 1.8 A, KRUPP ATLAS ELEKTRONIK GMBH, IABG, VAX 11/785 under VMS Version 4.5 combined with KRUPP ATLAS ELEKTRONIK GMBH MPR 1300 under MOS 1.213 (host) to KRUPP ATLAS ELEKTRONIK GMBH MPR 2300 under EOS 2300 Version 1.1 (target), ACVC 1.09

DD FORM 1473 EDITION OF 1 NOV 65 IS OBSOLETE

AVF Control Number: AVF-VSR-019
SZT-AVF-019

Ada COMPILER
VALIDATION SUMMARY REPORT:
Certificate Number: 881215I1.09159
KRUPP ATLAS ELEKTRONIK GMBH
KRUPP ATLAS ELEKTRONIK GMBH Ada Compiler VMME 1.8 A
Host: VAX 11/785 under VMS 4.5 combined with
KRUPP ATLAS ELEKTRONIK GMBH MPR 1300 under MOS 1.213
Target: KRUPP ATLAS ELEKTRONIK GMBH MPR 2300 under EOS 2300 1.1

Completion of On-Site Testing:
December 15th, 1988

Prepared By:
IABG mbH, Abt. SZT
Einsteinstr 20
D8012 Ottobrunn
West Germany

Prepared for:
Ada Joint Program Office
United States Department of Defense
Washington, D.C. 20301-3081

Ada Compiler Validation Summary Report:

Compiler Name: KRUPP ATLAS ELEKTRONIK GMBH Ada Compiler VVME 1.8 A
Compiler Version: 1.8 A

Certificate Number: 881215I1.09159

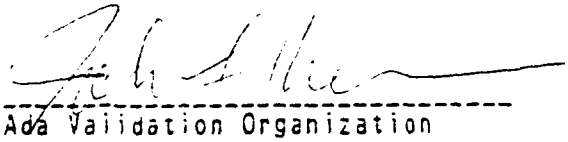
Host:
VAX 11/785 under VMS Version 4.5 combined with
KRUPP ATLAS ELEKTRONIK GMBH MPR 1300 under MOS 1.213

Target:
KRUPP ATLAS ELEKTRONIK GMBH MPR 2300 under
EOS 2300 Version 1.1

Testing Completed December 15th, 1988 Using ACVC 1.9

This report has been reviewed and is approved.

IABG mbH, Abt. SZT
Dr. H. Hummel
Einsteinstr 20
D8012 Ottobrunn
West Germany



Ada Validation Organization
Dr. John F. Kramer
Institute for Defense Analyses
Alexandria VA 22311

Ada Joint Program Office
William S. Ritchie, Acting Director
Department of Defense
Washington DC 20301

Ada Compiler Validation Summary Report:

Compiler Name: KRUPP ATLAS ELEKTRONIK GMBH Ada Compiler VVME 1.8 A
Compiler Version: 1.8 A

Certificate Number: 881215I1.09159

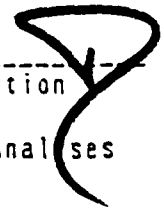
Host:
VAX 11/785 under VMS Version 4.5 combined with
KRUPP ATLAS ELEKTRONIK GMBH MPR 1300 under MOS 1.213

Target:
KRUPP ATLAS ELEKTRONIK GMBH MPR 2300 under
EOS 2300 Version 1.1

Testing Completed December 15th, 1988 Using ACVC 1.9

This report has been reviewed and is approved.

IABG mbH, Abt. SZT
Dr. H. Hummel
Einsteinstr 20
D8012 Ottobrunn
West Germany



Ada Validation Organization
Dr. John F. Kramer
Institute for Defense Analyses
Alexandria VA 22311

Ada Joint Program Office
William S. Ritchie, Acting Director
Department of Defense
Washington DC 20301

CONTENTS

CHAPTER 1

INTRODUCTION

This Validation Summary Report (VSR) describes the extent to which a specific Ada compiler conforms to the Ada Standard, ANSI/MIL-STD-1815A. This report explains all technical terms used within it and thoroughly reports the results of testing this compiler using the Ada Compiler Validation Capability (ACVC). An Ada compiler must be implemented according to the Ada Standard, and any implementation-dependent features must conform to the requirements of the Ada Standard. The Ada Standard must be implemented in its entirety, and nothing can be implemented that is not in the Standard.

Even though all validated Ada compilers conform to the Ada Standard, it must be understood that some differences do exist between implementations. The Ada Standard permits some implementation dependencies--for example, the maximum length of identifiers or the maximum values of integer types. Other differences between compilers result from the characteristics of particular operating systems, hardware, or implementation strategies. All the dependencies observed during the process of testing this compiler are given in this report.

The information in this report is derived from the test results produced during validation testing. The validation process includes submitting a suite of standardized tests, the ACVC, as inputs to an Ada compiler and evaluating the results. The purpose of validating is to ensure conformity of the compiler to the Ada Standard by testing that the compiler properly implements legal language constructs and that it identifies and rejects illegal language constructs. The testing also identifies behavior that is implementation dependent but permitted by the Ada Standard. Six classes of tests are used. These tests are designed to perform checks at compile time, at link time, and during execution.

1.1 PURPOSE OF THIS VALIDATION SUMMARY REPORT

This VSR documents the results of the validation testing performed on an Ada compiler. Testing was carried out for the following purposes:

- To attempt to identify any language constructs supported by the compiler that do not conform to the Ada Standard
- To attempt to identify any language constructs not supported by the compiler but required by the Ada Standard
- To determine that the implementation-dependent behavior is allowed by the Ada Standard

Testing of this compiler was conducted IABG mbH under the direction of the AVF according to procedures established by the Ada Joint Program Office and administered by the Ada Validation Organization (AVO). On-site testing was completed December 15th, 1988 at KRUPP ATLAS ELEKTRONIK GMBH, Bremen.

1.2 USE OF THIS VALIDATION SUMMARY REPORT

Consistent with the national laws of the originating country, the AVO 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 compiler has no nonconformities to the Ada Standard other than those presented. Copies of this report are available to the public from:

Ada Information Clearinghouse
Ada Joint Program Office
OUSDRE
The Pentagon, Rm 3D-139 (Fern Street)
Washington DC 20301-3081

or from:

IABG mbH, Abt. SZT,
Einsteinstr 20,
D8012 OTTOBRUNN,
West Germany.

Questions regarding this report or the validation test results should be directed to the AVF listed above or to:

Ada Validation Organization
Institute for Defense Analyses
1801 North Beauregard Street
Alexandria VA 22311

1.3 REFERENCES

1. Reference Manual for the Ada Programming Language, ANSI/MIL-STD-1815A, February 1983 and ISO 8652-1987.

2. Ada Compiler Validation Procedures and Guidelines, Ada Joint Program Office, 1 January 1987.

3. Ada Compiler Validation Capability Implementers' Guide, SofTech, Inc., December 1986.

4. Ada Compiler Validation Capability User's Guide, December 1986.

1.4 DEFINITION OF TERMS

ACVC	The Ada Compiler Validation Capability. The set of Ada programs that tests the conformity of an Ada compiler to the Ada programming language.
Ada Commentary	An Ada Commentary contains all information relevant to the point addressed by a comment on the Ada Standard. These comments are given a unique identification number having the form AI-ddddd.
Ada Standard	ANSI/MIL-STD-1815A, February 1983 and ISO 8652-1987.
Applicant	The agency requesting validation.
AVF	The Ada Validation Facility. The AVF is responsible for conducting compiler validations according to procedures contained in the Ada Compiler Validation Procedures and Guidelines.
AVO	The Ada Validation Organization. The AVO has oversight authority over all AVF practices for the purpose of maintaining a uniform process for validation of Ada compilers. The AVO provides administrative and technical support for Ada validations to ensure consistent practices.
Compiler	A processor for the Ada language. In the context of this report, a compiler is any language processor, including cross-compilers, translators, and interpreters.
Failed test	An ACVC test for which the compiler generates a result that demonstrates nonconformity to the Ada Standard.
Host	The computer on which the compiler resides.
Inapplicable test	An ACVC test that uses features of the language that a compiler is not required to support or may legitimately support in a way other than the one expected by the test.
Passed test	An ACVC test for which a compiler generates the expected result.
Target	The computer for which a compiler generates code.
Test	A program that checks a compiler's conformity regarding a particular feature or a combination of features to the Ada Standard. In the context of this report, the term is used to designate a single test, which may comprise one or more files.
Withdrawn test	An ACVC test found to be incorrect and not used to check conformity to the Ada Standard. A test may be incorrect

because it has an invalid test objective, fails to meet its test objective, or contains illegal or erroneous use of the language.

1.5 ACVC TEST CLASSES

Conformity to the Ada Standard is measured using the ACVC. The ACVC contains both legal and illegal Ada 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, and special program units are used to report their results during execution. Class B tests are expected to produce compilation errors. Class L tests are expected to produce compilation or link errors.

Class A tests check that legal Ada programs can be successfully compiled and executed. There are no explicit program components in a Class A test to check semantics. For example, a Class A test checks that reserved words of another language (other than those already reserved in the Ada language) are not treated as reserved words by an Ada compiler. A Class A test is passed if no errors are detected at compile time and the program executes to produce a PASSED message.

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 every syntax or semantic error in the test is detected. A Class B test is passed if every illegal construct that it contains is detected by the compiler.

Class C tests check that legal Ada programs can be correctly compiled and executed. Each Class C test is self-checking and produces a PASSED, FAILED, or NOT APPLICABLE message indicating the result when it is executed.

Class D tests check the compilation and execution capacities of a compiler. Since there are no capacity requirements placed on a compiler by the Ada Standard for some parameters--for example, the number of identifiers permitted in a compilation or the number of units in a library--a compiler may refuse to compile a Class D test and still be a conforming compiler. Therefore, if a Class D test fails to compile because the capacity of the compiler is exceeded, the test is classified as inapplicable. If a Class D test compiles successfully, it is self-checking and produces a PASSED or FAILED message during execution.

Each Class E test is self-checking and produces a NOT APPLICABLE, PASSED, or FAILED message when it is compiled and executed. However, the Ada Standard permits an implementation to reject programs containing some features addressed by Class E tests during compilation. Therefore, a Class E test is passed by a compiler if it is compiled successfully and executes to produce a PASSED message, or if it is rejected by the compiler for an allowable reason.

Class L tests check that incomplete or illegal Ada programs involving multiple, separately compiled units are detected and not allowed to execute. Class L tests are compiled separately and execution is attempted. A Class L test passes if it is rejected at link time--that is, an attempt to execute the main program must generate an error message before any declarations in the main program or any units referenced by the main program are elaborated.

Two library units, the package REPORT and the procedure CHECK_FILE, support the self-checking features of the executable tests. The package REPORT provides the mechanism by which executable tests report PASSED, FAILED, or NOT APPLICABLE results. It 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 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. These tests produce messages that are examined to verify that the units are operating correctly. If these units are not operating correctly, then the validation is not attempted.

The text of the tests in the ACVC follow conventions that are intended to ensure that the tests are reasonably portable without modification. For example, the tests make use of only the basic set of 55 characters, contain lines with a maximum length of 72 characters, use small numeric values, and place features that may not be supported by all implementations in separate tests. However, some tests contain values that require the test to be customized according to implementation-specific values--for example, an illegal file name. A list of the values used for this validation is provided in Appendix C.

A compiler must correctly process each of the tests in the suite and demonstrate conformity to the Ada Standard by either meeting the pass criteria given for the test or by showing that the test is inapplicable to the implementation. The applicability of a test to an implementation is considered each time the implementation is validated. A test that is inapplicable for one validation is not necessarily inapplicable for a subsequent validation. Any test that was determined to contain an illegal language construct or an erroneous language construct is withdrawn from the ACVC and, therefore, is not used in testing a compiler. The tests withdrawn at the time of this validation are given in Appendix D.

CHAPTER 2

CONFIGURATION INFORMATION

2.1 CONFIGURATION TESTED

The candidate compilation system for this validation was tested under the following configuration:

Compiler: KRUPP ATLAS ELEKTRONIK GMBH Ada Compiler VVME 1.8 A

ACVC Version: 1.9

Certificate Number: 88121SI1.09159

Host Computer:

Machine:	VAX 11/785 combined with KRUPP ATLAS ELEKTRONIK GMBH MPR 1300
Operating System:	VMS Version 4.5, MOS Version 1.213
Memory Size:	40 MBytes, 5.5 MBytes

Target Computer:

Machine:	KRUPP ATLAS ELEKTRONIK GMBH MPR 2300
Operating System:	EOS 2300 Version 1.1
Memory Size:	4 MBytes

Communications Network: Magnetic tape, HDLC connection, V24

It should be noted that the target computer does not contain permanent storage and that the target operating system does not provide a file system.

2.2 IMPLEMENTATION CHARACTERISTICS

One of the purposes of validating compilers is to determine the behavior of a compiler in those areas of the Ada Standard that permit implementations to differ. Class D and E tests specifically check for such implementation differences. However, tests in other classes also characterize an implementation. The tests demonstrate the following characteristics:

. Capacities.

  The compiler correctly processes tests containing loop statements nested to 65 levels, block statements nested to 65 levels, and recursive procedures separately compiled as subunits nested to 17 levels. It correctly processes a compilation containing 723 variables in the same declarative part. (See tests D55A03A..H (8 tests), D56001B, D64005E..G (3 tests), and D29002K.)

. Universal integer calculations.

  An implementation is allowed to reject universal integer calculations having values that exceed SYSTEM.MAX_INT. This implementation processes 64 bit integer calculations. (See tests D4A002A, D4A002B, D4A004A, and D4A004B.)

. Predefined types.

  This implementation supports the additional predefined types SHORT_INTEGER, SHORT_FLOAT and LONG_FLOAT in the package STANDARD. (See tests B86001C and B86001D.)

. Based literals.

  An implementation is allowed to reject a based literal with a value exceeding SYSTEM.MAX_INT during compilation, or it may raise NUMERIC_ERROR or CONSTRAINT_ERROR during execution. This implementation raises CONSTRAINT_ERROR during execution. (See test E24101A.)

. Expression evaluation.

  Apparently no default initialization expressions for record components are evaluated before any value is checked to belong to a component's subtype. (See test C32117A.)

  Assignments for subtypes are performed with the same precision as the base type. (See test C35712B.)

This implementation uses no extra bits for extra precision. This implementation uses all extra bits for extra range. (See test C35903A.)

No exception is raised when an integer literal operand in a comparison or membership test is outside the range of the base type. (See test C45232A.)

No exception is raised when a literal operand in a fixed-point comparison or membership test is outside the range of the base type. (See test C45252A.)

Apparently underflow is gradual. (See tests C45524A..Z.)

. Rounding.

The method used for rounding to integer is apparently round to even . (See tests C46012A..Z.)

The method used for rounding to longest integer is apparently round to even. (See tests C46012A..Z.)

The method used for rounding to integer in static universal real expressions is apparently round to even. (See test C4A014A.)

. Array types.

An implementation is allowed to raise NUMERIC_ERROR or CONSTRAINT_ERROR for an array having a 'LENGTH that exceeds STANDARD.INTEGER'LAST and/or SYSTEM.MAX_INT. For this implementation:

Declaration of an array type or subtype declaration with more than SYSTEM.MAX_INT components raises no exception. (See test C36003A.)

No exception is raised when 'LENGTH is applied to an array type with INTEGER'LAST + 2 components. (See test C36202A.)

No exception is raised when 'LENGTH is applied to an array type with SYSTEM.MAX_INT + 2 components. (See test C36202B.)

A packed BOOLEAN array having a 'LENGTH exceeding INTEGER'LAST raises CONSTRAINT_ERROR when the array type is declared. (See test C52103X.)

A packed two-dimensional BOOLEAN array with more than INTEGER'LAST components raises CONSTRAINT_ERROR when the array type is declared. (See test C52104Y.)

A null array with one dimension of length greater than INTEGER'LAST may raise NUMERIC_ERROR or CONSTRAINT_ERROR either when declared or assigned. Alternatively, an implementation may accept the declaration. However, lengths must match in array slice assignments. This implementation raises CONSTRAINT_ERROR when the array type is declared. (See test E52103Y.)

In assigning one-dimensional array types, the expression does not appear to be evaluated in its entirety before CONSTRAINT_ERROR is raised when checking whether the expression's subtype is compatible with the target's subtype. In assigning two-dimensional array types, the expression does not appear to be evaluated in its entirety before CONSTRAINT_ERROR is raised when checking whether the expression's subtype is compatible with the target's subtype. (See test C52013A.)

. Discriminated types.

During compilation, an implementation is allowed to either accept or reject an incomplete type with discriminants that is used in an access type definition with a compatible discriminant constraint. This implementation accepts such subtype indications. (See test E38104A.)

In assigning record types with discriminants, the expression does not appear to be evaluated in its entirety before CONSTRAINT_ERROR is raised when checking whether the expression's subtype is

compatible with the target's subtype. (See test C52013A.)

. Aggregates.

In the evaluation of a multi-dimensional aggregate, all choices appear to be evaluated before checking against the index type. (See tests C43207A and C43207B.)

In the evaluation of an aggregate containing subaggregates, not all choices are evaluated before being checked for identical bounds. (See test E43212B.)

All choices are evaluated before CONSTRAINT_ERROR is raised if a bound in a nonnull range of a nonnull aggregate does not belong to an index subtype. (See test E43211B.)

. Representation clauses.

An implementation might legitimately place restrictions on representation clauses used by some of the tests. If a representation clause is used by a test in a way that violates a restriction, then the implementation must reject it.

Enumeration representation clauses containing noncontiguous values for enumeration types other than character and boolean types are supported. (See tests C35502I..J, C35502M..N, and A39005F.)

Enumeration representation clauses containing noncontiguous values for character types are supported. (See tests C35507I..J, C35507M..N, and C55B16A.)

Enumeration representation clauses for boolean types containing representational values other than (FALSE => 0, TRUE => 1) are supported. (See tests C35508I..J and C35508M..N.)

Length clauses with SIZE specifications for enumeration types are supported. (See test A39005B.)

Length clauses with STORAGE_SIZE specifications for access types are supported. (See tests A39005C and C87B62B.)

Length clauses with STORAGE_SIZE specifications for task types are supported. (See tests A39005D and C87B62D.)

Length clauses with SMALL specifications are supported. (See tests A39005E and C87B62C.)

There are restrictions for alignment clauses within record representation clauses. (See test A39005G.)

Length clauses with SIZE specifications for derived integer types are supported. (See test C87B62A.)

. Pragmas.

The pragma INLINE is supported for procedures. The pragma INLINE is supported for functions. (See tests LA3004A, LA3004B, EA3004C, EA3004D, CA3004E, and CA3004F.)

. Input/output.

The package SEQUENTIAL_IO can be instantiated with unconstrained array types and record types with discriminants without defaults. (See tests AE2101C, EE2201D, and EE2201E.)

The package DIRECT_IO can be instantiated with unconstrained array types and record types with discriminants without defaults. (See tests AE2101H, EE2401D, and EE2401G.)

The director, AJPO, has determined (AI-00332) that every call to OPEN and CREATE must raise USE_ERROR or NAME_ERROR if file input/output is not supported. This implementation exhibits this behavior for SEQUENTIAL_IO, DIRECT_IO, and TEXT_IO.

. Generics.

Generic subprogram declarations and bodies can be compiled in separate compilations. (See tests CA1012A and CA2009F.)

Generic package declarations and bodies can be compiled in separate compilations. (See tests CA2009C, BC3204C, and BC3205D.)

Generic unit bodies and their subunits can be compiled in separate compilations. (See test CA3011A.)

CHAPTER 3

TEST INFORMATION

3.1 TEST RESULTS

Version 1.9 of the ACVC comprises 3122 tests. When this compiler was tested, 27 tests had been withdrawn because of test errors. The AVF determined that 375 tests were inapplicable to this implementation. All inapplicable tests were processed during validation testing except for 159 executable tests that use floating-point precision exceeding that supported by the implementation and 174 executable tests that use file operations not supported by the implementation. Modifications to the code, processing, or grading for 13 tests were required to successfully demonstrate the test objective. (See section 3.6.)

The AVF concludes that the testing results demonstrate acceptable conformity to the Ada Standard.

3.2 SUMMARY OF TEST RESULTS BY CLASS

RESULT	TEST CLASS						TOTAL
	A	B	C	D	E	L	
Passed	108	1048	1484	17	17	46	2720
Inapplicable	2	3	369	0	1	0	375
Withdrawn	3	2	21	0	1	0	27
TOTAL	113	1053	1874	17	19	46	3122

3.3 SUMMARY OF TEST RESULTS BY CHAPTER

RESULT	CHAPTER													TOTAL
	2	3	4	5	6	7	8	9	10	11	12	13	14	
Passed	182	513	560	245	166	98	141	326	137	36	234	3	79	2720
Inapplicable	22	59	114	3	0	0	2	1	0	0	0	0	174	375
Withdrawn	2	14	3	0	0	1	2	0	0	0	2	1	2	27
TOTAL	206	586	677	248	166	99	145	327	137	36	236	4	255	3122

3.4 WITHDRAWN TESTS

The following 27 tests were withdrawn from ACVC Version 1.9 at the time of this validation:

B28003A	E28005C	C34004A	C35502P	A35902C
C35904A	C35904B	C35A03E	C35A03R	C37213H
C37213J	C37215C	C37215E	C37215G	C37215H
C38102C	C41402A	C45332A	C45614C	A74106C
C85018B	C87B04B	CC1311B	BC3105A	AD1A01A
CE2401H	CE3208A			

See Appendix D for the reason that each of these tests was withdrawn.

3.5 INAPPLICABLE TESTS

Some tests do not apply to all compilers because they make use of features that a compiler is not required by the Ada Standard to support. Others may depend on the result of another test that is either inapplicable or withdrawn. The applicability of a test to an implementation is considered each time a validation is attempted. A test that is inapplicable for one validation attempt is not necessarily inapplicable for a subsequent attempt. For this validation attempt, 375 tests were inapplicable for the reasons indicated:

- Tests C24113D..N (11 tests) contain lines of lengths greater than 80 characters which is not supported by this compiler.
- A39005G uses a record representation clause which is not supported by this compiler.

. C34007P, C34007S are expected to raise CONSTRAINT_ERROR. This implementation optimizes the code at compile time on lines 201 and 217 respectively, thus avoiding the operation which would raise CONSTRAINT_ERROR and so no exception is raised. The AVO ruled this behaviour acceptable and the test NA.

. C41401A is expected to raise CONSTRAINT_ERROR for the evaluation of certain attributes, however this implementation derives the values from the subtype of the prefix at compile time, as allowed by 11.6(7) LRM. Therefore elaboration of the prefix is not involved and CONSTRAINT_ERROR is not raised. The AVO ruled this behaviour acceptable and the test NA.

. The following tests use LONG_INTEGER, which is not supported by this compiler:

C45231C	C45304C	C45502C	C45503C	C45504C
C45504F	C45611C	C45613C	C45614C	C45631C
C45632C	B52004D	C55B07A	B55B09C	

. C45231D requires a macro substitution for any predefined numeric types other than INTEGER, SHORT_INTEGER, LONG_INTEGER, FLOAT, SHORT_FLOAT, and LONG_FLOAT. This compiler does not support any such types.

. C45531M, C45531N, C45532M, and C45532N use fine 48-bit fixed-point base types which are not supported by this compiler.

. C45531O, C45531P, C45532O, and C45532P use coarse 48-bit fixed-point base types which are not supported by this compiler.

. C47004A is expected to raise CONSTRAINT_ERROR whilst evaluating the comparison on line 51, but this compiler evaluates the result without invoking the basic operation qualification (as allowed by 11.6(7) LRM) which would raise CONSTRAINT_ERROR and so no exception is raised. The AVO ruled this behaviour acceptable and the test NA.

. B86001D requires a predefined numeric type other than those defined by the Ada language in package STANDARD. There is no such type for this implementation.

. C86001F redefines package SYSTEM, but the body of the REPORT package is made obsolete by this new definition in this implementation because it depends on an IO package which uses SYSTEM. Therfore, the test cannot be executed.

. C96005B requires the range of type DURATION to be different from those of its base type; in this implementation they are the same.

. The following 174 tests are inapplicable because sequential, text, and direct access files are not supported.

CE2102C	CE2102G..H(2)	CE2102K	CE2104A..D(4)
CE2105A..B(2)	CE2106A..B(2)	CE2107A..I(9)	CE2108A..D(4)
CE2109A..C(3)	CE2110A..C(3)	CE2111A..E(5)	CE2111G..H(2)
CE2115A..B(2)	CE2201A..C(3)	CE2201F..G(2)	CE2204A..B(2)
CE2208B	CE2210A	CE2401A..C(3)	CE2401E..F(2)
CE2404A	CE2405B	CE2406A	CE2407A
CE2408A	CE2409A	CE2410A	CE2411A
AE3101A	CE3102B	EE3102C	CE3103A
CE3104A	CE3107A	CE3108A..B(2)	CE3109A
CE3110A	CE3111A..E(5)	CE3112A..B(2)	CE3114A..B(2)
CE3115A	CE3203A	CE3301A..C(3)	CE3302A
CE3305A	CE3402A..D(4)	CE3403A..C(3)	CE3403E..F(2)
CE3404A..C(3)	CE3405A..D(4)	CE3406A..D(4)	CE3407A..C(3)
CE3408A..C(3)	CE3409A	CE3409C..F(4)	CE3410A
CE3410C..F(4)	CE3411A	CE3412A	CE3413A
CE3413C	CE3602A..D(4)	CE3603A	CE3604A
CE3605A..E(5)	CE3606A..B(2)	CE3704A..B(2)	CE3704D..F(3)
CE3704M..O(3)	CE3706D	CE3706F	CE3804A..E(5)
CE3804G	CE3804I	CE3804K	CE3804M
CE3805A..B(2)	CE3806A	CE3806D..E(2)	CE3905A..C(3)
CE3905L	CE3906A..C(3)	CE3906E..F(2)	

Results of running a subset of these tests showed that the proper exceptions are raised for unsupported file operations.

. The following 159 tests require a floating-point accuracy that exceeds the maximum of 18 digits supported by this implementation:

C241130..Y (11 tests)	C357050..Y (11 tests)
C357060..Y (11 tests)	C357070..Y (11 tests)
C357080..Y (11 tests)	C358020..Z (12 tests)
C452410..Y (11 tests)	C453210..Y (11 tests)
C454210..Y (11 tests)	C455210..Z (12 tests)
C455240..Z (12 tests)	C456210..Z (12 tests)
C456410..Y (11 tests)	C460120..Z (12 tests)

3.6 TEST, PROCESSING, AND EVALUATION MODIFICATIONS

It is expected that some tests will require modifications of code, processing, or evaluation in order to compensate for legitimate implementation behavior. Modifications are made by the AVF in cases where legitimate implementation behavior prevents the successful completion of an (otherwise) applicable test. Examples of such modifications include: adding a length clause to alter the default size of a collection; splitting a Class B test into subtests so that all errors are detected; and confirming that messages produced by an executable test demonstrate conforming behavior that wasn't anticipated by the test (such as raising one exception instead of another).

The body of the supporting package REPORT was changed to use the simple IO package MINI_IO instead of TEXT_IO. MINI_IO contains a procedure which allows the transfer of strings from the target computer to the MPR 1300 computer.

The procedure TEST within the package REPORT was also changed to produce an IABG specific message displaying the time and date of execution of the test.

Modifications were required for 13 Class B tests.

The following Class B tests were split because syntax errors at one point resulted in the compiler not detecting other errors in the test:

B22003A	B24009A	B29001A	B38003A	B38009A
B38009B	B51001A	B91001H	BC2001D	BC2001E
BC3204B	BC3205B	BC3205D		

Test C45651A is ruled as PASSED although it produced the result FAILED and produced the following messages arising from the lines 227 and 252

"ABS 928.0 NOT IN CORRECT RANGE" and
"ABS (-928.0) NOT IN CORRECT RANGE"

The test requires the results of the disputed expressions to fall within too narrow a range. However, the test includes many other checks on fixed-point types, and the implementation is credited with passing these checks.

3.7 ADDITIONAL TESTING INFORMATION

3.7.1 Prevalidation

Prior to validation, a set of test results for ACVC Version 1.9 produced by the KRUPP ATLAS ELEKTRONIK GMBH Ada Compiler VVME 1.8 A was submitted to the AVF by the applicant for review. Analysis of these results demonstrated that the compiler successfully passed all applicable tests, and the compiler exhibited the expected behavior on all inapplicable tests.

3.7.2 Test Method

Testing of the KRUPP ATLAS ELEKTRONIK GMBH Ada Compiler VVME 1.8 A using ACVC Version 1.9 was conducted on-site by a validation team from the AVF. The configuration consisted of a VAX 11/785 operating under VMS, version 4.5 for compilation of Ada programs into assembly code, a MPP 1300 operating under MOS 1.213, for assembling and linking the intermediate assembly code programs, and a MPR 2300 target, operating under EOS version

1.1. Files were transferred between the VAX and the MPR 1300 via magnetic tape. Files were transferred between the MPR 1300 and the MPR 2300 via a HDLC connection for downloading and recapturing the results, and a V24 connection used for system initialization. Results were transferred from the MPR 1300 to a VAX 11/750 via Decnet and the Kermit file transfer protocol.

A Magnetic tape containing all tests except for withdrawn tests and tests requiring unsupported floating-point precisions was taken on-site by the validation team for processing. Tests that make use of implementation-specific values were customized before being written to the Magnetic tape. Tests requiring modifications during the prevalidation testing were included in their modified form on the Magnetic tape.

The contents of the Magnetic tape were loaded directly onto the VAX 11/785 computer.

After the test files were loaded to disk, the full set of tests was translated into assembly code on the VAX 11/785, and all executable tests were transferred to the MPR 1300, in chapter groups, via magnetic tapes where they were assembled and linked. All executable images were then transferred via the HDLC connection to the MPR 2300 where they were executed. Results were transferred via the HDLC connection to the MPR 1300, where they were transferred using the Kermit file transfer protocol to a VAX 11/750, where they were evaluated.

The compiler was tested using command scripts provided by KRUPP ATLAS ELEKTRONIK GMBH and reviewed by the validation team. The compiler was tested using all default option settings except for the listing option which was used for the class B and class L tests as well as for non-applicable tests.

Tests were compiled, linked, and executed (as appropriate) using a single host computer and a single target computer. Test output, compilation listings, and job logs were captured on Magnetic tape and archived at the AVF. The listings examined on-site by the validation team were also archived.

3.7.3 Test Site

Testing was conducted at KRUPP ATLAS ELEKTRONIK GMBH, Bremen and was completed on December 15th, 1988.

APPENDIX A

DECLARATION OF CONFORMANCE

KRUPP ATLAS ELEKTRONIK GMBH has submitted the following Declaration of Conformance concerning the KRUPP ATLAS ELEKTRONIK GMBH Ada Compiler VVME 1.8 A.

DECLARATION OF CONFORMANCE

Compiler Implementor: KRUPP ATLAS ELEKTRONIK GMBH
Ada Validation Facility: IABG MBH
Ada Compiler Validation Capability (ACVC) Version: 1.9

BASE CONFIGURATION

Base Compiler Name:	KRUPP ATLAS ELEKTRONIK Ada Compiler VVME 1.8 A
Base Compiler Version:	1.8 A
Host Architecture ISA:	VAX 11/785 under VMS 4.5 combined with KRUPP ATLAS ELEKTRONIK GMBH MPR 1300 under MOS 1.213
Target Architecture ISA:	KRUPP ATLAS ELEKTRONIK GMBH MPR 2300 under EOS 2300 1.1

IMPLEMENTOR's DECLARATION

We, the undersigned, representing KRUPP ATLAS ELEKTRONIK GMBH have implemented no deliberate extensions to the Ada Language Standard ANSI/MIL-STD-1815A in the compiler listed in this declaration. We declare that KRUPP ATLAS ELEKTRONIK GMBH is the owner of record of the Ada language compiler listed above and, as such, is responsible for maintaining said compiler in conformance to ANSI/MIL-STD-1815A. All certificates and registrations for Ada language compiler listed in this declaration shall be made only in the owner's corporate name.

KRUPP ATLAS ELEKTRONIK GMBH

i.A. - Brötje -

i.V. - Barg -

OWNER's DECLARATION

We, the undersigned, representing KRUPP ATLAS ELEKTRONIK GMBH, take full responsibility for implementation and maintenance of the Ada compiler listed above, and agree to the public disclosure of the final Validation Summary Report. We further agree to continue to comply with the Ada trademark policy, as defined by the Ada Joint Program Office. We declare that all of the Ada language compilers listed, and their host/target performance, are in compliance with the Ada Language Standard ANSI/MIL-STD-1815A.

KRUPP ATLAS ELEKTRONIK GMBH

i.A. - Brötje -

i.V. - Barg -

Bremen, December 16th, 1988

APPENDIX B

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 the KRUPP ATLAS ELEKTRONIK GMBH Ada Compiler VMME 1.8 A, are described in the following sections, which discuss topics in Appendix F of the Ada Standard. Implementation-specific portions of the package STANDARD are also included in this appendix.

5 Predefined Language Environment

The predefined language environment comprises the **PACKAGE standard**, the language-defined library units and the implementation-defined library units.

5.1 The Package STANDARD

The specification of the PACKAGE standard is outlined here; it contains all predefined identifiers of the implementation.

The operations defined for the predefined types are not mentioned here, since they are implicitly declared according to the language rules. Anonymous types (such as universal_integer) are not mentioned either.

```
PACKAGE standard IS

   TYPE boolean IS (false, true);

   TYPE short_integer IS RANGE - 32_768 .. 32_767;

   TYPE integer IS RANGE - 2_147_483_648 .. 2_147_483_647;

   TYPE short_float IS DIGITS 6 RANGE
      - 16#0.7FFF_FF8#E+32 .. 16#0.7FFF_FF8#E+32;
   -- the corresponding machine type is F-FLOAT

   TYPE float IS DIGITS 9 RANGE
      - 16#0.7FFF_FFFF_FFFF_FF8#E+32 ..
        16#0.7FFF_FFFF_FFFF_FF8#E+32;
   -- the corresponding machine type is D-FLOAT

   TYPE long_float IS DIGITS 15 RANGE
      - 16#0.7FFF_FFFF_FFFF_FC#E+256 ..
        16#0.7FFF_FFFF_FFFF_FC#E+256;
   -- the corresponding machine type is G-FLOAT

-- TYPE character IS ... as in [Ada,Appendix C]

-- FOR character USE ... as in [Ada,Appendix C]

-- PACKAGE ascii IS ... as in [Ada,Appendix C]

-- Predefined subtypes and string types as in [Ada,Appendix C]

   TYPE duration IS DELTA 2#1.0#E-7 RANGE
      - 16_777_216.0 .. 16_777_216.0;

-- The predefined exceptions are as in (Ada,Appendix C)

END standard;
```

7.2 Length Clauses

SIZE

for all integer, fixed point and enumeration types the value must be <= 32; for short_float types the value must be = 32 (this is the amount of storage which is associated with these types anyway);
for float and long_float types the value must be = 64 (this is the amount of storage which is associated with these types anyway).
for access types the value must be = 32 (this is the amount of storage which is associated with these types anyway).
If any of the above restrictions are violated, the compiler responds with a RESTRICTION error message in the compiler listing.

STORAGE_SIZE

Collection size: If no length clause is given, the storage spaceneeded to contain objects designated by values of the access type and by values of other types derived from it is extended dynamically at runtime as needed. If, on the other hand, a length clause is given, the number of storage units stipulated in the length clause is reserved, and no dynamic extension at runtime occurs.

Storage for tasks: The memory space reserved for a task is 4K bytes if no length clause is given (cf. Chapter 6). If the task is to be allotted either more or less space, a length clause must be given for its task type, and then all tasks of this type will be allotted the amount of space stipulated in the length clause (maximum length is 256 kByte for each task virtuell.)

SMALL

there is no implementation-dependent restriction. Any specification for SMALL that is allowed by the LRM can be given. In particular those values for SMALL are also supported which are not a power of two.

7.3 Enumeration Representation Clauses

The integer codes specified for the enumeration type have to lie inside the range of the largest integer type which is supported; that is the type integer defined in **PACKAGE standard**.

7.4 Record Representation Clauses

Record representation clauses are supported. The value of the expression given in an alignment clause must be 0, 1, 2 or 4. If this restriction is violated, the compiler responds with a RESTRICTION error message in the compiler listing. If the value is 0 the objects of the corresponding record type will not be aligned, if it is 1 (resp. 2 or 4) the starting address of an object will be a multiple of 1 (resp. 2 or 4) * storage unit size.

The number of bits specified by the range of a component clause must not be greater than the amount of storage occupied by this component. (Gaps between components can be forced by leaving some bits unused but not by specifiing a bigger range than needed.) Violation of this restriction will produce a RESTRICTION error message.

There are implementation-dependent components generated to hold the size of the record object if the corresponding record type includes variant parts or to hold the offset of a record component (relative to this generated component) if the size of the record component is dynamic. But there are no implementation-generated names (cf. [Ada,$13.4(8)]) denoting these components. So the mapping of these components cannot be influenced by a representation clause.

7.5 Address Clauses

Address clauses are supported for objects declared by an object declaration. If an address clause is given for a subprogram, package, task unit or single entry, the compiler responds with a RESTRICTION error message in the compiler listing.

9 Appendix F

This is the Appendix F required in [Ada], in which all implementation-dependent characteristics of an Ada implementation are described.

9.1 Implementation-Dependent Pragmas

The form, allowed places, and effect of every implementation-dependent pragma is stated in this section.

9.1.1 Predefined Language Pragmas

The form and allowed places of the following pragmas are defined by the language; their effect is (at least partly) implementation-dependent and stated here. All the other pragmas listed in Appendix B of [Ada] are implemented and have the effect described there.

CONTROLLED

has no effect.

INLINE

Inline expansion of subprograms is supported with following restrictions: the subprogram must not contain declarations of other subprograms, tasks, generic units or body stubs. If the subprogram is called recursively only the outer call of this subprogram will be expanded.

INTERFACE

is supported for the languages Assembler and Meta. For each Ada subprogram for which

PRAGMA interface (<language> , <ada_name>);

is specified, a routine implementing the body of the subprogram <ada_name> must be provided, written in the specified language. The name of the routine, which implements the subprogram <ada_name>, should be specified using the **PRAGMA EXTERNAL_NAME**, otherwise the compiler will generate an enternal name that leads to an unsolved reference during linking.

The subprogram <ada_name> specified in the PRAGMA INTERFACE may be a function or a procedure. For the interface (META,...) there are following conventions. Far parameter passing the programmer has to specify a record in Ada with USE- and AT-clauses. The META-program takes a pointer to the physical structure of the record as its one and only parameter. There are also restrictions in the use of the META-language.

- META-Taskink actions (e.g. META-parallel) are not allowed.
- The User-semaphores from 0 to 1023 are Used for the runtime-system and so far not available for the programmer. A use of them may lead to an erronous program.
- The use of the Space-Monitor (SPAMON) of the MOS is restricted, too. For Ada-Dataspace the first segmented and the first not segmented subspace available through SPAMON are already used.

Example:

Ada:

```
..
TYPE parameter IS
  RECORD
     length : natural;
     addr   : system.address;
  END RECORD;


FOR parameter USE
 RECORD AT MOD 4;
    length AT 0 RANGE 0 .. 31;
    addr   AT 4 RANGE 0 .. 31;
 END RECORD;


PROCEDURE put_line ( x : IN parameter );
PRAGMA INTERFACE (META, put_line);
PRAGMA EXTERNAL NAME ("P_L", put_line);
..
```

META:

```
PROC: P_L (PB POINTER_TO_PARAMETER);

TYPE PARAMETER STRUCT /
               LENGTH  LONG;
               ADDRESS POINTER;
```

For the interface (ASSEMBLER,...) the parameter passing is similiar to interface (META,...). You specify a record in Ada with the correct structure of the parameters. You can reference them in the Assemblerprogram via the A1-register, which is known as parameter. The following conventions have to be obayed by the assemblerprogrammer:

- First of all save all used registers by a MOVEM.L instruction onto the userstack.
- Take care of the stackpointer.
- Reference parameters via A1.
- In teh end restore all saved registers.
- Return to the calling routine with an RTS-instruction.

Example:

Ada:

```
..
TYPE mos_time IS
      RECORD
        year        : integer;
        month       : integer;
        day         : integer;
        seconds     : integer;
      END RECORD;


FOR mos_time USE
       RECORD
         year       AT 0*4 RANGE 0 .. 31;
         month      AT 1*4 RANGE 0 .. 31;
         day        AT 2*4 RANGE 0 .. 31;
         seconds    AT 3*4 RANGE 0 .. 31;
       END RECORD;


FOR mos_time'size USE 16*system.storage_unit;


FUNCTION mos_clock RETURN mos_time;
  PRAGMA INTERFACE (Assembler, mos_clock);
  PRAGMA EXTERNAL_NAME ("_TSKCLCK", mos_clock);
..
```

Assembler:

```
          XDEF      _TSKCLCK
_TSKCLCK  EQU       *

* Save registers

          MOVEM.L   A0-A7/D0-D7,-(A7)

* Take Time from MOS

          MOVE.L    A1,-(A7)
          MOVEQ.L   #5, D0
          TRAP      #6
          MOVEA.L   (A7)+,A1

* Parameters via A1-registers

          BFEXTU    D1{16:7},D0
          ADD.L     #1900,D0
          MOVE.L    D0,([A1],0)
          BFEXTU    D1{23:4},D0
          MOVE.L    D0,([A1],4)
          BFEXTU    D1{27:5},D0
          MOVE.L    D0,([A1],8)
          TMULS.L   #DU_SMALL,D2
          ADD.L     #99,D2
```

```
                MOVE.L   D2,([A1],12)

        * restore registers

                MOVEM.L  (A7)+,A0-A7/D0-D7
                RTS
```

The KRUPP ATLAS ELEKTRONIK Ada Compiler does not provide checking the observance of the procedure calling standard. If it is violated the call of the system routine will be erroneous.

MEMORY_SIZE

has no effect.

OPTIMIZE

has no effect.

PACK

see User Manual

PRIORITY

There are two implementation-defined aspects of this pragma: First, the range of the subtype **PRIORITY**, and second, the effect on scheduling of not giving this pragma for a task or main program. The range of subtype **PRIORITY** is 0 .. 63 as declared in the predefined library **PACKAGE system** and the effect on scheduling of leaving the priority of a task or main program undefined by not giving **PRAGMA priority** for it is the same as if **PRAGMA priority** 63 had been given (i.e. the task has the highest priority). Moreover, in this implementation the **PACKAGE system** must be named by a with clause of a compilation unit if the predefined **PRAGMA priority** is used within that unit.

SHARED

is supported.

STORAGE_UNIT

has no effect.

SUPPRESS

has no effect, but see for the implementation-defined PRAGMA suppress_all.

SYSTEM_NAME

has no effect.

9.1.2 Implementation-Defined Pragmas

SQUEEZE

is supported

SUPPRESS_ALL

causes all the run_time checks except ELEBORATION_CHEK to be suppressed; this pragma is only allowed at the start of a compilation before the first compilation unit; it applies to the whole compilation.

EXTERNAL_NAME (<string>, <ada_name>)

<ada_name> specifies the name of a subprogram, <string> must be a string literal. The string has a maximum length of 8 charcters. It denotes the external name that the compiler place with the entry point of the specified sub-program. The suprogram declaration of <ada_name> must precede this pragma. If several subprograms with the same name satisfy this requirement the pragma refers to that subprogram which preceds immediately. This pragma will be used in connection with **PRAGMA interface (Meta)** or **interface (Assembler)**

RESIDENT (<ada_name>)

this pragma causes that no assigment of a value to the object <ada_name> will be eliminated by the optimizer of the **KRUPP ATLAS ELEKTRONIK Ada Compiler**. The following code sequence demonstrates the intended usage of the pragma:

```
..
   x : integer;
   a : SYSTEM.address;
   PROCEDURE do_something (a : SYSTEM.address);

   ..
   BEGIN
     x := 5;
     a := x'ADDRESS;
     do_something (a);  -- a.ALL will be read in the body
                        -- of do_something
     x := 6;
..
```

If this code sequence is compiled by the **KRUPP ATLAS ELEKTRONIK Ada Compiler** with the option

```
OPTIMIZER=>ON
```

the statement x := 5; will be eliminated because from the point of view of the optimizer the value of x is not used before the next assignment to x. Therefore

```
PRAGMA resident (x);
```

should be inserted after the declaration of x. This pragma can be applied to all those kinds of objects for which the address clause is supported

9.2 Implementation-Dependent Attributes

The name, type and implementation-dependent aspects of every implementation-dependent attribute is stated in this chapter.

9.2.1 Language-Defined Attributes

The name and type of all the language-defined attributes are as given in [Ada]. We note here only the implementation-dependent aspects.

ADDRESS

The value delivered by this attribute applied to an object is the address of the storage unit where this object starts. For any other entity this attribute is not supported and will return the value system.address_zero.

STORAGE_SIZE

The value delivered by this attribute applied to an access type is as follows: If a length specification (STORAGE_SIZE) has been given for that type (static collection), the attribute delivers that specified value. In case of a dynamic collection, i.e. no length specification by STORAGE_SIZE given for the access type, the attribute delivers the number of storage units currently allocated for the collection. Note that dynamic collections are extended if needed. If the collection manager is used for a dynamic collection the attribute delivers the number of storage units currently allocated for the collection. Note that in this case the number of storage units currently allocated may be decreased by release operations.
The value delivered by this attribute applied to a task type or task object is as follows:
If a length specification (STORAGE_SIZE) has been given for the task type, the attribute delivers that specified value; otherwise, the default value is returned elsewhere.

9.2.2 Implementation-Defined Attributes

There are no implementation-defined attributes.

9.3 Specification of the Package SYSTEM

The PACKAGE system of [Ada,$13.7)) is reprinted here with all implementation-dependent characteristics and extensions filled in.

```
PACKAGE system IS
   TYPE designated_by_address IS LIMITED PRIVATE;
   TYPE address IS ACCESS designated_by_address;
   FOR address'size USE 32;
   For address'storage_size USE 0;
    -- Logically, the type address is defined by:
    --   TYPE Address IS PRIVATE;
    -- However, in this case no representation specification
    -- can be given for record components of type system.address.
    -- The storage size specification assures that any attempt
    -- to create an address value with an allocator raises
    -- STORAGE ERROR
   address_zero : CONSTANT address := NULL;
   FUNCTION "+" (left : address; right : integer) RETURN address;
      PRAGMA _built_in (address_plus_integer, "+");
   FUNCTION "+" (left : integer; right : address) RETURN address;
      PRAGMA _built_in (integer_plus_address, "+");
   FUNCTION "-" (left : address; right : integer) RETURN address;
      PRAGMA _built_in (address_minus_integer, "-");
   FUNCTION "-" (left : address; right : address) RETURN integer;
      PRAGMA _built_in (address_minus_address, "-");
   SUBTYPE external_address IS STRING;
    -- External addresses use hexadecimal notation with characters
    -- '0'..'9', 'a'..'f' and 'A'..'F'. For instance:
    --    "7FFFFFFF"
    --    "80000000"
    --    "8" represents the same address as "00000008"
   FUNCTION convert_address (addr:external_address) RETURN
            address;
    -- convert_address raises CONSTRAINT_ERROR if the external
            address
      -- addr is the empty string, contains characters other than
      -- '0'..'9', 'a'..'f', 'A'..'F' or if the resulting address
      -- value cannot be represented with 32 bits.

  FUNCTION convert_address (addr : address) RETURN
            external_address;
    -- The resulting external address consists of exactly 8
            characters
      -- '0'..'9', 'A'..'F'.

   TYPE name IS (motorola_68020_kae);
   system_name  : CONSTANT name := motorola_68020_kae;
   storage_unit : CONSTANT := 8;
   memory_size  : CONSTANT := 2 ** 31;
   min_int      : CONSTANT := - 2 ** 31;
   max_int      : CONSTANT := 2 ** 31 - 1;
   max_digits   : CONSTANT := 18;
   max_mantissa : CONSTANT := 31;
   fine_delta   : CONSTANT := 2.0 ** (-31);
   tick         : CONSTANT := 0.01;
   SUBTYPE priority IS integer RANGE 0 .. 63;

   non_ada_error : EXCEPTION RENAMES _no_ada_error;
    -- non_ada_error is raised if some event occurs which does not
    -- correspond to any situation covered by Ada, e.g.:
    --    illegal instruction encountered
    --    error during address translation
    --    illegal address
```

```
   TYPE exception_id IS NEW integer;

   no_exception_id      : CONSTANT exception_id := 0;
    -- Coding of the predefined exceptions:
   constraint_error_id : CONSTANT exception_id :=  16#0002_0000;
   numeric_error_id    : CONSTANT exception_id :=  16#0002_0001;
   program_error_id    : CONSTANT exception_id :=  16#0002_0002;
   storage_error_id    : CONSTANT exception_id :=  16#0002_0003;
   tasking_error_id    : CONSTANT exception_id :=  16#0002_0004;
   non_ada_error_id    : CONSTANT exception_id :=  16#0002_0005;
   status_error_id     : CONSTANT exception_id :=  16#0002_0006;
   mode_error_id       : CONSTANT exception_id :=  16#0002_0007;
   name_error_id       : CONSTANT exception_id :=  16#0002_0008;
   use_error_id        : CONSTANT exception_id :=  16#0002_0009;
   device_error_id     : CONSTANT exception_id :=  16#0002_000A;
   end_error_id        : CONSTANT exception_id :=  16#0002_000B;
   data_error_id       : CONSTANT exception_id :=  16#0002_000C;
   layout_error_id     : CONSTANT exception_id :=  16#0002_000D;
   time_error_id       : CONSTANT exception_id :=  16#0002_000E;
PRIVATE
   TYPE designated_by_address IS NEW integer;
END system;
```

9.4 Restrictions on Representation Clauses

See $$7.2-7.5 of this manual.

9.5 Conventions for Implementation-Generated Names

There are no implementation-generated names denoting implementation-dependent components [Ada,$13.4].

9.6 Expressions in Address Clauses

Address clauses [Ada,$13.5] are supported for objects except for task objects. The object starts at the given address.

9.7 Restrictions on Unchecked Conversions

9.8 Characteristics of the Input-Output Packages

The implementation-dependent characteristics of the input-output packages as defined in Chapter 14 of [Ada] are reported in Chapter 8 of this manual.

10 References

[Ada] The Programming Language Ada Reference Manual,
American National Standards Institute, Inc.
ANSI/MIL-STD-1815A-1983, Springer Lecture Notes in
Computer Science 155, 1983)

[ST16/85] Jansohn, Hans-Stephan
SYSTEAM Ada System, Cross Reference Generator User
Manual for VAX/VMS,
SYSTEAM Document No. 16/85, 1986)

[ST16/87] P. Dencker
SYSTEAM Ada System, Debugger User Manual for VAX/VMS,
SYSTEAM Document No. 16/87, 1987)

[ST19/84] Zimmermann, Erich
SYSTEAM Ada System, Installation Manual for VAX/VMS,
SYSTEAM Document No. 19/84, 1984)

[ST21/84] Lindenmeyer, Wolf-Dieter
SYSTEAM Ada System, Source Generator User Manual for
VAX/VMS,
SYSTEAM Document No. 21/84, 1986)

[ST22/84] Lindenmeyer, Wolf-Dieter
SYSTEAM Ada System, External-Diana Generator User Manual
for VAX/VMS,
SYSTEAM Document No. 22/84, 1986)

[ST27/84] Lindenmeyer, Wolf-Dieter
SYSTEAM Ada System, Pretty Printer User Manual for
VAX/VMS,
SYSTEAM Document No. 27/84, 1986)

[ST30/84] Lindenmeyer, Wolf-Dieter
SYSTEAM Ada System, Syntax Checker User Manual for
VAX/VMS,
SYSTEAM Document No. 30/84, 1986)

[ST33/84] Lindenmeyer, Wolf-Dieter,
SYSTEAM Ada System, NonInit User Manual for VAX/VMS,
SYSTEAM Document No. 33/84, 1986)

[ST37/88] Martin Schoch
SYSTEAM Ada System, Execution Time Profiler User Manual
SYSTEAM Document No. 37/88, 1988)

[ST4/84] Lindenmeyer, Wolf-Dieter; Schmidt, Diana;
Dausmann, Manfred
SYSTEAM Ada System, Library User System User Manual for
VAX/VMS,
SYSTEAM Document No. 4/84, 1988)

APPENDIX C

TEST PARAMETERS

Certain tests in the ACVC make use of implementation-dependent values, such as the maximum length of an input line and invalid file names. A test that makes use of such values is identified by the extension .TST in its file name. Actual values to be substituted are represented by names that begin with a dollar sign. A value must be substituted for each of these names before the test is run. The values used for this validation are given below.

The use of the '*' operator signifies a multiplication of the following character. The use of the '&' operator signifies concatenation of the preceeding and following strings. The values within quotes are to highlight character or multi character values.

Name and Meaning	Value
$BIG_ID1 Identifier the size of the maximum input line length with varying last character.	79 * 'A' & '1'
$BIG_ID2 Identifier the size of the maximum input line length with varying last character.	79 * 'A' & '2'
$BIG_ID3 Identifier the size of the maximum input line length with varying middle character.	40 * 'A' & '3' & 39 * 'A'
$BIG_ID4 Identifier the size of the maximum input line length with varying middle character.	40 * 'A' & '4' & 39 * 'A'

TEST PARAMETERS

Name and Meaning	Value
$BIG_INT_LIT An integer literal of value 298 with enough leading zeroes so that it is the size of the maximum line length.	77 * '0' & '298'
$BIG_REAL_LIT A universal real literal of value 690.0 with enough leading zeroes to be the size of the maximum line length.	'75' * '0' & '690.0'
$BIG_STRING1 A string literal which when catenated with BIG_STRING2 yields the image of BIG_ID1.	'"' & 40 * 'A' & '"'
$BIG_STRING2 A string literal which when catenated to the end of BIG_STRING1 yields the image of BIG_ID1.	'"' & 39 * 'A' & '1' & '"'
$BLANKS A sequence of blanks twenty characters less than the size of the maximum line length.	60 * ' '
$COUNT_LAST A universal integer literal whose value is TEXT_IO.COUNT'LAST.	2147483647
$FIELD_LAST A universal integer literal whose value is TEXT_IO.FIELD'LAST.	255
$FILE_NAME_WITH_BAD_CHARS An external file name that either contains invalid characters or is too long.	MUCH_TOO_LONG_XY
$FILE_NAME_WITH_WILD_CARD_CHAR An external file name that either contains a wild card character or is too long.	MUCH_TOO_LONG_XY

Name and Meaning	Value
$GREATER_THAN_DURATION A universal real literal that lies between DURATION'BASE'LAST and DURATION'LAST or any value in the range of DURATION.	75000.0
$GREATER_THAN_DURATION_BASE_LAST A universal real literal that is greater than DURATION'BASE'LAST.	16777217.0
$ILLEGAL_EXTERNAL_FILE_NAME1 An external file name which contains invalid characters.	MUCH_TOO_LONG_XY
$ILLEGAL_EXTERNAL_FILE_NAME2 An external file name which is too long.	MUCH_TOO_LONG_XY
$INTEGER_FIRST A universal integer literal whose value is INTEGER'FIRST.	-2147483648
$INTEGER_LAST A universal integer literal whose value is INTEGER'LAST.	2147483647
$INTEGER_LAST_PLUS_1 A universal integer literal whose value is INTEGER'LAST + 1.	2147483648
$LESS_THAN_DURATION A universal real literal that lies between DURATION'BASE'FIRST and DURATION'FIRST or any value in the range of DURATION.	-75000.0
$LESS_THAN_DURATION_BASE_FIRST A universal real literal that is less than DURATION'BASE'FIRST.	-16777217.0
$MAX_DIGITS Maximum digits supported for floating-point types.	18
$MAX_IN_LEN Maximum input line length permitted by the implementation.	80

Name and Meaning	Value
$MAX_INT A universal integer literal whose value is SYSTEM.MAX_INT.	2147483647
$MAX_INT_PLUS_1 A universal integer literal whose value is SYSTEM.MAX_INT+1.	2147483648
$MAX_LEN_INT_BASED_LITERAL A universal integer based literal whose value is 2#11# with enough leading zeroes in the mantissa to be MAX_IN_LEN long.	'2:' & 75 * '0' & '11:'
$MAX_LEN_REAL_BASED_LITERAL A universal real based literal whose value is 16:F.E: with enough leading zeroes in the mantissa to be MAX_IN_LEN long.	'16:' & 73 * '0' & 'F.E:'
$MAX_STRING_LITERAL A string literal of size MAX_IN_LEN, including the quote characters.	'"' & 78 * 'A' & '"'
$MIN_INT A universal integer literal whose value is SYSTEM.MIN_INT.	-2147483648
$NAME A name of a predefined numeric type other than FLOAT, INTEGER, SHORT_FLOAT, SHORT_INTEGER, LONG_FLOAT, or LONG_INTEGER.	NO_SUCH_TYPE
$NEG_BASED_INT A based integer literal whose highest order nonzero bit falls in the sign bit position of the representation for SYSTEM.MAX_INT.	16#FFFFFFFE#

APPENDIX D

WITHDRAWN TESTS

Some tests are withdrawn from the ACVC because they do not conform to the Ada Standard. The following 27 tests had been withdrawn at the time of validation testing for the reasons indicated. A reference of the form "AI-ddddd" is to an Ada Commentary.

- B28003A: A basic declaration (line 36) wrongly follows a later declaration.
- E28005C: This test requires that 'PRAGMA LIST (ON);' not appear in a listing that has been suspended by a previous "pragma LIST (OFF);"; the Ada Standard is not clear on this point, and the matter will be reviewed by the ARG.
- C34004A: The expression in line 168 wrongly yields a value outside of the range of the target type T, raising CONSTRAINT_ERROR.
- C35502P: Equality operators in lines 62 & 69 should be inequality operators.
- A35902C: Line 17's assignment of the nomimal upper bound of a fixed-point type to an object of that type raises CONSTRAINT_ERROR, for that value lies outside of the actual range of the type.
- C35904A: The elaboration of the fixed-point subtype on line 28 wrongly raises CONSTRAINT_ERROR, because its upper bound exceeds that of the type.
- C35904B: The subtype declaration that is expected to raise CONSTRAINT_ERROR when its compatibility is checked against that of various types passed as actual generic parameters, may in fact raise NUMERIC_ERROR or CONSTRAINT_ERROR for reasons not anticipated by the test.

- C35A03E, C35A03R: These tests assume that attribute 'MANTISSA returns 0 when applied to a fixed-point type with a null range, but the Ada Standard doesn't support this assumption.

- C37213H: The subtype declaration of SCONS in line 100 is wrongly expected to raise an exception when elaborated.

- C37213J: The aggregate in line 451 wrongly raises CONSTRAINT_ERROR.

- C37215C, C37215E, C37215G, C37215H: Various discriminant constraints are wrongly expected to be incompatible with type CONS.

- C38102C: The fixed-point conversion on line 23 wrongly raises CONSTRAINT_ERROR.

- C41402A: 'STORAGE_SIZE is wrongly applied to an object of an access type.

- C45332A: The test expects that either an expression in line 52 will raise an exception or else MACHINE_OVERFLOWS is FALSE. However, an implementation may evaluate the expression correctly using a type with a wider range than the base type of the operands, and MACHINE_OVERFLOWS may still be TRUE.

- C45614C: REPORT.IDENT_INT has an argument of the wrong type (LONG_INTEGER).

- A74106C, C85018B, C87B04B, CC1311B: A bound specified in a fixed-point subtype declaration lies outside of that calculated for the base type, raising CONSTRAINT_ERROR. Errors of this sort occur in lines 37 & 59, 142 & 143, 16 & 48, and 252 & 253 of the four tests, respectively (and possibly elsewhere).

- BC3105A: Lines 159..168 are wrongly expected to be illegal; they are legal.

- AD1A01A: The declaration of subtype INT3 raises CONSTRAINT_ERROR for implementations that select INT'SIZE to be 16 or greater.

- CE2401H: The record aggregates in lines 105 & 117 contain the wrong values.

- CE3208A: This test expects that an attempt to open the default output file (after it was closed) with mode IN_FILE raises NAME_ERROR or USE_ERROR; by Commentary AI-00048, MODE_ERROR should be raised.