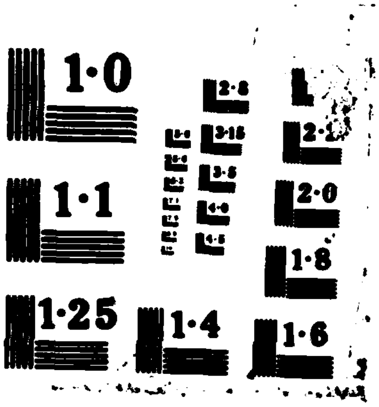
AD-A191 636 ADA (TRADE NAME) COMPILER VALIDATION SUMMARY REPORT 1/1
TLD SYSTEMS LTD TLD V (U) AERONAUTICAL SYSTEMS DIV
WRIGHT-PATTERSON AFB OH ADA VALIDATI 22 JUN 87

UNCLASSIFIED F/G 12/5 NL


1·0
2·8
3·2
2·5
3·6
2·2
4·0
2·0
1·1
1·8
1·25
1·4
1·6

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

DTIC FILE COPY

2

AD-A191 636

IENTATION PAGE

	READ INSTRUCTIONS BEFORE COMPLETING FORM
2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE *(and Subtitle)* Ada Compiler Validation Summary Report: TLD Systems Ltd. TLD VAX/1750A Ada Compiler System.Ver 1.0.0.,MicroVAX II Host and TLD 1750A Instruction	5. TYPE OF REPORT & PERIOD COVERED 22 June 1987 to 22 June 1988
	6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Wright-Patterson AFB	8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION AND ADDRESS Ada Validation Facility 9/785-[illegible] ASD/SIOL Wright-Patterson AFB OH 45433-6503	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 22 June 1987
	13. NUMBER OF PAGES 35
14. MONITORING AGENCY NAME & ADDRESS*(If different from Controlling Office)* Wright-Patterson	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)*

See Attached

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

12 29 641

EXECUTIVE SUMMARY

This Validation Summary Report ~~(VSR)~~ summarizes the results and conclusions of validation testing performed on the TLD VAX/1750A Ada Compiler System, Version 1.0.0, using Version 1.8 of the Ada® Compiler Validation Capability (ACVC). The TLD VAX/1750A Ada Compiler System is hosted on a MicroVAX II operating under MicroVMS, Version 4.5. Programs processed by this compiler may be executed on a TLD 1750A Instruction Level Simulator, Version 0.4.4 running the TLD 1750A Single Program Kernel.

On-site testing was performed 19 June 1987 through 22 June 1987 at TLD Systems Ltd., Torrance CA, under the direction of the Ada Validation Facility (AVF), according to Ada Validation Organization ~~(AVO)~~ policies and procedures. The AVF identified 2102 of the 2399 tests in ACVC Version 1.8 to be processed during on-site testing of the compiler. The 19 tests withdrawn at the time of validation testing, as well as the 278 executable tests that make use of floating-point precision exceeding that supported by the implementation, were not processed. After the 2102 tests were processed, results for Class A, C, D, and E tests were examined for correct execution. Compilation listings for Class B tests were analyzed for correct diagnosis of syntax and semantic errors. Compilation and link results of Class L tests were analyzed for correct detection of errors. There were 185 of the processed tests determined to be inapplicable. The remaining 1917 tests were passed.

The results of validation are summarized in the following table:

RESULT	CHAPTER												TOTAL
	2	3	4	5	6	7	8	9	10	11	12	14	
Passed	93	205	280	244	159	97	135	261	124	32	218	69	1917
Failed	0	0	0	0	0	0	0	0	0	0	0	0	0
Inapplicable	23	120	140	3	2	0	4	1	6	0	0	164	463
Withdrawn	0	5	5	0	0	1	1	2	4	0	1	0	19
TOTAL	116	330	425	247	161	98	140	264	134	32	219	233	2399

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

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

AVF Control Number: AVF-VSR-89.0887
87-03-09-TLD

Ada® COMPILER
VALIDATION SUMMARY REPORT:
TLD Systems Ltd.
TLD VAX/1750A Ada Compiler System
Version 1.0.0
MircoVAX II Host
and
TLD 1750A Instruction
Level Simulator Target

Completion of On-Site Testing:
22 June 1987

Prepared By:
Ada Validation Facility
ASD/SCOL
Wright-Patterson AFB OH 45433-6503

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 +
+                      +
++++++++++++++++++++++++
```

Ada® Compiler Validation Summary Report:

Compiler Name: TLD VAX/1750A Ada Compiler System, Version 1.0.0

Host:	MicroVAX II under MicroVMS, Version 4.5	Target:	TLD 1750A Instruction Level Simulator, Version 0.4.4 under TLD 1750A Single Program Kernel

Testing Completed 22 June 1987 Using ACVC 1.8

This report has been reviewed and is approved.

Georgeanne Chitwood

Ada Validation Facility
Georgeanne Chitwood
ASD/SCOL
Wright-Patterson AFB OH 45433-6503

John F Kramer

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

Virginia L. Castor

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

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

EXECUTIVE SUMMARY

This Validation Summary Report (VSR) summarizes the results and conclusions of validation testing performed on the TLD VAX/1750A Ada Compiler System, Version 1.0.0, using Version 1.8 of the Ada® Compiler Validation Capability (ACVC). The TLD VAX/1750A Ada Compiler System is hosted on a MicroVAX II operating under MicroVMS, Version 4.5. Programs processed by this compiler may be executed on a TLD 1750A Instruction Level Simulator, Version 0.4.4 running the TLD 1750A Single Program Kernel.

On-site testing was performed 19 June 1987 through 22 June 1987 at TLD Systems Ltd., Torrance CA, under the direction of the Ada Validation Facility (AVF), according to Ada Validation Organization (AVO) policies and procedures. The AVF identified 2102 of the 2399 tests in ACVC Version 1.8 to be processed during on-site testing of the compiler. The 19 tests withdrawn at the time of validation testing, as well as the 278 executable tests that make use of floating-point precision exceeding that supported by the implementation, were not processed. After the 2102 tests were processed, results for Class A, C, D, and E tests were examined for correct execution. Compilation listings for Class B tests were analyzed for correct diagnosis of syntax and semantic errors. Compilation and link results of Class L tests were analyzed for correct detection of errors. There were 185 of the processed tests determined to be inapplicable. The remaining 1917 tests were passed.

The results of validation are summarized in the following table:

RESULT	CHAPTER												TOTAL
	2	3	4	5	6	7	8	9	10	11	12	14	
Passed	93	205	280	244	159	97	135	261	124	32	218	69	1917
Failed	0	0	0	0	0	0	0	0	0	0	0	0	0
Inapplicable	23	120	140	3	2	0	4	1	6	0	0	164	463
Withdrawn	0	5	5	0	0	1	1	2	4	0	1	0	19
TOTAL	116	330	425	247	161	98	140	264	134	32	219	233	2399

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

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

TABLE OF 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 characteristics of particular operating systems, hardware, or implementation strategies. All of 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 unsupported language constructs required by the Ada Standard

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

Testing of this compiler was conducted by SofTech, Inc., under the direction of the AVF according to policies and procedures established by the Ada Validation Organization (AVO). On-site testing was conducted from 19 June 1987 through 22 June 1987 at TLD Systems Ltd., Torrance CA.

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:

Ada Validation Facility
ASD/SCOL
Wright-Patterson AFB OH 45433-6503

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, FEB 1983.

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

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

1.4 DEFINITION OF TERMS

ACVC The Ada Compiler Validation Capability. A set of programs that evaluates the conformity 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 Ada Validation Facility. 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 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 nonconformity 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.

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 checks a compiler's conformity regarding a particular feature or 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 — A test found to be incorrect and not used to check conformity to the Ada language specification. 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 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 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 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.

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 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: TLD VAX/1750A Ada Compiler System, Version 1.0.0

ACVC Version: 1.8

Certificate Number: 870622W1.08085

Host Computer:

Machine: MicroVAX II

Operating System: MicroVMS, Version 4.5

Memory Size: 9 megabytes

Target Computer:

Machine: TLD 1750A Instruction Level Simulator, Version 0.4.4

Operating System: TLD 1750A Single Program Kernel

Memory Size: 64K 16 bit words

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. This compiler is characterized by the following interpretations of the Ada Standard:

- 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 8 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 does not reject such calculations and processes them correctly. (See tests D4A002A, D4A002B, D4A004A, and D4A004B.)

- Predefined types.

  This implementation supports the additional predefined type LONG_INTEGER 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 NUMERIC_ERROR during execution. (See test E24101A.)

- 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.

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 one packed boolean array 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 NUMERIC_ERROR when the array type is declared. (See test E52103Y.)

In assigning one-dimensional array types, the expression appears 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 appears 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 appears 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, index subtype checks appear to be made as choices are evaluated. (See tests C43207A and C43207B.)

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

All choices are not 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.

An implementation may allow the declaration of a parameterless function and an enumeration literal having the same profile in the same immediate scope, or it may reject the function declaration. If it accepts the function declaration, the use of the enumeration literal's identifier denotes the function. This implementation rejects the declaration. (See test E66001D.)

- Representation clauses.

The Ada Standard does not require an implementation to support representation clauses. If a representation clause is not supported, then the implementation must reject it. While the operation of representation clauses is not checked by Version 1.8 of the ACVC, they are used in testing other language features. This implementation accepts 'SIZE and 'STORAGE_SIZE for tasks; it rejects 'STORAGE_SIZE for collections, and 'SMALL clauses. Enumeration representation clauses, including those that specify noncontiguous values, appear to be supported. (See tests C55B16A, C87B62A, C87B62B, C87B62C, and BC1002A.)

- Pragmas.

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

- Input/output.

This implementation supports only the package TEXT_IO for file operations on STANDARD_INPUT and STANDARD_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. However, any call to OPEN or CREATE of such instances of SEQUENTIAL_IO or DIRECT_IO with these types raises an exception. (See tests AE2101C, AE2101H, CE2201D, CE2201E, and CE2401D.)

- Generics.

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

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

CHAPTER 3

TEST INFORMATION

3.1 TEST RESULTS

Version 1.8 of the ACVC contains 2399 tests. When validation testing of the TLD VAX/1750A Ada Compiler System was performed, 19 tests had been withdrawn. The remaining 2380 tests were potentially applicable to this validation. The AVF determined that 463 tests were inapplicable to this implementation, and that the 1917 applicable tests were passed by the implementation.

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	68	864	916	15	10	44	1917
Failed	0	0	0	0	0	0	0
Inapplicable	1	3	452	2	3	2	463
Withdrawn	0	7	12	0	0	0	19
TOTAL	69	874	1380	17	13	46	2399

3.3 SUMMARY OF TEST RESULTS BY CHAPTER

RESULT	CHAPTER												TOTAL
	2	3	4	5	6	7	8	9	10	11	12	14	
Passed	93	205	280	244	159	97	135	261	124	32	218	69	1917
Failed	0	0	0	0	0	0	0	0	0	0	0	0	0
Inapplicable	23	120	140	3	2	0	4	1	6	0	0	164	463
Withdrawn	0	5	5	0	0	1	1	2	4	0	1	0	19
TOTAL	116	330	425	247	161	98	140	264	134	32	219	233	2399

3.4 WITHDRAWN TESTS

The following 19 tests were withdrawn from ACVC Version 1.8 at the time of this validation:

C32114A
B33203C
C34018A
C35904A
B37401A
C41404A
B45116A
C48008A
B49006A
B4A010C
B74101B
C87B50A
C92005A
C940ACA
CA3005A..D (4 tests)
BC3204C

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, 463 tests were inapplicable for the reasons indicated:

- C34001D, B52004E, B55B09D, and C55B07B use SHORT_INTEGER which is not supported by this compiler.

- C34001F and C35702A use SHORT_FLOAT which is not supported by this compiler.

- C34001G and C35702B use LONG_FLOAT which is not supported by this compiler.

- D64005F and D64005G are inapplicable because they make use of nested procedures as subunits to levels of 10 and 17 which exceed the capacity of the compiler.

- 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 TEXT_IO is made obsolete by this new definition in this implementation and the test cannot be executed since the package REPORT is dependent on the package TEXT_IO.

- C87B62B..C (2 tests) use length clauses which are not supported by this compiler. The length clauses 'STORAGE_SIZE for access types and 'SMALL are rejected during compilation.

- C96005B checks implementations for which the smallest and largest values in type DURATION are different from the smallest and largest values in DURATION's base type. This is not the case for this implementation.

- CA3004E, EA3004C, and LA3004A use INLINE pragma for procedures which is not supported by this compiler.

- CA3004F, EA3004D, and LA3004B use INLINE pragma for functions which is not supported by this compiler.

- The following 278 tests require a floating-point accuracy that exceeds the maximum of 6 supported by the implementation:

  C24113C..Y (23 tests)
  C35705C..Y (23 tests)
  C35706C..Y (23 tests)
  C35707C..Y (23 tests)
  C35708C..Y (23 tests)
  C35802C..Y (23 tests)
  C45241C..Y (23 tests)
  C45321C..Y (23 tests)
  C45421C..Y (23 tests)
  C45424C..Y (23 tests)
  C45521C..Z (24 tests)
  C45621C..Z (24 tests)

- The following 164 tests require the use of external files. This implementation supports only the files STANDARD_INPUT and STANDARD_OUTPUT:

CE2102C
CE2102G
CE2104A..D (4 tests)
CE2105A
CE2106A
CE2107A..F (6 tests)
CE2108A..D (4 tests)
CE2109A
CE2110A..C (3 tests)
CE2111A..E (5 tests)
CE2111G..H (2 tests)
CE2201A..F (6 tests)
CE2204A..B (2 tests)
CE2210A
CE2401A..F (6 tests)
CE2404A
CE2405B
CE2406A
CE2407A
CE2408A
CE2409A
CE2410A
AE3101A
CE3102B
EE3102C
CE3103A
CE3104A
CE3107A
CE3108A..B (2 tests)
CE3109A
CE3110A
CE3111A..E (5 tests)
CE3112A..B (2 tests)
CE3114A..B (2 tests)
CE3115A
CE3203A
CE3208A
CE3301A..C (3 tests)
CE3302A
CE3305A
CE3402A..D (4 tests)
CE3403A..C (3 tests)
CE3403E..F (2 tests)
CE3404A..C (3 tests)
CE3405A..D (4 tests)
CE3406A..D (4 tests)
CE3407A..C (3 tests)
CE3408A..C (3 tests)
CE3409A
CE3409C..F (4 tests)
CE3410A
CE3410C..F (4 tests)
CE3411A
CE3412A
CE3413A
CE3413C
CE3602A..D (4 tests
CE3603A
CE3604A
CE3605A..E (5 tests
CE3606A..B (2 tests
CE3704A..B (2 tests
CE3704D..F (3 tests
CE3704M..O (3 tests
CE3706D
CE3706F
CE3804A..E (5 tests
CE3804G
CE3804I
CE3804K
CE3804M
CE3805A..B (2 tests
CE3806A
CE3806D..E (2 tests
CE3905A..C (3 tests
CE3905L
CE3906A..C (3 tests
CE3906E..F (2 tests

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 subtests that can be processed.

Splits were required for three Class B tests:

B59001A B59001E B59001F

3.7 ADDITIONAL TESTING INFORMATION

3.7.1 Prevalidation

Prior to validation, a set of test results for ACVC Version 1.8 produced by the TLD VAX/1750A Ada Compiler System was submitted to the AVF by the applicant for review. Analysis of these results demonstrated that the compiler successfully passed all applicable tests, and that the compiler exhibited the expected behavior on all inapplicable tests.

3.7.2 Test Method

Testing of the TLD VAX/1750A Ada Compiler System using ACVC Version 1.8 was conducted on-site by a validation team from the AVF. The configuration consisted of a MicroVAX II host operating under MicroVMS, Version 4.5, and a TLD 1750A Instruction Level Simulator, Version 0.4.4 target operating under the TLD 1750A Single Program Kernel.

A magnetic tape, containing all tests except for withdrawn tests, tests requiring unsupported floating-point precisions, and tests requiring external files, 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 splits during the prevalidation testing were included in their split form on the magnetic tape. After reading the tape, all of the tests were grouped into larger tests by producing an enclosing procedure containing an Ada block statement for each ACVC test enclosed.

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 was compiled on the MicroVAX II, and all executable tests were linked and run on the TLD 1750A Instruction Level Simulator. Results were printed from the host computer.

The compiler was tested using command scripts provided by TLD Systems Ltd., and was reviewed by the validation team. All of the default options were in effect for testing except for the following:

<u>Option</u> <u>Effect</u>

NOCOD*GEN The NOCODEGEN switch causes the compiler to check syntax and update the Ada Program Library, but no code is generated. This option is only used for B tests.

LIS*T The LIST switch generates a listing file. The listing-file-spec can be completely, partially, or optionally specified. If only the listing file name is specified, the default listing file type, ".LIS", is utilized to form the listing-file-spec. If only the file type is specified, the file name of the input-file-spec is used to form the listing-file-spec. If no listing-file-spec is specified, the listing file name is formed from the file name of the input-file-spec and the default listing file type, ".LIS". This option was only used for B and E tests.

NOPHASETIME — The NOPHASETIME switch suppresses the output of the CPU time at the beginning of each compiler phase.

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

The validation team arrived at TLD Systems Ltd., Torrance CA on 19 June 1987, and departed after testing was completed on 22 June 1987.

APPENDIX A

DECLARATION OF CONFORMANCE

TLD Systems Ltd. has submitted the following declaration of conformance concerning the TLD VAX/1750A Ada Compiler System.

DELCARATION OF CONFORMANCE

Compiler Implementor: TLD Systems, Ltd.
Ada[©] Validation Facility: ASD/SCOL, Wright-Patterson AFB, OH
Ada Compiler Validation Capability (ACVC) Version: 1.8

Base Configuration

Base Compiler Name: TLD VAX/1750A Ada Compiler System Version: 1.0.0

Host Architecture ISA: MicroVAX II
OS&VER #: MicroVMS, Version 4.5
Target Architecture ISA: TLD 1750A Instruction Level Simulator
OS&VER #: TLD 1750A Single Program Kernel Version: 0.4.4

Implementor's Declaration

I, the undersigned, representing TLD Systems Ltd., have implemented no deliberate extensions to the Ada Language Standard ANSI/MIL-STD-1815A in the compiler listed in this declaration. I declare that TLD Systems Ltd. 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 the Ada language compiler listed in this declaration shall be made only in the owner's corporate name.

Date: 6/22/87

TLD Systems, Ltd.
Terry L. Dunbar, President

Owner's Declaration

I, the undersigned, representing TLD Systems Ltd., 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. I further agree to continue to comply with the Ada trademark policy, as defined by the Ada Joint Program Office. I 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.

Date: 6/22/87

TLD Systems Ltd.
Terry L. Dunbar, President

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

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 clauses. The implementation-dependent characteristics of the TLD VAX/1750A Ada Compiler System, Version 1.0.0, are described in the following sections which discuss topics in Appendix F of the Ada Language Reference Manual (ANSI/MIL-STD-1815A). Implementation-specific portions of the package STANDARD are also included in this appendix.

```
package STANDARD is

     ...

     type INTEGER is range -32768 .. 32767;
     type LONG_INTEGER is range -1_073_741_824 .. 1_073_741_823;

     type FLOAT is digits 6 range -1.0*2.0**127 .. 0.999999*2.0**127;
     type DURATION is delta 2.0**(-12) range -86_400.0 .. 86_400.0;

     ...

end STANDARD;
```

APPENDIX F

The Ada language definition allows for certain machine_dependencies in a controlled manner. No machine-dependent syntax of semantic extensions or restrictions are allowed. The only allowed implementation-dependencies correspond to implementaton-dependent pragmas and attributes, certain machine-dependent conventions as mentioned in chapter 13, and certain allowed restrictions on representation clauses.

The full definition of the implementation-dependent characteristics of the TLD VAX/1750A Ada Compiler System is presented in this Appendix F.

Implementation-Dependent Pragmas

The TLD ACS supports pragma identifiers Interface, its logical complement, Export; Compress; and integrated preprocessor commands in the form of pragma syntax: If, Elsif, Else, EndIf, and Include.

Pragma Export(Language, Name {, Optional_String});

This pragma is used to identify a static object name or procedure name that is to be exposed to the linker for reference by object modules written in other than the Ada Language. The third parameter is the name by which the Ada entity named by the second parameter may be referenced rather than a name assigned by the compiler. The only language supported at present is Assembly. If the entity named is a subprogram, this pragma must be placed in the declarative region of the subprogram. If the entity named is an Ada object, this pragma must appear following the declaration of the object but within the same declarative region as the object.

Pragma Compress (Subtype_Name);

This pragma is used to instruct the compiler to minimize the storage occupied by the indicated subtype. This pragma must occur following the declaration of the subtype but prior to any use of the subtype and in the same declarative region as the subtype declaration.

Pragma Include (File_Path_Name_String);

This directive in the form of a language pragma is processed by the source preprocessor to permit inclusion of another source file in place of the pragma. This pragma may occur any place a language defined pragma, statement, or declaration may occur. This directive is used to facilitate source program portability and configurability.

```
Pragma If ( Compile_Time_Expression );
Pragma Elsif (Compile_Time_Expression );
Pragma Else;
Pragma Endif;
```

These source preprocessor directives may be used to enclose conditionally compiled source to enhance program portability and configuration adaptation. These directives may occur at the place that language defined pragmas, statements, or declarations may occur. Source occurring following these pragmas will be compiled or ignored similar to the semantics of the corresponding Ada statements depending upon whether the compile time expression is true or false, respectively. The primary difference between these directives and the corresponding Ada statements are that the directives may enclose declarations and other pragmas.

Implementation-Dependent Attributes

None.

Package System

The following declarations are defined in package System:

```
     type name is (none, ns16000, vax, af1750, z8002, z8001, gould, pdp11,
          m68000, pe3200, caps, amdahl, i8086, i80286, i80386, z80000,
          ns32000, ibms1, m68020, nebula, name_x, hp);

     system_name: constant name := name'af1750;

     subtype priority is integer range 1..16#3FEE#; -- 1 is default priority.

     type address is range 0..65535 ;
        for address'size use 16 ;
     subtype unsigned is address ;

     -- Language Defined Constants

     storage_unit: constant := 16;
     memory_size:  constant := 65536;
     min_int:      constant := -2**31+1;
     max_int:      constant := 2**31-1;
     max_digits:   constant := 6;
     max_mantissa: constant := 31;
     fine_delta:   constant := 2.0**(-30);
     tick:         constant := 1.0/200.0;  -- Clock ticks are 5 msecs.
```

Representation Clause Restrictions

Pragma Pack is not supported.

Length clauses are supported for 'size applied to objects other than task and access type objects and denote the number of bits allocated to the object.

Length clauses are not supported for 'Storage_Size when applied to access types.

Length clauses are supported for 'Storage_Size when applied to a task type and denote the number of words of stack to be allocated to the task.

Length clauses are not supported for 'Small.

Enumeration representation clauses are supported for value ranges of Integer'First to Integer'Last.

Record representation clauses are supported to arrange record components within a record. Record components may not be specified to cross a word boundary unless they are arranged to encompass two or more whole words. A record component of type record that has itself been "rep specificationed" may only be allocated at bit 0. Bits are numbered from left to right with bit 0 indicating the sign bit.

The alignment clause is not supported.

Address clauses are supported for both variable and constant objects and designate the virtual address of the object. The TLD Ada Compiler System treats the address specification as a means to access objects allocated by other than Ada means and accordingly does not treat the clause as a request to allocate the object at the indicated address.

Address clauses are not supported for packages, tasks, or task entries.

Implementation-Generated Names

The TLD Ada Compiler System defines no implementation dependent names for compiler generated components.

Address Clause Expressions

Address expression values and type Address represent a location in logical memory, (the program's current address state). For objects, the address specifies a location within the 64K word logical operand space. The 'Address attribute applied to a subprogram represents a 16 bit word address within the logical instruction space.

Unchecked Conversion Restrictions

None

I/O Package Characteristics

The following implementation-defined types are declared in Text_Io.

```
     subtype Count is integer range 0 .. 511;

     subtype Field is Integer range 0 .. 127;
```

The implementation-defined types of package Standard are:

```
     type Integer is        range -32_768 .. 32_767;
     type Long_Integer is   range -1_073_741_824 .. 1_073_741_823;
     type Float is digits 6 range -1.0*2.0**127 .. 0.999999*2.0**127;
     type Duration is delta 2.0**(-12) range -86_400.0..86_400.0;
```

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.

Name and Meaning	Value
$BIG_ID1 Identifier the size of the maximum input line length with varying last character.	(1..119 => 'A', 120 => '1')
$BIG_ID2 Identifier the size of the maximum input line length with varying last character.	(1..119 => 'A', 120 => '2')
$BIG_ID3 Identifier the size of the maximum input line length with varying middle character.	(1..59|61..120 => 'A', 60 => '4')
$BIG_ID4 Identifier the size of the maximum input line length with varying middle character.	(1..59|61..120 => 'A', 60 => '4');
$BIG_INT_LIT An integer literal of value 298 with enough leading zeroes so that it is the size of the maximum line length.	(1..117 => '0', 118..120 => "298")

Name and Meaning	Value
$BIG_REAL_LIT A real literal that can be either of floating- or fixed-point type, has value 690.0, and has enough leading zeroes to be the size of the maximum line length.	(1..114 => '0', 115..120 => "69.0E1")
$BLANKS A sequence of blanks twenty characters fewer than the size of the maximum line length.	(1..100 => ' ')
$COUNT_LAST A universal integer literal whose value is TEXT_IO.COUNT'LAST.	511
$EXTENDED_ASCII_CHARS A string literal containing all the ASCII characters with printable graphics that are not in the basic 55 Ada character set.	"abcdefghijklmnopqrstuvwxyz" & "!$%?@[\]^`{}~"
$FIELD_LAST A universal integer literal whose value is TEXT_IO.FIELD'LAST.	127
$FILE_NAME_WITH_BAD_CHARS An illegal external file name that either contains invalid characters, or is too long if no invalid characters exist.	"X}]!@#$^&~Y"
$FILE_NAME_WITH_WILD_CARD_CHAR An external file name that either contains a wild card character, or is too long if no wild card character exists.	"XYZ*"
$GREATER_THAN_DURATION A universal real value that lies between DURATION'BASE'LAST and DURATION'LAST if any, otherwise any value in the range of DURATION.	524_287.5
$GREATER_THAN_DURATION_BASE_LAST The universal real value that is greater than DURATION'BASE'LAST, if such a value exists.	524_288.0

Name and Meaning	Value
$ILLEGAL_EXTERNAL_FILE_NAME1 An illegal external file name.	"BAD_CHARACTER*^"
$ILLEGAL_EXTERNAL FILE_NAME2 An illegal external file name that is different from $ILLEGAL_EXTERNAL_FILE_NAME1.	"THIS_FILENAME_IS_ONE_TOO_LONG_" & "FOR_A_FILE"
$INTEGER_FIRST The universal integer literal expression whose value is INTEGER'FIRST.	-32768
$INTEGER_LAST The universal integer literal expression whose value is INTEGER'LAST.	32767
$LESS_THAN_DURATION A universal real value that lies between DURATION'BASE'FIRST and DURATION'FIRST if any, otherwise any value in the range of DURATION.	-524_287.5
$LESS_THAN_DURATION_BASE_FIRST The universal real value that is less than DURATION'BASE'FIRST, if such a value exists.	-524 288.0
$MAX_DIGITS The universal integer literal whose value is the maximum digits supported for floating-point types.	6
$MAX_IN_LEN The universal integer literal whose value is the maximum input line length permitted by the implementation.	120
$MAX_INT The universal integer literal whose value is SYSTEM.MAX_INT.	2**31-1

Name and Meaning	Value
$NAME A name of a predefined numeric type other than FLOAT, INTEGER, SHORT_FLOAT, SHORT_INTEGER, LONG_FLOAT, or LONG_INTEGER if one exists, otherwise any undefined name.	LONG_LONG_INTEGER
$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#
$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. The following 19 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.

- C32114A: An unterminated string literal occurs at line 62.
- B33203C: The reserved word "IS" is misspelled at line 45.
- C34018A: The call of function G at line 114 is ambiguous in the presence of implicit conversions.
- C35904A: The elaboration of subtype declarations SFX3 and SFX4 may raise NUMERIC_ERROR instead of CONSTRAINT_ERROR as expected in the test.
- B37401A: The object declarations at lines 126 through 135 follow subprogram bodies declared in the same declarative part.
- C41404A: The values of 'LAST and 'LENGTH are incorrect in the if statements from line 74 to the end of the test.
- B45116A: ARRPRIBL1 and ARRPRIBL2 are initialized with a value of the wrong type--PRIBOOL_TYPE instead of ARRPRIBOOL_TYPE--at line 41.
- C48008A: The assumption that evaluation of default initial values occurs when an exception is raised by an allocator is incorrect according to AI-00397.
- B49006A: Object declarations at lines 41 and 50 are terminated incorrectly with colons, and end case; is missing from line 42.
- B4A010C: The object declaration in line 18 follows a subprogram body of the same declarative part.

- B74101B: The begin at line 9 causes a declarative part to be treated as a sequence of statements.

- C87B50A: The call of "/=" at line 31 requires a use clause for package A.

- C92005A: The "/=" for type PACK.BIG_INT at line 40 is not visible without a use clause for the package PACK.

- C940ACA: The assumption that allocated task TT1 will run prior to the main program, and thus assign SPYNUMB the value checked for by the main program, is erroneous.

- CA3005A..D (4 tests): No valid elaboration order exists for these tests.

- BC3204C: The body of BC3204C0 is missing.

END
DATE
FILMED
5-88
DTIC