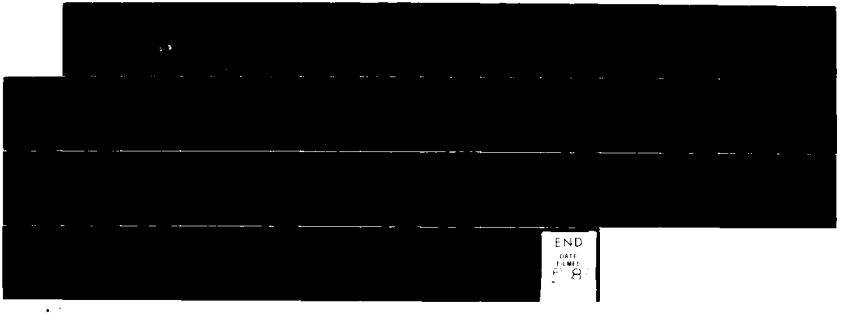
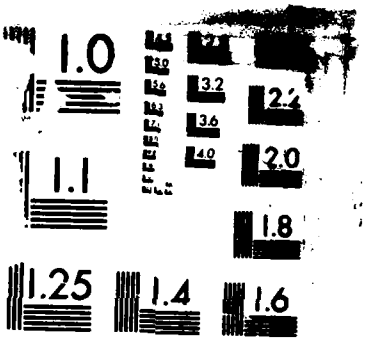
AD-A180 063 ADA (TRADENAME) COMPILER VALIDATION SUMMARY REPORT SYSTEMS DESIGNERS SD V..(U) NATIONAL COMPUTING CENTRE LTD MANCHESTER (ENGLAND) 16 JUN 86 1/1

UNCLASSIFIED F/G 12/5 NL




1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
3.2
3.6
4.0

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

DTIC FILE COPY

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETEING FORM

1. REPORT NUMBER | 2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE *(and Subtitle)* Ada Compiler Validation Summary Report: Systems Designers, SD VAX x Motorola M68000/10 Ada-Plus, 2A.00 Host: VAX 8600 Target: Mc68010	5. TYPE OF REPORT & PERIOD COVERED 16 JUN 1986 to 16 JUN 1987 6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) National Computing Centre Limited	8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION AND ADDRESS National Computing Centre Limited Oxford Road, Manchester M1 7ED United Kingdom	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 16 JUN 1986 13. NUMBER OF PAGES 47
14. MONITORING AGENCY NAME & ADDRESS*(If different from Controlling Office)* National Computing Centre LImited	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

DTIC ELECTE MAY 07 1987 S D E

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)*

See Attached.

AD-A180 063

DD FORM 1473 1 JAN 73 EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-LF-014-6601

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

12

Ada* Compiler Validation Summary Report:

Compiler Name: SD VAX x Motorola M68000/10 Ada-Plus, 2A100

Host Computer:
Digital Equipment VAX 8600
under
VMS
4.2

Target Computer:
MC68010
under
no operating system

Testing Completed 16 June 1986 Using ACVC 1.7

This report has been reviewed and is approved.

The National Computing Centre Ltd
Vony Gwillim
Oxford Road
Manchester
M1 7ED
U K

Ada Validation Office
Dr. John F. Kramer
Institute for Defense Analyses
Alexandria VA
U S A

Ada Joint Program Office
Virginia L. Castor
Director
Department of Defense
Washington DC
U S A

*Ada is a registered trademark of the United States Government (Ada Joint Program Office).

90502/02

Ada* COMPILER
VALIDATION SUMMARY REPORT:
Systems Designers
SD VAX x Motorola M68000/10 Ada-Plus, 2A.00
Host: VAX 8600
Target: MC68010

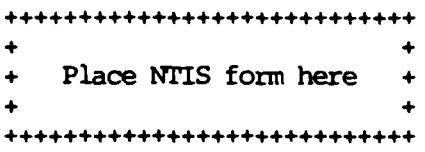
Completion of On-Site Validation:
16 June 1986

Prepared By:
National Computing Centre Limited
Oxford Road
Manchester
M1 7ED
U K

Prepared For:
Ada Joint Program Office
United States Department of Defense
Washington, D.C.

---

*Ada is a registered trademark of the United States Government (Ada Joint Program Office).


Place NTIS form here

EXECUTIVE SUMMARY

This Validation Summary Report (VSR) summarizes the results and conclusions of validation testing performed on the SD VAX x Motorola M68000/10 Ada-Plus,2A.00, using Version 1.7 of the Ada Compiler Validation Capability (ACVC).

The validation process includes submitting a suite of standardized tests (the ACVC) as inputs to an Ada compiler and evaluating the results. The purpose is to ensure conformance of the compiler to ANSI/MIL-STD-1815A Ada by testing that it properly implements legal language constructs and that it identifies and rejects illegal language constructs. The testing also identifies behaviour that is implementation dependent but permitted by ANSI/MIL-STD-1815A. Six classes of tests are used. These tests are designed to perform checks at compile time, at link time, or during execution.

On-site testing was performed 13 June 1986 through 16 June 1986 at Systems Designers, Camberley under the auspices of the National Computing Centre (AVF), according to Ada Validation Organization (AVO) policies and procedures. The SD VAX x Motorola M68000/10 Ada-Plus, 2A.00, is hosted on Digital Equipment VAX 8600 operating under VMS 4.2.

The results of validation are summarized in the following table:

RESULT	TEST CLASS						TOTAL
	A	B	C	D	E	L	
Passed	68	824	984	17	8	21	1922
Failed	0	0	0	0	0	0	0
Inapplicable	0	0	337	0	3	2	342
Withdrawn	0	4	11	0	0	0	15
TOTAL	68	828	1332	17	11	23	2279

Ada is a registered trademark of the United States Government (Ada Joint Program Office).

There were 15 withdrawn tests in ACVC Version 1.7 at the time of this validation attempt. A list of these tests appears in Appendix D.

Some tests demonstrate that some language features are or are not supported by an implementation. For this implementation, the tests determined the following:

- SHORT_INTEGER, LONG_INTEGER, SHORT_FLOAT and LONG_FLOAT are not supported.
- No additional predefined types are supported.
- Representation specifications for noncontiguous enumeration representations are supported.
- The 'SIZE and 'SMALL clauses are supported.
- The 'STORAGE_SIZE clause is supported.
- Generic unit specifications and bodies can be compiled in separate compilations.
- Pragma INLINE is not supported for procedures or functions.
- The package SYSTEM is used by package TEXT_IO.
- Mode IN_FILE and OUT_FILE are not supported for sequential I/O.
- Instantiation of the package SEQUENTIAL_IO with unconstrained array types is supported.
- Instantiation of the package SEQUENTIAL_IO with unconstrained record types with discriminants is supported.
- RESET and DELETE are not supported for sequential and direct I/O.
- Modes IN_FILE and OUT_FILE are not supported for direct I/O.
- Instantiation of package DIRECT_IO with unconstrained array types and unconstrained types with discriminants is supported.
- Dynamic creation and deletion of files is not supported.
- Illegal file names cannot exist.

ACVC Version 1.7 was taken on-site via magnetic tape to Systems Designers, Camberley. All tests, except the withdrawn tests and any executable tests that make use of a floating-point precision greater than SYSTEM.MAX_DIGITS, were compiled on a Digital Equipment VAX 8600. Class A, C, D, and E tests were executed on a MC68010.

On completion of testing, execution results for Class A, C, D, or E tests were examined. Compilation results for Class B were analysed for correct diagnosis of syntax and semantic errors. Compilation and link results of Class L tests were analysed for correct detection of errors.

The AVF identified 1986 of the 2279 tests in Version 1.7 of the ACVC as potentially applicable to the validation of SD VAX x Motorola M68000/10 Ada-Plus, 2A.00. Excluded were 278 tests requiring a floating-point precision greater than that supported by the implementation and the 15 withdrawn tests. After the 1986 tests were processed, 73 tests were determined to be inapplicable. The remaining 1913 tests were passed by the compiler.

The AVF concludes that these results demonstrate conformance to ANSI/MIL-STD-1815A.

TABLE OF CONTENTS

CHAPTER 1 INTRODUCTION

1.1 PURPOSE OF THIS VALIDATION SUMMARY REPORT 1-1
1.2 USE OF THIS VALIDATION SUMMARY REPORT 1-2
1.3 RELATED DOCUMENTS 1-3
1.4 DEFINITION OF TERMS 1-3
1.5 ACVC TEST CLASSES 1-4

CHAPTER 2 CONFIGURATION INFORMATION

2.1 CONFIGURATION TESTED 2-1
2.2 CERTIFICATE INFORMATION 2-2
2.3 IMPLEMENTATION CHARACTERISTICS 2-3

CHAPTER 3 TEST INFORMATION

3.1 TEST RESULTS 3-1
3.2 SUMMARY OF TEST RESULTS BY CLASS 3-1
3.3 SUMMARY OF TEST RESULTS BY CHAPTER 3-2
3.4 WITHDRAWN TESTS 3-2
3.5 INAPPLICABLE TESTS 3-2
3.6 SPLIT TESTS 3-5
3.7 ADDITIONAL TESTING INFORMATION 3-5
3.7.1 Prevalidation 3-5
3.7.2 Test Method 3-5
3.7.3 Test Site 3-6

APPENDIX A COMPLIANCE STATEMENT

APPENDIX B APPENDIX F OF THE Ada STANDARD

APPENDIX C TEST PARAMETERS

APPENDIX D WITHDRAWN TESTS

CHAPTER 1

INTRODUCTION

This Validation Summary Report (VSR) describes the extent to which a specific Ada compiler conforms to 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 (ANSI/MIL-STD-1815A). Any implementation-dependent features must conform to the requirements of the Ada Standard. The entire Ada Standard must be implemented, and nothing can be implemented that is not in the Standard.

Even though all validated Ada compilers conform to ANSI/MIL-STD-1815A, 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 limitations imposed on a compiler by the operating system and by the hardware. All of the dependencies demonstrated during the process of testing this compiler are given in this report.

VSRs are written according to a standardized format. The reports for several different compilers may, therefore, be easily compared. The information in this report is derived from the test results produced during validation testing. Additional testing information as well as details which are unique for this compiler are given in section 3.7. The format of a validation report limits variance between reports, enhances readability of the report and minimizes the delay between completion of validation testing and the publication of the report.

1.1 PURPOSE OF THIS VALIDATION SUMMARY REPORT

The 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 unsupported language constructs required by the Ada Standard.

- To determine that the implementation-dependent behaviour is allowed by the Ada Standard

Testing of this compiler was conducted by NCC under the direction of the AVF according to policies and procedures established by the Ada Validation Organisation (AVO). Testing was conducted from 13 June 1986 through 16 June 1986 at Systems Designers, Camberley.

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 organisations 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 nonconformances to ANSI/MIL-STD-1815A 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
1211 S. Fern, C-107
Washington DC 20301-3081

or from:

Ada Validation Facility
The National Computing Centre Ltd
Oxford Road
Manchester
M1 7ED
U K

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
Alexandria VA 22311

1.3 RELATED DOCUMENTS

1. Reference Manual for the Ada Programming Language, ANSI/MIL-STD-1815A, FEB 1983.

2. Ada Validation Organization: Policies and Procedures, MITRE Corporation, JUN 1982, PB 83-110601.

3. Ada Compiler Validation Capability Implementer's Guide, SofTech, Inc., DEC 1984.

1.4 DEFINITION OF TERMS

ACVC — The Ada Compiler Validation Capability. A set of programs that evaluates the conformance of a compiler to the Ada language specification, ANSI/MIL-STD-1815A.

Ada Standard — ANSI/MIL-STD-1815A, February 1983.

Applicant — The agency requesting validation.

AVF — The National Computing Centre Ltd. In the context of this report, the AVF is responsible for conducting compiler validations according to established policies and procedures.

AVO — The Ada Validation Organization. In the context of this report, the AVO is responsible for setting policies and procedures for compiler validations.

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 — A test for which the compiler generates a result that demonstrates nonconformance to the Ada Standard.

Host	The computer on which the compiler resides.
Inapplicable test	A 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.
LMP	The Ada Board, Language Maintenance Panel whose function is to resolve issues concerning the Ada language.
Passed test	A test for which a compiler generates the expected result.
Target	The computer for which a compiler generates code.
Test	A program that evaluates the conformance of a compiler to a language specification. In the context of this report, the term is used to designate a single ACVC test. The text of a program may be the text of one or more compilations.
Withdrawn test	A test found to be inaccurate in checking conformance to the Ada language specification. A withdrawn test 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

Conformance to ANSI/MIL-STD-1815A is measured using the Ada Compiler Validation Capability (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. Special program units are used to report the results of the Class A, C, D, and E tests during execution. Class B tests are expected to produce compilation errors, and Class L tests are expected to produce link errors.

Class A tests check that legal Ada programs can be successfully compiled and executed. (However, no checks are performed during execution to see if the test objective has been met.) 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 message indicating that it has passed.

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

syntactical 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 capabilities of a compiler. Since there are no requirements placed on a compiler by the Ada Standard for some parameters (e.g., the number of identifiers permitted in a compilation, the number of units in a library, and the number of nested loops in a subprogram body), 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 results. It also provides a set of identity functions used to defeat some compiler optimization strategies and force computations to be made by the target computer instead of by the compiler on the host computer. 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 these units 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.

Some of the conventions followed in the ACVC 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. The values used for this validation are listed in Appendix C.

A compiler must correctly process each of the tests in the suite and demonstrate conformance to the Ada Standard either meeting the pass criteria given for the test or by showing that the test is inapplicable to the implementation. 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 nonconformant tests 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: SD VAX x Motorola M68000/10 Ada-Plus, 2A.00

Test Suite: Ada Compiler Validation Capability, Version 1.7

Host Computer:

Machine(s):	Digital Equipment VAX 8600
Operating System:	VMS 4.2
Memory Size:	20.00 Mb

Target Computer:

Machine(s):	MC68010 implemented on the MVME 117-3FP board
Operating System:	no operating system
Memory Size:	512K bytes RAM

Communications Network: RS232C connector

2.2 CERTIFICATE INFORMATION

Base Configuration:

Compiler: SD VAX x Motorola M68000/10 Ada-Plus, 2A.00

Test Suite: Ada Compiler Validation Capability, Version 1.7

Validation Date: 13 June 1986

Host Computer:

Machine(s): Digital Equipment VAX 8600

Operating System: VMS 4.2

Target Computer:

Machine(s): MC68010 implemented on the MVME 117-3FP board

Operating System: no operating system

Communications Network: RS232C connector

2.3 IMPLEMENTATION CHARACTERISTICS

One of the purposes of validating compilers is to determine the behaviour 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. This compiler is characterized by the following interpretations of the Ada Standard:

- Nongraphic characters.

  Nongraphic characters are defined in the ASCII character set but are not permitted in Ada programs, even within character strings. The compiler correctly recognises these characters as illegal in Ada compilations. The characters are not printed in the output listing. (See test B26005A.)

- Capacities.

  The compiler correctly processes compilations containing loop statements nested to 65 levels, block statements nested to 65 levels, recursive procedures nested to 17 levels. It correctly processes a compilation containing 723 variables in the same declarative part. (See tests D55A03A through D55A03H, D56001B, D64005E through D64005G, and D29002K.)

- Universal integer calculations.

  An implementation is allowed to reject universal integer calculations having values that exceed SYSTEM.MAX_INT. This implementation does not reject such calculations and processes them correctly. (See tests D4A002A, D4A002B, D4A004A, and D4A004B.)

- Predefined types.

  This implementation does not support additional predefined types in the package STANDARD. (See tests B86001CR, B86001CS, B86001CP, B86001CQ, and B86001DT.)

- 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 during execution. This implementation raises NUMERIC_ERROR during execution. (See test E24101A.)

- Array Types.

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

  When an array type is declared with an index range exceeding the INTEGER'LAST values and with a component that is a null BOOLEAN array, this compiler raises NUMERIC_ERROR when the type is declared. (See tests E36202A and E36202B.)

  A packed BOOLEAN array having a 'LENGTH exceeding INTEGER'LAST raises NUMERIC_ERROR when the array objects are declared. (See test C52103X.)

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

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

  In assigning one-dimensional array types, the entire expression appears to be evaluated 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 entire expression does not appear to be evaluated 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 during compilation. (See test E38104A.)

  In assigning record types with discriminants, the entire expression appears to be evaluated 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, 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.)

- Functions

  The declaration of a parameterless function with the same profile as an enumeration literal in the same immediate scope is allowed by the implementation. (See test E66001D.)

- Representation clauses.

  'SMALL length clauses are supported. (See test C87B62C.)

  Enumeration representation clauses are supported. (See test BC1002A.)

- Pragmas.

  The pragma INLINE is not supported for procedures. The pragma INLINE is not supported for functions. (See tests CA3004E and CA3004F.)

- Input/Output.

  The package SEQUENTIAL_IO can be instantiated with unconstrained array types and record types with discriminants. The package DIRECT_IO can be instantiated with unconstrained array types and record types with discriminants without defaults. (See tests CE2201D, CE2201E, and CE2401D.)

  This implementation implements input/output packages SEQUENTIAL_IO, DIRECT_IO and TEXT_IO as "null" packages. The packages raise two possible exceptions, details of which are given in paragraph F.8 of Appendix B.

CHAPTER 3

TEST INFORMATION

3.1 TEST RESULTS

The AVF identified 1986 of the 2279 tests in version 1.7 of the Ada Compiler Validation Capability as potentially applicable to the validation of the SD VAX x Motorola M68000/10 Ada-Plus, 2A.00. Excluded were 278 tests requiring a floating-point precision greater than that supported by the implementation and the 15 withdrawn tests. After they were processed, 73 tests were determined to be inapplicable. The remaining 1913 tests were passed by the compiler.

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

3.2 SUMMARY OF TEST RESULTS BY CLASS

RESULT	TEST CLASS						TOTAL
	A	B	C	D	E	L	
Passed	68	815	984	17	8	21	1913
Failed	0	0	0	0	0	0	0
Inapplicable	0	9	337	0	3	2	351
Withdrawn	0	4	11	0	0	0	15
TOTAL	68	828	1332	17	11	23	2279

3.3 SUMMARY OF TEST RESULTS BY CHAPTER

RESULT	CHAPTER												
	2	3	4	5	6	7	8	9	10	11	12	14	TOTAL
Passed	93	186	253	241	160	97	155	196	99	28	216	189	1913
Failed	0	0	0	0	0	0	0	0	0	0	0	0	0
Inapplicable	23	121	141	6	1	0	6	4	6	0	0	43	351
Withdrawn	0	1	4	0	0	0	1	1	6	0	1	1	15
TOTAL	116	308	398	247	161	97	162	201	111	28	217	233	2279

3.4 WITHDRAWN TESTS

The following tests have been withdrawn from the ACVC Version 1.7:

C35904A	C41404A	CA1003B	C48008A	BA2001E	B4A010C
CA3005A	CA3055B	CA3055C	CA3055D	C4A014A	BC3204C
B83A06B	CE2107E	C92005A			

See Appendix D for the test descriptions.

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. For this validation attempt, 351 tests were inapplicable for the reasons indicated:

C34001D
B52004E
B55B09D
C55B07B
B86001CR

These tests were inapplicable because the implementation does not support SHORT_INTEGER.

C34001F
C35702A
B86001CP

These tests were inapplicable because the implementation does not support SHORT_FLOAT.

C34001G
C35702B
B86001CQ

These tests were inapplicable because the implementation not support LONG_FLOAT.

C34001E
B52004D
B55B09C
C55B07A
B86001CS

These tests are inapplicable because the implementation does not support LONG_INTEGER.

B86001DT

This test is inapplicable because no integer types other than INTEGER, SHORT_INTEGER and LONG_INTEGER is supported.

C48006B

This test has been ruled inapplicable for this implementation. For access types with discriminant constraints it is unclear where the check should be produced.

C64104M

This test has been ruled inapplicable for this implementation. On the M68000/10 Target Computer the default storage allocation for a collection is 1K bytes. STORAGE_ERROR is raised during execution because the total size of the objects within the collection is greater than the default storage size. Although this test was ruled inapplicable, a modified version of this test was run to demonstrate that the compiler conforms to the standard. Test 64104M_REP is the same as test C64104M with the addition of a representation clause which increases the size of a collection from a default 1K bytes to 4K bytes. The test then passed successfully.

C86001F

A separate package is used to collect the executable test results from the M68000/10 target. The package TEST_IO uses the package SYSTEM, thus when this test recompiles package SYSTEM it invalidates the package TEST_IO. This means that the test cannot be built and executed.

C94004A
C94004B
C94004C

These tests have been ruled inapplicable for this implementation. This implementation terminates the library tasks when execution of the main program finishes. (Reference. Language Reference Manual, section 9.4).

C96005B

This test is inapplicable as, for this implementation, DURATION'BASE values below DURATION'FIRST and above DURATION'LAST do not exist.

CA3004E
CA3004F
EA3004C
EA3004D
LA3004A
LA3004B

These tests are inapplicable as the implementation does not support pragma INLINE for procedures and functions.

This implementation raises USE_ERROR when an attempt is made to create/open a file. As a result, the following tests are NOT APPLICABLE:

CE2104A	CE2111E	CE2410A
CE2104B	CE2111G	CE3102B
CE2104C	CE2111H	CE3107A
CE2104D	CE2201C	CE3108A
CE2107A	CE2401A	CE3108B
CE2107B	CE2401B	CE3112B
CE2107F	CE2401C	CE3114A
CE2110A	CE2401E	CE3114B
CE2110B	CE2401F	CE3115A
CE2110C	CE2404A	CE3305A
CE2111A	CE2405B	CE3704F
CE2111B	CE2406A	CE3704M
CE2111C	CE2408A	CE3704N
CE2111D	CE2409A	CE3905L
EE3102C		

The following tests were inapplicable because they exceed the accuracy of the floating point definition for the target implementation.

C24113C through C24113Y (23 tests)
C35705C through C35705Y (23 tests)
C35706C through C35706Y (23 tests)
C35707C through C35707Y (23 tests)
C35708C through C35708Y (23 tests)
C35802C through C35802Y (23 tests)
C45241C through C45241Y (23 tests)
C45321C through C45321Y (23 tests)
C45421C through C45421Y (23 tests)
C45424C through C45424Y (23 tests)
C45521C through C45521Z (24 tests)
C45621C through C45621Z (24 tests)

Also one of the support tests, CZ1103A does not produce output equivalent to the expected output. This is because the exception USE_ERROR is raised on all attempts to create a file within this test.

CZ1201D fails with the message "unknown exception generated" because the collection for the type text in package VAR_STRINGS has been exhausted and STORAGE_ERROR is raised. When package VAR_STRINGS was edited to increase the collection size to 4K the test ran successfully.

3.6 SPLIT TESTS

If one or more errors do not appear to have been detected in a Class B test because of compiler error recovery, then the test is split into a set of smaller tests that contain the undetected errors. These splits are then compiled and examined. The splitting process continues until all errors are detected by the compiler or until there is exactly one error per split. Any Class A, Class C, or Class E test that cannot be compiled and executed because of its size is split into a set of smaller subsets that can be processed.

Splits were required for only 9 Class B tests.

B22003A	BB3005A
B29001A	BC10AEA
B74401C	BC10AEB
B950ABA	BC3204B
B97101E	

3.7 ADDITIONAL TESTING INFORMATION

3.7.1 Prevalidation

Prior to validation, a set of test results for ACVC Version 1.7 produced by SD VAX x Motorola M68000/10 Ada-Plus, 2A.00, was submitted to the AVF by the applicant for prevalidation review. Analysis of these results demonstrated that the compiler successfully passed all applicable tests.

3.7.2 Test Method

Testing of SD VAX x Motorola M68000/10 Ada-Plus using ACVC Version 1.7 was conducted on-site by a validation team. The base configuration consisted of a Digital Equipment VAX 8600 host operating under VMS and a MC68010 target. The host and target computers were linked using a null modem implemented via a protocol conforming to RS232C.

A magnetic tape containing ACVC Version 1.7 was taken on-site by the validation team. The magnetic tape contained all tests, both applicable and inapplicable. Tests that make use of values that are specific to an implementation were customized before being written to the magnetic tape. The validation package REPORT was modified to use package TEST_IO instead of TEXT_IO. For targets without filing systems TEXT_IO cannot in general be used. The results from the target were sent back up to the host computer and stored in a file, which was then printed.

Tests requiring splits during the prevalidation testing were not included in their split form on the magnetic tape. Editing of the test files was necessary when the validation team arrived on-site.

The contents of the magnetic tape were loaded directly onto the host computer.

After the test files were loaded to disk, the full set of tests were compiled on the Digital Equipment VAX 8600, and all executable tests were run on the MC68010. For the base configuration, object files were linked on the host computer, and executable images were transferred to the target computer using a null modem implemented via a protocol conforming to RS232C. Results were printed from the target computer. Tests that were withdrawn from ACVC Version 1.7 were not run.

The compiler was tested using command scripts provided by Systems Designers. These scripts were reviewed by the validation team. The following options were in effect for testing:

Option

For the purpose of running the validation suite the following compiler options were used.

All tests were compiled with the option

option = list => on

to ensure that the compilation listings produced by the compiler contain the full listing of the test source.

The test C52008B requires a larger stack than all the other tests. The size of stack was increased at build-time to enable this test to run correctly.

3.7.3 TEST SITE

The validation team arrived at Systems Designers, Camberley on 13 June 1986 and departed after testing was completed on 16 June 1986.

APPENDIX A

COMPLIANCE STATEMENT

Systems Designers has submitted the following compliance statement concerning the SD VAX x Motorola M68000/10 Ada-Plus.

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 MIL-STD-1815A, and to certain allowed restrictions on representation classes. The implementation-dependent characteristics of the SD Vax x Motorola M68000/10 Ada Plus, 2A.00, are described in the following sections which discuss topics one through eight as stated in Appendix F of the Ada Language Reference Manual (ANSI/MIL-STD-1815A). Two other sections, package STANDARD and file naming conventions, are also included in this appendix.

CONTENTS

PREFACE

APPENDIX F IMPLEMENTATION-DEPENDENT CHARACTERISTICS

F.1 IMPLEMENTATION-DEPENDENT PRAGMAS
F.1.1 Pragma EXPORT
F.1.2 Pragma DEBUG
F.1.3 Pragma SUPPRESS_ALL
F.2 IMPLEMENTATION-DEPENDENT ATTRIBUTES
F.3 PACKAGE SYSTEM
F.4 RESTRICTIONS ON REPRESENTATION CLAUSES
F.4.1 Length Clauses
F.4.1.1 Attribute SIZE
F.4.1.2 Attribute STORAGE_SIZE
F.4.1.3 Attribute SMALL
F.4.2 Record Representation Clauses
F.4.2.1 Alignment Clause
F.4.2.2 Component Clause
F.4.3 Address Clauses
F.4.3.1 Object Addresses
F.4.3.2 Entry Addresses
F.5 IMPLEMENTATION-GENERATED NAMES
F.6 INTERPRETATION OF EXPRESSIONS IN ADDRESS CLAUSES
F.7 UNCHECKED CONVERSIONS
F.8 CHARACTERISTICS OF THE INPUT/OUTPUT PACKAGES
F.8.1 The Package TEXT_IO
F.8.2 The Package IO_EXCEPTIONS
F.9 PACKAGE STANDARD
F.10 PACKAGE MACHINE_CODE
F.11 LANGUAGE-DEFINED PRAGMAS
F.11.1 Pragma INLINE
F.11.2 Pragma INTERFACE
F.11.2.1 Assembler Names
F.11.2.2 Parameter Passing Conventions
F.11.2.3 Procedure-Calling Mechanism
F.11.3 Pragma OPTIMISE
F.11.4 Pragma SUPPRESS

FIGURES

Fig. F.1 Package SYSTEM
Fig. F.2 Package STANDARD
Fig. F.3 Routine Activation Record on Entry to Called Subprogram
Fig. F.4 Routine Entry And Exit Code

PREFACE

This document describes the implementation-dependent characteristics of the VAX/VMS x MC68000/10 SD-Ada Compiler.

The document should be considered as Appendix F of the Reference Manual for the Ada Programming Language.

APPENDIX F

IMPLEMENTATION-DEPENDENT CHARACTERISTICS

F.1 IMPLEMENTATION-DEPENDENT PRAGMAS

F.1.1 Pragma EXPORT

Form

```
pragma EXPORT ([ADA_NAME=>] simple_name,
               [EXT_NAME=>] "name_string");
```

The pragma EXPORT takes the name of an Ada variable in the first parameter position and a string in the second parameter position. The name must be the simple name of a variable in the package level static data area in scope, and name-string must be a string literal which is unique in any program produced for the target, otherwise the program is erroneous.

The parameter name_string must be a string literal which conforms to the naming conventions imposed by the MC68000/10 linker. The name must be no more than eight characters in length and start with a dot or upper case letter. The rest of the characters are restricted to being a digit, dot, dollar, underline or upper case letter.

Position

The pragma EXPORT may be placed at the position of a basic declarative item of a library package specification or in the declarative part of a library package body.

Effect

Use of this pragma causes the compiler to generate additional linkage information. This associates the string literal of the second parameter with the variable nominated by the first parameter. This external naming facility is restricted to data objects held in static areas.

F.1.2 Pragma DEBUG

Form

```
pragma DEBUG ([NAME=>]name);
```

The pragma DEBUG takes a name as the single argument. The value yielded by the parameter must be scalar or access type.

Position

The pragma DEBUG may be placed at the position of a basic_declarative_item or a statement where the name is in scope.

Effect

Use of this pragma causes the compiler to generate tracing code, and auxiliary information in debug symbol tables. This tracing code is loaded into the target computer in such a way that the main thread of normal execution perceives no reference to the trace code, and the values embedded in the main thread code, such as offsets, remain unaffected.

The tracing code may be activated by use of the Debug System.

F.1.3 Pragma SUPPRESS_ALL

Form

pragma SUPPRESS_ALL;

This pragma has no parameters.

Position

The pragma SUPPRESS_ALL is only allowed at the start of a compilation before the first compilation unit.

Effect

Use of this pragma prevents the compiler from generating any run-time checks for CONSTRAINT_ERROR or NUMERIC_ERROR.

F.2 IMPLEMENTATION-DEPENDENT ATTRIBUTES

There are no such attributes.

F.3 PACKAGE SYSTEM

The specification of the package SYSTEM is given in Figure F.1.

```
package SYSTEM  is

   type ADDRESS is private

   type NAME    is (MC68000,MC68010);

   SYSTEM_NAME   : constant NAME := MC68010
   STORAGE_UNIT  : constant      := 8;
   MEMORY_SIZE   : constant      := 1677721
   MIN_INT       : constant      := -2147483648
   MAX_INT       : constant      := 2147483647
   MAX_DIGITS    : constant      := 6;
   MAX_MANTISSA  : constant      := 31;
   FINE_DELTA    : constant      := 2#1.0#E-30;
   TICK          : constant      := 2#1.0#E-7;

   subtype PRIORITY is INTEGER range 0 .. 15;

   type UNIVERSAL_INTEGER is range MIN_INT .. MAX_INT;

   subtype EXTERNAL_ADDRESS is STRING;

   subtype BYTE is INTEGER range -128 .. 127;

   type    LONG_WORD is array (0..3) of BYTE;

   pragma  PACK(LONG_WORD);

   function   CONVERT_ADDRESS (ADDR   : EXTERNAL_ADDRESS)
       return ADDRESS;

   function   CONVERT_ADDRESS (ADDR   : ADDRESS)
       return EXTERNAL_ADDRESS;

   function   CONVERT_ADDRESS (ADDR   : LONG_WORD)
       return ADDRESS;

   function   CONVERT_ADDRESS (ADDR   : ADDRESS)
       return LONG_WORD;

   function   "+"              (ADDR   : ADDRESS;
                                OFFSET : UNIVERSAL_INTEGER)
       return ADDRESS;

   private

       -- type ADDRESS is system-dependent

   end SYSTEM;
```

Figure F.1

Package SYSTEM

F.4 RESTRICTIONS ON REPRESENTATION CLAUSES

F.4.1 Length Clauses

F.4.1.1 Attribute SIZE

The value specified for SIZE must not be less than that chosen by default by the compiler (e.g. 8 for enumeration types, 32 for integer types, real types and access types, etc.). The value given is ignored.

F.4.1.2 Attribute STORAGE_SIZE

For access types the limit is governed by the indexing range of the target machine and the maximum is equivalent to SYSTEM.ADDRESS'LAST.

For task types the limit is also SYSTEM.ADDRESS'LAST.

F.4.1.3 Attribute SMALL

Only values which are powers of two are supported for this attribute.

F.4.2 Record Representation Clauses

F.4.2.1 Alignment Clause

The static_simple_expression used to align records onto storage unit boundaries must deliver the values 1 or 2.

F.4.2.2 Component Clause

The static range is restricted to ranges which force component alignment onto storage unit boundaries only, (i.e. multiples of 8 bits).

The component size defined by the static range must not be less than the minimum number of bits required to hold every allowable value of the component. For a component of non-scalar type, the size must not be larger than that chosen by the compiler for the type.

F.4.3 Address Clause

F.4.3.1 Object Addresses

For objects with an address clause, a pointer is declared which points to the object at the given address. There is a restriction however that the object cannot be initialised either explicitly or implicitly (i.e the object cannot be an access type).

F.4.3.2 Entry Addresses

Address clauses for entries are not supported by this version of the SD-Ada Compiler.

F.5 IMPLEMENTATION-GENERATED NAMES

There are no implementation-generated names denoting implementation-dependent components.

F.6 INTERPRETATION OF EXPRESSIONS IN ADDRESS CLAUSES

The expressions in an address clause are interpreted as absolute addresses on the target.

F.7 UNCHECKED CONVERSIONS

The implementation imposes the restriction on the use of the generic function UNCHECKED_CONVERSION that the size of the target type must not be greater than the size of the source type.

F.8 CHARACTERISTICS OF THE INPUT/OUTPUT PACKAGES

Packages SEQUENTIAL_IO, DIRECT_IO and the predefined input/output package TEXT_IO are implemented as "null" packages which conform to the specification given in the Ada Language Reference Manual. This package raises the exceptions specified in Chapter 14 of the Language Reference Manual. There are two possible exceptions which are raised by this package. These are given here in the order in which they will be raised.

a) The exception STATUS_ERROR is raised by an attempt to operate upon a file that is not open (no files can be opened).

b) The exception USE_ERROR is raised if exception STATUS_ERROR is not raised.

Note that MODE_ERROR cannot be raised as no file can be opened (therefore it cannot have a current mode) and NAME_ERROR cannot be raised since there are no restrictions on file names.

The predefined package IO_EXCEPTIONS is defined in the Ada Language Reference Manual.

The predefined package LOW_LEVEL_IO is not provided.

The implementation-dependent characteristics are described in Sections F.8.1 to F.8.2.

F.8.1 The Package TEXT_IO

When any procedure is called the exception STATUS_ERROR or USE_ERROR is raised (there are no restrictions on the format of the NAME or FORM parameters).

The type COUNT is defined:-

```
type COUNT is range 0 .. INTEGER'LAST;
```

and the subtype FIELD is defined:

```
subtype FIELD is INTEGER range 0 .. 132;
```

F.8.2 The Package IO_EXCEPTIONS

The specification of the package is the same as that given in the Ada Language Reference Manual.

F.9 PACKAGE STANDARD

The specification of package STANDARD is given in Figure F.2.

```
package STANDARD is

  type BOOLEAN is (FALSE, TRUE);

  type INTEGER is range
    - 2147483648 .. 2147483647;

  type FLOAT is digits 6 range
    - 16#0.FFFFFF#E32 .. 16#0.FFFFFF#E32;

   type CHARACTER is

     (nul, soh, stx, etx,      eot, enq, ack, bel,

      bs , ht , lf , vt ,      ff , cr , so , si ,

      dle, dc1, dc2, dc3,      dc4, nak, syn, etb,

      can, em , sub, esc,      fs , gs , rs , us ,

      ' ', '!', '"', '#',      '$', '%', '&', ''',
      '(', ')', '*', '+',      ',', '-', '.', '/',
      '0', '1', '2', '3',      '4', '5', '6', '7',
      '8', '9', ':', ';',      '<', '=', '>', '?',

      '@', 'A', 'B', 'C',      'D', 'E', 'F', 'G',
      'H', 'I', 'J', 'K',      'L', 'M', 'N', 'O',
      'P', 'Q', 'R', 'S',      'T', 'U', 'V', 'W',
      'X', 'Y', 'Z', '[',      '\', ']', '^', '_',

      '`', 'a', 'b', 'c',      'd', 'e', 'f', 'g',
      'h', 'i', 'j', 'k',      'l', 'm', 'n', 'o',
      'p', 'q', 'r', 's',      't', 'u', 'v', 'w',
      'x', 'y', 'z', '{',      '|', '}', '~', del);
```

Figure F.2 (1 of 4)

Package STANDARD

```
for   CHARACTER use   -- ASCII characters without holes
   (0   , 1   , 2   , 3   , 4   , 5   , 6   , 7   ,
    8   , 9   , 10  , 11  , 12  , 13  , 14  , 15  ,
    16  , 17  , 18  , 19  , 20  , 21  , 22  , 23  ,
    24  , 25  , 26  , 27  , 28  , 29  , 30  , 31  ,
    32  , 33  , 34  , 35  , 36  , 37  , 38  , 39  ,
    40  , 41  , 42  , 43  , 44  , 45  , 46  , 47  ,
    48  , 49  , 50  , 51  , 52  , 53  , 54  , 55  ,
    56  , 57  , 58  , 59  , 60  , 61  , 62  , 63  ,
    64  , 65  , 66  , 67  , 68  , 69  , 70  , 71  ,
    72  , 73  , 74  , 75  , 76  , 77  , 78  , 79  ,
    80  , 81  , 82  , 83  , 84  , 85  , 86  , 87  ,
    88  , 89  , 90  , 91  , 92  , 93  , 94  , 95  ,
    96  , 97  , 98  , 99  , 100, 101, 102, 103,
    104, 105, 106, 107, 108, 109, 110, 111,
    112, 113, 114, 115, 116, 117, 118, 119,
    120, 121, 122, 123, 124, 125, 126, 127);

package ASCII is

   -- Control characters:

   NUL         : constant CHARACTER := nul;
   SOH         : constant CHARACTER := soh;
   STX         : constant CHARACTER := stx;
   ETX         : constant CHARACTER := etx;
   EOT         : constant CHARACTER := eot;
   ENQ         : constant CHARACTER := enq;
   ACK         : constant CHARACTER := ack;
   BEL         : constant CHARACTER := bel;
   BS          : constant CHARACTER := bs;
   HT          : constant CHARACTER := ht;
   LF          : constant CHARACTER := lf;

   VT          : constant CHARACTER := vt;
   FF          : constant CHARACTER := ff;
   CR          : constant CHARACTER := cr;
   SO          : constant CHARACTER := so;
   SI          : constant CHARACTER := si;
   DLE         : constant CHARACTER := dle;
   DC1         : constant CHARACTER := dc1;
   DC2         : constant CHARACTER := dc2;
   DC3         : constant CHARACTER := dc3;
   DC4         : constant CHARACTER := dc4;
   NAK         : constant CHARACTER := nak;
   SYN         : constant CHARACTER := syn;
```

Figure F.2 (2 of 4)

Package STANDARD

```
ETB          : constant CHARACTER := etb;
CAN          : constant CHARACTER := can;
EM           : constant CHARACTER := em;
SUB          : constant CHARACTER := sub;
ESC          : constant CHARACTER := esc;
FS           : constant CHARACTER := fs;
GS           : constant CHARACTER := gs;
RS           : constant CHARACTER := rs;
US           : constant CHARACTER := us;
DEL          : constant CHARACTER := del;

-- Other characters:

EXCLAM       : constant CHARACTER := '!';
QUOTATION    : constant CHARACTER := '"';
SHARP        : constant CHARACTER := '#';
DOLLAR       : constant CHARACTER := '$';
PERCENT      : constant CHARACTER := '%';
AMPERSAND    : constant CHARACTER := '&';
COLON        : constant CHARACTER := ':';
SEMICOLON    : constant CHARACTER := ';';
QUERY        : constant CHARACTER := '?';
AT_SIGN      : constant CHARACTER := '@';

L_BRACKET    : constant CHARACTER := '[';
BACK_SLASH   : constant CHARACTER := '\';
R_BRACKET    : constant CHARACTER := ']';
CIRCUMFLEX   : constant CHARACTER := '^';
UNDERLINE    : constant CHARACTER := '_';
GRAVE        : constant CHARACTER := '`';
L_BRACE      : constant CHARACTER := '{';
BAR          : constant CHARACTER := '|';
R_BRACE      : constant CHARACTER := '}';
TILDE        : constant CHARACTER := '~';

-- Lower case letters:

LC_A         : constant CHARACTER := 'a';
LC_B         : constant CHARACTER := 'b';
LC_C         : constant CHARACTER := 'c';
LC_D         : constant CHARACTER := 'd';
LC_E         : cosntant CHARACTER := 'e';
LC_F         : constant CHARACTER := 'f';
LC_G         : constant CHARACTER := 'g';
LC_H         : constant CHARACTER := 'h';
```

Figure F.2 (3 of 4)

Package STANDARD

```
      LC_I        : constant CHARACTER := 'i';
      LC_J        : constant CHARACTER := 'j';
      LC_K        : constant CHARACTER := 'k';
      LC_L        : constant CHARACTER := 'l';
      LC_M        : constant CHARACTER := 'm';
      LC_N        : constant CHARACTER := 'n';
      LC_O        : constant CHARACTER := 'o';
      LC_P        : constant CHARACTER := 'p';
      LC_Q        : constant CHARACTER := 'q';
      LC_R        : constant CHARACTER := 'r';
      LC_S        : constant CHARACTER := 's';
      LC_T        : constant CHARACTER := 't';
      LC_U        : constant CHARACTER := 'u';
      LC_V        : constant CHARACTER := 'v';
      LC_W        : constant CHARACTER := 'w';
      LC_X        : constant CHARACTER := 'x';
      LC_Y        : constant CHARACTER := 'y';
      LC_Z        : constant CHARACTER := 'z';

   end ASCII;

   -- Predefined subtypes:

   subtype NATURAL is INTEGER
                         range 0 .. INTEGER'LAST;

   subtype POSITIVE is INTEGER
                         range 1 .. INTEGER'LAST;

   -- Predefined string type:

   type STRING is array (POSITIVE range <>)
                  of CHARACTER;

   type DURATION is delta 2#1.0#E-7
                    range -16777216.0 .. 16777215.0;

   -- The predefined exceptions:

   CONSTRAINT_ERROR : exception;
   NUMERIC_ERROR    : exception;
   PROGRAM_ERROR    : exception;
   STORAGE_ERROR    : exception;
   TASKING_ERROR    : exception;

end STANDARD;
```

Figure F.2 (4 of 4)

Package STANDARD

F.10 PACKAGE MACHINE_CODE

Package MACHINE_CODE is not supported by the SD-Ada Compiler.

F.11 LANGUAGE-DEFINED PRAGMAS

The definition of certain language-defined pragmas is incomplete in the Ada Language Reference Manual. The implementation restrictions imposed on the use of such pragmas are specified in Sections F.11.1 to F.11.4.

F.11.1 Pragma INLINE

This pragma supplies a recommendation for inline expansion of a subprogram to the compiler. This pragma is ignored by the SD-Ada Compiler.

F.11.2 Pragma INTERFACE

This pragma allows subprograms written in another language to be called from Ada. The SD-Ada Compiler only supports pragma INTERFACE for the language ASSEMBLER. Normal Ada calling conventions are used by the SD-Ada Compiler when generating a call to an ASSEMBLER subprogram.

F.11.2.1 Assembler Names

The name of an interface routine must conform to the naming conventions both of Ada and of the MC68000/10 linker.

F.11.2.2 Parameter Passing Conventions

Parameters are passed to the called procedure in the order given in the specification of the subprogram, with default expressions evaluated, if present.

Scalars are passed by copy for all parameter modes (the value is copied out for parameters with mode out).

Composite types are passed by reference for all parameter modes.

F.11.2.3 Procedure-Calling Mechanism

The procedure-calling mechanism uses the run-time stack organisation shown in Figure F.3 and the routine entry and exit code shown in Figure F.4.

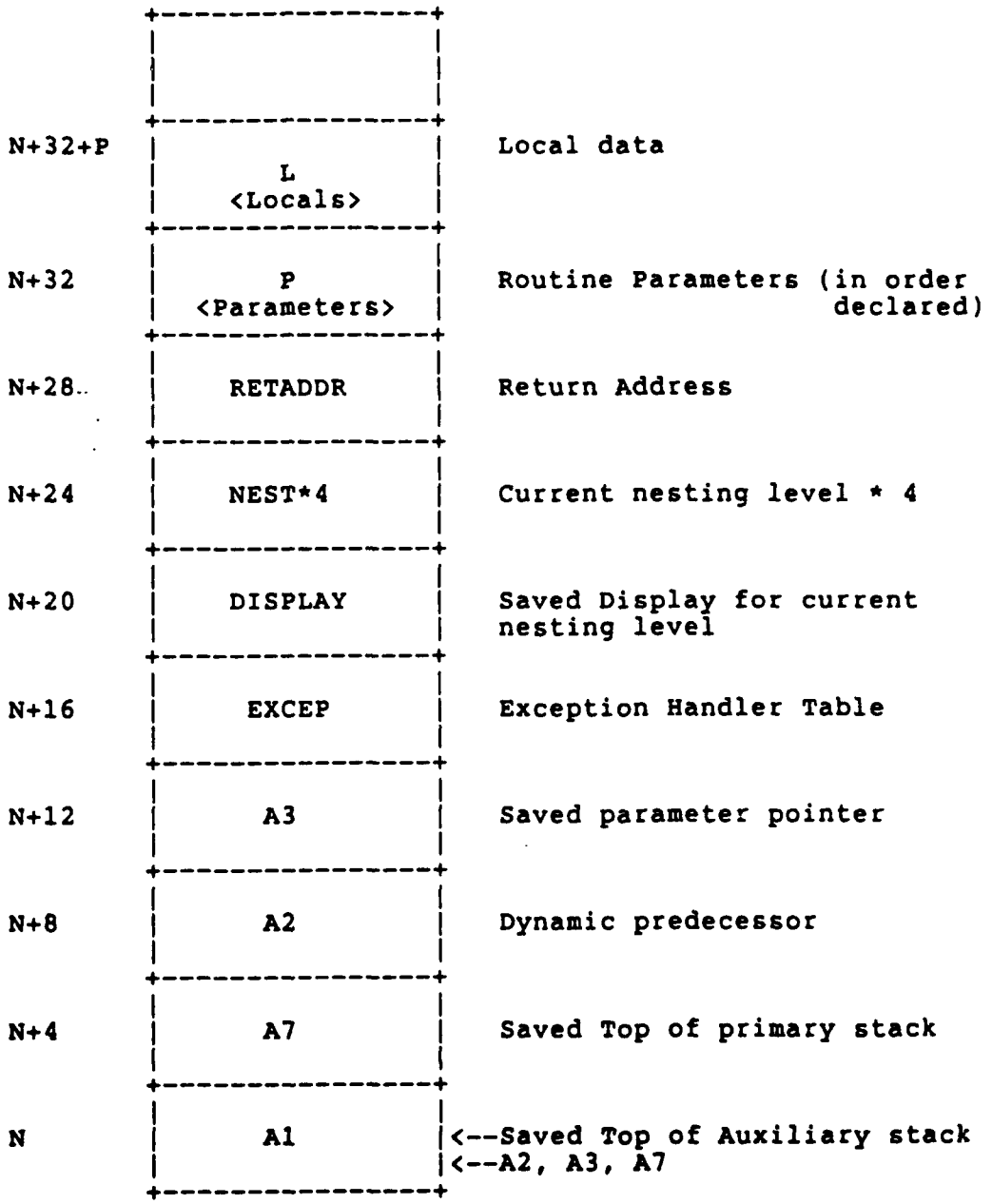


Figure F.3

**Routine Activation Record
on Entry to Called Subprogram**

The implementation uses the following dedicated and temporary registers:

SP	-	Main Stack Pointer	A7
FP	-	Frame Pointer	A2
PM	-	Parameter Frame Pointer	A3
DP	-	Display Pointer	A0
AP	-	Auxiliary Stack Pointer	A1

The routine entry and exit code is shown in Figure F.4.

Routine Entry Code

```
MOVE.L (A7)+, <return_address> (A2)
MOVE.L #<nest*4>, <saved_nest> (A2)
MOVE.L <nest*4> (A0), <saved_display> (A2)
MOVE.L A2, <nest*4> (A0)
```

Routine Exit Code

```
MOVE.L <Saved_display> (A2), A0(<nest*4>)
MOVEA.L <return_address> (A2), Ax
JMP    (Ax)
```

Figure F.4

Routine Entry And Exit Code

F.11.3 Pragma OPTIMISE

This pragma supplies a recommendation to the compiler for the criterion upon which optimisation is to be performed. This pragma is ignored by the SD-Ada Compiler.

F.11.4 Pragma SUPPRESS

This pragma gives permission for specified run-time checks to be omitted by the compiler. This pragma is ignored by the SD-Ada Compiler.

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 identified by names that begin with a dollar sign. A value is substituted for each of these names before the test is run. The values used for this validation are given below.

NAME AND MEANING	VALUE
$BIG_ID1 Identifier of size MAX_IN_LEN with varying last character.	A....A1 |----| 79 characters
$BIG_ID2 Identifier of size MAX_IN_LEN with varying last character.	A....A2 |----| 79 characters
$BIG_ID3 Identifier of size MAX_IN_LEN with varying middle character.	A....A3A....A |----| |----| 38 41 characters
$BIG_ID4 Identifier of size MAX_IN_LEN with varying middle character.	A....A4A....A |----| |----| 38 41 characters
$BIG_INT_LIT An integer literal of value 298 with enough leading zeroes so that is is MAX_INT_LEN characters long.	0....0298 |----| 77 characters
$BIG_REAL_LIT A real literal that can be either of floating- or fixed-point type, has value of 690.0, and has enough leading zeroes to be MAX_IN_LEN characters long.	0....0690.0 |----| 75 characters
$BLANKS Blanks of length MAX_IN_LEN - 20	60 blanks

NAME AND MEANING	VALUE
$COUNT_LAST Value of COUNT'LAST in TEXT_IO package.	2147483647
$EXTENDED_ASCII_CHARS A string literal containing all the ASCII characters with printable graphics that are not in the basic 55 Ada character set.	"abcdefghijklmnopqrstuvwxyz !$%?@[\]^`{}"
$FIELD_LAST Value of FIELD'LAST in TEXT_IO package.	255
$FILE_NAME_WITH_BAD_CHARS An illegal external file name that either contains invalid characters or is too long.	x}]!.dat
$FILE_NAME_WITH_WILD_CARD_CHAR An external file name that either contains a wild card character or is too long.	file*.dat
$GREATER_THAN_DURATION A universal real value that lies between DURATION'BASE'LAST and DURATION'LAST or any value in the range of DURATION.	-2.0
$GREATER_THAN_DURATION_BASE_LAST The universal real value that is greater than DURATION'BASE'LAST.	16777216.0
$ILLEGAL_EXTERNAL_FILE_NAME1 Illegal external file name.	bad_char^
$ILLEGAL_EXTERNAL_FILE_NAME2 Illegal external file name.	this_is_a_remarkably_long_name _for_afile.dat
$INTEGER_FIRST The universal integer literal expression whose value is INTEGER'FIRST.	-2147483648

NAME AND MEANING	VALUE
$INTEGER_LAST The universal integer literal expression whose value is INTEGER'LAST.	2147483647
$LESS_THAN_DURATION A universal real value that lies between DURATION'BASE'FIRST and DURATION'FIRST or any value in the range of DURATION.	-2.0
$LESS_THAN_DURATION_BASE_FIRST The universal real value that is less than DURATION'BASE'FIRST.	-16777217.0
$MAX_DIGITS Maximum digits supported for floating-point types.	6
$MAX_IN_LEN Maximum input line length permitted by the implementation.	80
$NAME A name of a predefined numeric type other than FLOAT, INTEGER, SHORT_FLOAT, SHORT_INTEGER, LONG_FLOAT, or LONG_INTEGER.	$NAME
$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#FFFFFFFF#
$NON_ASCII_CHAR_TYPE An enumerated type definition for a character type whose literals are the identifier NON_NULL and all non_ASCII characters with printable graphics.	(NON_NULL)

APPENDIX D

WITHDRAWN TESTS

Some tests are withdrawn from the ACVC because they do not conform to the Ada Standard. When testing was performed, the following 15 tests had been withdrawn at the time of validation testing for the reasons indicated.

ACVC 1.7 Withdrawn Tests
86-03-31

The following tests are incorrect and have been withdrawn from ACVC 1.7.

C35904A: The elaboration of subtype declarations SFX3 & SFX4 may raise NUMERIC_ERROR vs. CONSTRAINT_ERROR.

C41404A: The values of 'LAST and 'LENGTH in the "if" statement from line 74 to the end of the test are incorrect.

C48008A: This test requires that the evaluation of default initial values not occur if an exception is raised by an allocator. However, the LMC has ruled that such a requirement is incorrect (AI-00397).

B4A010C: The object_declaration in line 18 follows a subprogram body of the same declarative part.

C4A014A: The number declarations in lines 19-22 are not correct, because conversions are not static.

B83A06B: The Ada Standard 8.3(17) and AI-00330 permit the label LAB_ENUMERAL of line 80 to be considered a homograph of the enumeration literal in line 25.

C92005A: At line 40, "/=" for type PACK.BIG_INT is not visible without a "use" clause for package PACK.

CA1003B: This test requires all of the legal compilation units of a file containing some illegal units to be compiled and executed. But according to AI-00255 such a file may be rejected as a whole.

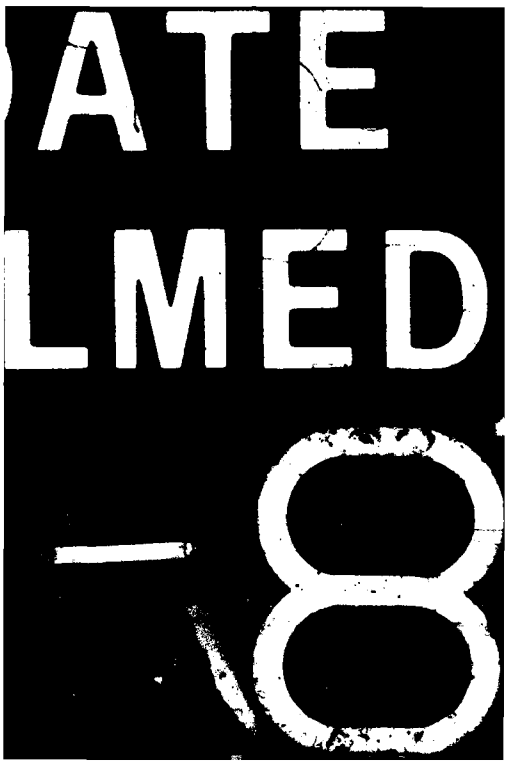
BA2001E: The Ada Standard 10.2(5) states that "simple names of all subunits that have the same ancestor library unit must be distinct identifiers." This test checks for the above condition when stubs are declared; but it is not clear that the check must be made then, as opposed to when the subunit is compiled.

CA3005A..D (4 tests): There exists no valid elaboration order for these tests.

BC3204C: The file BC3204C4 should contain the body for BC3204C0 --as indicated in line 25 of BC3204C3M.

CE2107E: TEMP_HAS_NAME must be given an initial value of TRUE.

*****END OF LIST*****


ATE
LMED
-8