UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE	READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER 2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) **Ada Compiler Validation Summary Report:** SofTech, Inc., AdaDIPS, Version 1.0, NTT DIPS V20 (Host & Target) 890901S1.10132	5. TYPE OF REPORT & PERIOD COVERED 01 Sept. 1989 to 01 Dec. 1990 6. PERFORMING ORG. REPORT NUMBER
AD-A219 439	
7. AUTHOR(s) National Institute of Standards and Technology Gaithersburg, Maryland, USA	8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION AND ADDRESS National Institute of Standards and Technology Gaithersburg, Maryland, USA	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Ada Joint Program Office United States Department of Defense Washington, DC 20301-3081	12. REPORT DATE 13. NUMBER OF PAGES
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office) National Institute of Standards and Technology Gaithersburg, Maryland, USA	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)

SofTech, Inc., AdaDIPS, Version 1.0, Gaithersburg, Maryland, NTT DIPS V20 under DIPS-UX (VO), ACVC 1.10

DD FORM 1473 1 JAN 73 EDITION OF 1 NOV 65 IS OBSOLETE S/N 0102-LF-014-6601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

AVF Control Number: NIST89VSOF520
DATE COMPLETED PRE-VAL 07-12-89
DATE COMPLETED ON-SITE 09-01-89
DATE MODIFIED PER AVO COMMENTS:
10-09-89
DATE MODIFIED PER AVO COMMENTS:
12-28-89

Ada COMPILER
VALIDATION SUMMARY REPORT:
Certificate Number: 890901S1.10132
OWNER: NIPPON TELEGRAPH and TELEPHONE CORPORATION
IMPLEMENTOR: SofTech, Inc.
AdaDIPS, Version 1.0
NTT DIPS V20 Host and NTT DIPS V20 Target

Completion of On-Site Testing:
September 1, 1989

Prepared By:
Software Standards Validation Group
National Computer Systems Laboratory
National Institute of Standards and Technology
Building 225, Room A266
Gaithersburg, Maryland 20899

Prepared For:
Ada Joint Program Office
United States Department of Defense
Washington DC 20301-3081

AVF Control Number: NIST89VSOF520

Ada Compiler Validation Summary Report:

Compiler Name: AdaDIPS, Version 1.0

Certificate Number: 890901S1.10132

Host: NTT DIPS V20 under DIPS-UX (VO)

Target: NTT DIPS V20 under DIPS-UX (VO)

Testing Completed September 1, 1989 Using ACVC 1.10

This report has been reviewed and is approved.

Ada Validation Facility
Dr. David K. Jefferson
Chief, Information Systems
Engineering Division
National Computer Systems
Laboratory (NCSL)
National Institute of
Standards and Technology
Building 225, Room A266
Gaithersburg, MD 20899

Ada Validation Facility
Mr. L. Arnold Johnson
Manager, Software Standards
Validation Group
Engineering Division
National Computer Systems
Laboratory (NCSL)
National Institute of
Standards and Technology
Building 225, Room A266
Gaithersburg, MD 20899

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

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

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

1.1 PURPOSE OF THIS VALIDATION SUMMARY REPORT

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

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

On-site testing was completed September 1, 1989 at Waltham, MA..

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:

Software Standards Validation Group
National Computer Systems Laboratory
National Institute of Standards and Technology
Building 225, Room A266
Gaithersburg, Maryland 20899

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

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

1.3 REFERENCES

1. Reference Manual for the Ada Programming Language, ANSI/MIL-STD-1815A, February 1983 and ISO 8652-1987.
2. Ada Compiler Validation Procedures and Guidelines, Ada Joint Program Office, 1 January 1987.
3. Ada Compiler Validation Capability Implementers' Guide, SofTech, Inc., December 1986.
4. Ada Compiler Validation Capability User's Guide, December 1986.

1.4 DEFINITION OF TERMS

ACVC	The Ada Compiler Validation Capability. The set of Ada programs that tests the conformity of an Ada compiler to the Ada programming language.
Ada	An Ada Commentary contains all information relevant to the Commentary point addressed by a comment on the Ada Standard. These comments are given a unique identification number having the form AI-ddddd.
Ada Standard	ANSI/MIL-STD-1815A, February 1983 and ISO 8652-1987.
Applicant	The agency requesting validation.
AVF	The Ada Validation Facility. The AVF is responsible for conducting compiler validations according to procedures contained in the Ada Compiler Validation Procedures and Guidelines.
AVO	The Ada Validation Organization. The AVO has oversight authority over all AVF practices for the purpose of maintaining a uniform process for validation of Ada compilers. The AVO provides administrative and technical support for Ada validations to ensure consistent practices.
Compiler	A processor for the Ada language. In the context of

this report, a compiler is any language processor, including cross-compilers, translators, and interpreters.

Failed test — An ACVC test for which the compiler generates a result that demonstrates nonconformity to the Ada Standard.

Host — The computer on which the compiler resides.

Inapplicable test — An ACVC test that uses features of the language that a compiler is not required to support or may legitimately support in a way other than the one expected by the test.

Passed test — An ACVC test for which a compiler generates the expected result.

Target — The computer which executes the code generated by the compiler.

Test — A program that checks a compiler's conformity regarding a particular feature or a combination of features to the Ada Standard. In the context of this report, the term is used to designate a single test, which may comprise one or more files.

Withdrawn — An ACVC test found to be incorrect and not used to check test conformity to the Ada Standard. A test may be incorrect because it has an invalid test objective, fails to meet its test objective, or contains illegal or erroneous use of the language.

1.5 ACVC TEST CLASSES

Conformity to the Ada Standard is measured using the ACVC. The ACVC contains both legal and illegal Ada programs structured into six test classes: A, B, C, D, E, and L. The first letter of a test name identifies the class to which it belongs. Class A, C, D, and E tests are executable, and special program units are used to report their results during execution. Class B tests are expected to produce compilation errors. Class L tests are expected to produce errors because of the way in which a program library is used at link time.

Class A tests ensure the successful compilation and execution of legal Ada programs with certain language constructs which cannot be verified at run time. There are no explicit program components in a Class A test to check semantics. For example, a Class A test checks that reserved words of another language (other than those already reserved in the Ada language) are not treated as reserved words by an Ada compiler. A Class A test is passed if no errors are detected at compile time and the

program executes to produce a PASSED message.

Class B tests check that a compiler detects illegal language usage. Class B tests are not executable. Each test in this class is compiled and the resulting compilation listing is examined to verify that every syntax or semantic error in the test is detected. A Class B test is passed if every illegal construct that it contains is detected by the compiler.

Class C tests check the run time system to ensure 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.

Class E tests are expected to execute successfully and check implementation-dependent options and resolutions of ambiguities in the Ada Standard. 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. In some cases, an implementation may legitimately detect errors during compilation of the test.

Two library units, the package REPORT and the procedure CHECK_FILE, support the self-checking features of the executable tests. The package REPORT provides the mechanism by which executable tests report PASSED, FAILED, or NOT APPLICABLE results. It also provides a set of identity functions used to defeat some compiler optimizations allowed by the Ada Standard that would circumvent a test objective. The procedure CHECK_FILE is used to check the contents of text files written by some

of the Class C tests for Chapter 14 of the Ada Standard. The operation of REPORT and CHECK_FILE is checked by a set of executable tests. These tests produce messages that are examined to verify that the units are operating correctly. If these units are not operating correctly, then the validation is not attempted.

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

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

CHAPTER 2

CONFIGURATION INFORMATION

2.1 CONFIGURATION TESTED

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

Compiler: AdaDIPS, Version 1.0

ACVC Version: 1.10

Certificate Number: 890901S1.10132

Host Computer:

Machine: NTT DIPS V20

Operating System: DIPS-UX (VO)

Memory Size: 16MBytes

Target Computer:

Machine: NTT DIPS V20

Operating System: DIPS-UX (VO)

Memory Size: 16MBytes

2.2 IMPLEMENTATION CHARACTERISTICS

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

a. Capacities.

(1) The compiler correctly processes a compilation containing 723 variables in the same declarative part. (See test D29002K.)

(2) The compiler correctly processes tests containing loop statements nested to 33 levels. (See tests D55A03A..H (8 tests).)

(3) The compiler rejects tests containing block statements nested to 65 levels. (See test D56001B.)

(4) The compiler correctly processes tests containing recursive procedures separately compiled as subunits nested to 10 levels. (See tests D64005E..G (3 tests).)

b. Universal integer calculations.

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

c. Predefined types.

(1) This implementation supports the additional predefined types SHORT_INTEGER, LONG_INTEGER, SHORT_FLOAT in the package STANDARD. (See tests B86001T..Z (7 tests).)

d. Expression evaluation.

The order in which expressions are evaluated and the time at which constraints are checked are not defined by the language. While the ACVC tests do not specifically attempt to determine the order of evaluation of expressions, test results indicate the following:

(1) All of the default initialization expressions for record

components are evaluated before any value is checked for membership in a component's subtype. (See test C32117A.)

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

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

(4) NUMERIC_ERROR is raised for pre-defined integer comparison and for pre-defined integer membership. NO EXCEPTION is raised for large_int comparison or for large_int membership. NUMERIC_ERROR is raised for small_int comparison and for small_int membership when an integer literal operand in a comparison or membership test is outside the range of the base type. (See test C45232A.)

(5) NUMERIC_ERROR is raised when a literal operand in a fixed-point comparison or membership test is outside the range of the base type. (See test C45252A.)

(6) Underflow is not gradual. (See tests C45524A..J (10 tests).)

e. Rounding.

The method by which values are rounded in type conversions is not defined by the language. While the ACVC tests do not specifically attempt to determine the method of rounding, the test results indicate the following:

(1) The method used for rounding to integer is round away from zero. (See tests C46012A..J (10 tests).)

(2) The method used for rounding to longest integer is round away from zero. (See tests C46012A..J (10 tests).)

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

f. Array types.

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

(1) Declaration of an array type or subtype declaration with more than SYSTEM.MAX_INT components raises NUMERIC_ERROR. (See test C36003A.)

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

(3) NUMERIC_ERROR is raised when 'LENGTH is applied to an array type with SYSTEM.MAX_INT + 2 components. (See test C36202B.)

(4) A packed BOOLEAN array having a 'LENGTH exceeding INTEGER'LAST raises no exception. (See test C52103X.)

(5) A packed two-dimensional BOOLEAN array with more than INTEGER'LAST components raises CONSTRAINT_ERROR when the length of a dimension is calculated and exceeds INTEGER'LAST. (See test C52104Y.)

(6) 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 no exception. (See test E52103Y.)

(7) In assigning one-dimensional array types, the expression is 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.)

(8) In assigning two-dimensional array types, the expression is not 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.)

g. Discriminated types.

(1) 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.)

(2) In assigning record types with discriminants, the expression is 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.)

h. Aggregates.

(1) In the evaluation of a multi-dimensional aggregate, the test results indicate that all choices are evaluated before checking against the index type. (See tests C43207A and C43207B.)

(2) In the evaluation of an aggregate containing subaggregates, all choices are evaluated before being checked for identical bounds. (See test E43212B.)

(3) All choices were not evaluated before CONSTRAINT_ERROR is raised when a bound in a non-null range of a non-null aggregate does not belong to an index subtype. (See test E43211B.)

i. Pragmas.

(1) The pragma INLINE is supported for functions or procedures. (See tests LA3004A..B (2 tests), EA3004C..D (2 tests), and CA3004E..F (2 tests).)

j. Generics.

(1) Generic specifications and bodies can be compiled in separate compilations. (See tests CA1012A, CA2009C, CA2009F, BC3204C, and BC3205D.)

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

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

(4) Generic library subprogram specifications and bodies can be compiled in separate compilations. (See test CA1012A.)

(5) Generic non-library subprogram bodies can be compiled in separate compilations from their stubs. (See test CA2009F.)

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

(7) Generic library package specifications and bodies can be compiled in separate compilations. (See tests BC3204C and BC3205D.)

(8) Generic non-library package bodies as subunits can be compiled in separate compilations. (See test CA2009C.)

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

k. Input and output.

(1) The package SEQUENTIAL_IO cannot be instantiated with unconstrained array types or record types with discriminants without defaults. (See tests AE2101C, EE2201D, and EE2201E.)

(2) The package DIRECT_IO cannot be instantiated with unconstrained array types or record types with discriminants without defaults. (See tests AE2101H, EE2401D, and EE2401G.)

(3) Mode IN_FILE is supported for the operation of CREATE for SEQUENTIAL_IO. (See test CE2102D.)

(4) Mode IN_FILE is supported for the operation of CREATE for DIRECT_IO. (See test CE2102I.)

(5) Mode IN_FILE is supported for the operation of CREATE for text files. (See test CE3102E.)

(6) Modes IN_FILE and OUT_FILE are supported for text files. (See tests CE3102E and CE3102I..K (3 tests).)

(7) RESET and DELETE operations are supported for SEQUENTIAL_IO. (See tests CE2102G and CE2102X.)

(8) RESET and DELETE operations are supported for DIRECT_IO. (See tests CE2102K and CE2102Y.)

(9) RESET and DELETE operations are supported for text files. (See tests CE3102F..G (2 tests), CE3104C, CE3110A, and CE3114A.)

(10) Overwriting to a sequential file truncates to the last element written. (See test CE2208B.)

(11) Temporary sequential files are given names and not deleted when closed. (See test CE2108A.)

(12) Temporary direct files are given names and not deleted when closed. (See test CE2108C.)

(13) Temporary text files are given names and not deleted when closed. (See test CE3112A.)

(14) More than one internal file can be associated with a single

external file for sequential files when reading only. (See tests CE2107A..E (5 tests), CE2102L, CE2110B, and CE2111D.)

(15) More than one internal file can be associated with a single external file for direct files when reading only. (See tests CE2107F..H (3 tests), CE2110D and CE2111H.)

(16) More than one internal file can be associated with a single external file for text files when reading only. (See tests CE3111A..E (5 tests), CE3114B, and CE3115A.)

CHAPTER 3

TEST INFORMATION

3.1 TEST RESULTS

Version 1.10 of the ACVC comprises 3717 tests. When this compiler was tested, 44 tests had been withdrawn because of test errors. The AVF determined that 466 tests were inapplicable to this implementation. All inapplicable tests were processed during validation testing except for 215 executable tests that use floating-point precision exceeding that supported by the implementation. Modifications to the code, processing, or grading for 33 tests were required to successfully demonstrate the test objective. (See section 3.6.)

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

3.2 SUMMARY OF TEST RESULTS BY CLASS

RESULT	TEST CLASS						TOTAL
	A	B	C	D	E	L	
Passed	126	1133	1868	13	21	46	3207
Inapplicable	3	5	447	4	7	0	466
Withdrawn	1	2	35	0	6	0	44
TOTAL	130	1140	2350	17	34	46	3717

3.3 SUMMARY OF TEST RESULTS BY CHAPTER

RESULT	CHAPTER													TOTAL
	2	3	4	5	6	7	8	9	10	11	12	13	14	
Passed	194	571	547	245	171	99	160	332	137	36	252	184	279	3207
Inapplicable	18	78	133	3	1	0	6	0	0	0	0	185	42	466
Wdrn	1	1	0	0	0	0	0	2	0	0	1	35	4	44
TOTAL	213	650	680	248	172	99	166	334	137	36	253	404	325	3717

3.4 WITHDRAWN TESTS

The following 44 tests were withdrawn from ACVC Version 1.10 at the time of this validation:

A39005G	B97102E	C97116A	BC3009B	CD2A62D	CD2A63A
CD2A63B	CD2A63C	CD2A63D	CD2A66A	CD2A66B	CD2A66C
CD2A66D	CD2A73A	CD2A73B	CD2A73C	CD2A73D	CD2A76A
CD2A76B	CD2A76C	CD2A76D	CD2A81G	CD2A83G	CD2A84M
CD2A84N	CD2B15C	CD2D11B	CD5007B	CD5011O	CD7105A
CD7203B	CD7204B	CD7205C	CD7205D	CE2107I	CE3111C
CE3301A	CE3411B	E28005C	ED7004B	ED7005C	ED7005D
ED7006C	ED7006D				

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, 466 tests were inapplicable for the reasons indicated:

a. The following 215 tests are not applicable because they have floating-point type declarations requiring more digits than SYSTEM.MAX_DIGITS:

C24113K..Y (15 tests)	C35705K..Y (15 tests)
C35706K..Y (15 tests)	C35707K..Y (15 tests)

C35708K..Y (15 tests)	C35802K..Z (16 tests)
C45241K..Y (15 tests)	C45321K..Y (15 tests)
C45421K..Y (15 tests)	C45521K..Z (16 tests)
C45524K..Z (16 tests)	C45621K..Z (16 tests)
C45641K..Y (15 tests)	C46012K..Z (16 tests)

b. C24113H, C24113I, C24113J (3 tests) are not applicable because this implementation supports a line length of 120 characters.

c. C35702B and B86001U are not applicable because this implementation supports no predefined type LONG_FLOAT.

d. A39005C, C87B62B, CD1009J, CD1009R, CD1009S, CD1C03C, CD2A83A, CD2A83B, CD2A83C, CD2A83E, CD2A83F, CD2A84B, CD2A84C, CD2A84D, CD2A84E, CD2A84F, CD2A84G, CD2A84H, CD2A84I, CD2A84K, CD2A84L, CD2B11B, CD2B11C, CD2B11D, CD2B11E, CD2B11F, CD2B11G, CD2B15B, CD2B16A, ED2A86A (30 test) are not applicable because this implementation does not allow representation specifications for 'STORAGE_SIZE on access types.

e. C45531M..P (4 tests), C45532M..P (4 tests) are not applicable because this implementation does not support a 48 bit integer machine size.

f. D55A03G, D55A03H, D56001B, D64005G are not applicable because they exceed capabity limitations of the compiler.

g. C86001F is not applicable because, for this implementation, the package TEXT_IO is dependent upon package SYSTEM. This test recompiles package SYSTEM, making package TEXT_IO, and hence package REPORT, obsolete.

h. B86001X, C45231D, and CD7101G (3 tests) are not applicable because this implementation does not support any predefined integer type with a name other than INTEGER, LONG_INTEGER, or SHORT_INTEGER.

i. B86001Y is not applicable because this implementation supports no predefined fixed-point type other than DURATION.

j. B86001Z is not applicable because this implementation supports no predefined floating-point type with a name other than FLOAT, LONG_FLOAT, or SHORT_FLOAT.

k. BD5002A is not applicable because this implementation does not process address clauses.

l. CD1009C, CD2A41A, CD2A41B, CD2A41C, CD2A41D, CD2A41E, CD2A42A, CD2A42B, CD2A42C, CD2A42D, CD2A42E, CD2A42F, CD2A42G, CD2A42H, CD2A42I, CD2A42J (16 tests) are not applicable because this implementation does not support 'SIZE representations for floating-

point types.

m. CD1009D, CD1009Q, CD1C04C, CD2A51A, CD2A51B, CD2A51C, CD2A51D, CD2A51E, CD2A52A, CD2A52B, CD2A52C, CD2A52D, CD2A52G, CD2A52H, CD2A52I, CD2A52J, CD2A53A, CD2A53B, CD2A53C, CD2A53D, CD2A53E, CD2A54A, CD2A54B, CD2A54C, CD2A54D, CD2A54G, CD2A54H, CD2A54I, CD2A54J, ED2A56A (30 tests) are not applicable because this implementation does not support 'SIZE representations for fixed-point types.

n. CD1009N, CD1009X, CD1009Y, CD1009Z, CD1C04E, ED1D04A (6 tests) are not applicable because this implementation does not support the overlapping of record components.

o. CD2A61B, CD2A61D, CD2A61F, CD2A61H, CD2A61I, CD2A61J, CD2A61L, CD2A62B (8 tests) are not applicable because this implementation does not support size specifications for array types that imply compression of component storage.

p. CD2A71B, CD2A71D, CD2A72B, CD2A72D (4 tests) are not applicable because this implementation does not support the 'SIZE specification for record types implying compression of component storage.

q. CD2A81A, CD2A81B, CD2A81C, CD2A81D, CD2A81E, CD2A81F, CD2A87A, (7 tests) are not applicable because this implementation does not support the 'SIZE representation clauses for access types.

r. CD2A91A, CD2A91B, CD2A91C, CD2A91D, CD2A91E (5 tests) are not applicable because this implementation does not support the 'SIZE representation clauses for task types.

s. CD4051C, CD4051D, CD1C03H (3 tests) are not applicable because this implementation does not support a 2 byte boolean. This implementation uses a 32 bit size for its boolean.

t. CD5003B, CD5003C, CD5003D, CD5003E, CD5003F, CD5003G, CD5003H, CD5003I, CD5011A, CD5011C, CD5011E, CD5011G, CD5011I, CD5011K, CD5011M, CD5011Q, CD5012A, CD5012B, CD5012E, CD5012F, CD5012I, CD5012J, CD5012M, CD5013A, CD5013C, CD5013E, CD5013G, CD5013I, CD5013K, CD5013M, CD5013O, CD5013S, CD5014A, CD5014C, CD5014E, CD5014G, CD5014I, CD5014K, CD5014M, CD5014O, CD5014S, CD5014T, CD5014V, CD5014X, CD5014Y, CD5014Z (46 tests) are not applicable because this implementation does not support 'ADDRESS clauses for variables.

u. CD5011B, CD5011D, CD5011F, CD5011H, CD5011L, CD5011N, CD5011R, CD5011S, CD5012C, CD5012D, CD5012G, CD5012H, CD5012L, CD5013B, CD5013D, CD5013F, CD5013H, CD5013L, CD5013N, CD5013R, CD5014B, CD5014D, CD5014F, CD5014H, CD5014J, CD5014L, CD5014N, CD5014R, CD5014U, CD5014W (30 tests) are not applicable because this implementation does not support 'ADDRESS clauses for constants.

v. AE2101C, EE2201D, and EE2201E use instantiations of package SEQUENTIAL_IO with unconstrained array types and record types with discriminants without defaults. These instantiations are rejected by this compiler.

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

x. CE2102D is inapplicable because this implementation supports CREATE with IN_FILE mode for SEQUENTIAL_IO.

y. CE2102E is inapplicable because this implementation supports CREATE with OUT_FILE mode for SEQUENTIAL_IO.

z. CE2102F is inapplicable because this implementation supports CREATE with INOUT_FILE mode for DIRECT_IO.

aa. CE2102I is inapplicable because this implementation supports CREATE with IN_FILE mode for DIRECT_IO.

ab. CE2102J is inapplicable because this implementation supports CREATE with OUT_FILE mode for DIRECT_IO.

ac. CE2102N is inapplicable because this implementation supports OPEN with IN_FILE mode for SEQUENTIAL_IO.

ad. CE2102O is inapplicable because this implementation supports RESET with IN_FILE mode for SEQUENTIAL_IO.

ae. CE2102P is inapplicable because this implementation supports OPEN with OUT_FILE mode for SEQUENTIAL_IO.

af. CE2102Q is inapplicable because this implementation supports RESET with OUT_FILE mode for SEQUENTIAL_IO.

ag. CE2102R is inapplicable because this implementation supports OPEN with INOUT_FILE mode for DIRECT_IO.

ah. CE2102S is inapplicable because this implementation supports RESET with INOUT_FILE mode for DIRECT_IO.

ai. CE2102T is inapplicable because this implementation supports OPEN with IN_FILE mode for DIRECT_IO.

aj. CE2102U is inapplicable because this implementation supports RESET with IN_FILE mode for DIRECT_IO.

ak. CE2102V is inapplicable because this implementation supports OPEN with OUT_FILE mode for DIRECT_IO.

al. CE2102W is inapplicable because this implementation supports RESET with OUT_FILE mode for DIRECT_IO.

am. CE2107B..D (3 tests), CE2107G, CE2110B, CE2110D, CE2111D, CE2111H, CE3111B, CE3111D..E (2 tests), CE3114B, and CE3115A are not applicable because multiple internal text, direct, or sequential files cannot be associated with the same external file for sequential files. The proper exception is raised when multiple access is attempted.

an. CE2107E, CE2107L are not applicable because this implementation does not support the association of an internal sequential file and an internal direct access file to a single external file.

ao. CE3102E is inapplicable because text file CREATE with IN_FILE mode is supported by this implementation.

ap. CE3102F is inapplicable because text file RESET is supported by this implementation.

aq. CE3102G is inapplicable because text file deletion of an external file is supported by this implementation.

ar. CE3102I is inapplicable because text file CREATE with OUT_FILE mode is supported by this implementation.

as. CE3102J is inapplicable because text file OPEN with IN_FILE mode is supported by this implementation.

at. CE3102K is inapplicable because text file OPEN with OUT_FILE mode is not supported by this implementation.

3.6 TEST, PROCESSING, AND EVALUATION MODIFICATIONS

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

Modifications were required for 33 tests.

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

B2A003A	B33201C	B33202C	B33203C	B33301B	B37106A	B37201A
B37301I	B38003A	B38003B	B38009A	B38009B	B44001A	B44004A
B54A01L	B91001H	B95063A	BB1006B	BC1102A	BC1109A	BC1109B
BC1109C	BC1109D	BC1201F	BC1201G	BC1201H	BC1201I	BC1201J
BC1201L	BC3013A					

The following tests were modified in accordance with an AVO ruling for each:

C96001A had two lines of code modified as follows:

```
-- THE ORIGINAL CODE FOLLOWS:
-- WITH SYSTEM;  USE SYSTEM;

-- MODIFIED CODE REPLACING ABOVE LINE OF CODE
   WITH SYSTEM;

-- THE ORIGINAL CODE FOLLOWS:
-- IF TICK < DURATION'SMALL THEN;

-- MODIFIED CODE REPLACING ABOVE LINE OF CODE
   IF SYSTEM.TICK < DURATION'SMALL THEN;
```

CD2C11A and CD2C11B had the same lines of code modified as follows:

```
-- THE ORIGINAL CODE FOLLOWS:
-- FOR TTYPE'STORAGE_SIZE USE 1024;  --N/A => ERROR

-- MODIFIED CODE REPLACING ABOVE LINE OF CODE
-- FOR TTYPE'STORAGE_SIZE USE 1896;  --N/A => ERROR
```

3.7 ADDITIONAL TESTING INFORMATION

3.7.1 Prevalidation

Prior to validation, a set of test results for ACVC Version 1.10 produced by the AdaDIPS, Version 1.0 was submitted to the AVF by the applicant for review. Analysis of these results demonstrated that the compiler successfully passed all applicable tests, and the compiler exhibited the expected behavior on all inapplicable tests.

3.7.2 Test Method

Testing of the AdaDIPS, Version 1.0 using ACVC Version 1.10 was conducted on-site by a validation team from the AVF. The configuration in which the testing was performed is described by the following designations of hardware and software components:

Host computer:	NTT DIPS V20
Host operating system:	DIPS-UX (VO)
Target computer:	NTT DIPS V20

Target operating system:	DIPS-UX (V0)
Compiler:	AdaDIPS, Version 1.0
Linker:	lnkunix, Version 1.0
Loader/Downloader:	expunix, Versi n 1.0

A magnetic tape containing all tests except for withdrawn tests was taken on-site by the validation team for processing. Tests that make use of implementation-specific values were customized before being written to the magnetic tape. Tests requiring modifications during the prevalidation testing were modified on-site.

TEST INFORMATION

The contents of the magnetic tape were loaded directly onto a VAX 785 where the B tests were split, the 3 C tests were modified, all .DEP and .TST tests name extensions were change to .ADA. The modified tests were then written to a tar tape and loaded onto the host computer, the NTT DIPS V20 computer.

After the test files were loaded to disk, the full set of tests was compiled, linked, and all executable tests were run on the NTT DIPS V20. Results were written to tar tape and loaded onto the VAX 785 computer where the results were printed.

The compiler was tested using command scripts provided by SofTech, Inc. and reviewed by the validation team. See Appendix E for a complete listing of the compiler options for this implementation. The compiler options invoked during this test were:

Options specified in testing command scripts:

/SOURCE (*) /NO_SOURCE (**) /NO_OPTIMIZE/NO_TRACE_BACK/NO_ATTRIBUTE

* SOURCE used for B-TESTS, E-TESTS, and all not applicable tests.
** NO_SOURCE used for all other tests.

Default Options:

PRIVATE NO_NOTES NO_ATTRIBUTE NO_XREF NO_MACHINE NO_READ_ONLY_DATA NO_READ_WRITE_DATA NO_STACK_DATA NO_DIAGNOSTICS NO_SUMMARY NO_SOURCE NO_COMPILER_MAINT NO_STANDARD_COMPILE NO_SYSTEM_COMPILE NO_RSL_COMPILE NO_BIS_COMPILE CHECKS CODE_ON_WARNING CONTAINER_GENERATION TRACE_BACK NO_FREQUENCY NO_KANA NO_OBJECT_ONLY NO_OPTIMIZE NO_SFD_OPT

Tests were compiled, linked, and executed (as appropriate) using a single computer. Test output, compilation listings, and job logs were captured on magnetic tape and archived at the AVF.

3.7.3 Test Site

Testing was conducted at Waltham, MA. and was completed on September 1,

1989.

APPENDIX A

DECLARATION OF CONFORMANCE

SofTech, Inc. has submitted the following Declaration of Conformance concerning the AdaDIPS, Version 1.0.

DECLARATION OF CONFORMANCE

Compiler Implementer: SofTech, Inc.

Ada Validatic Facility:

Ada Validation Facility
National Computer Systems Laboratory (NCSL)
National Institute of Standards and Technology
Building 225, Room A266
Gaithersburg, MD 20899

Ada Compiler Validation Capability (ACVC) Version: 1.10

Base Configuration

Base Compiler Name:	AdaDIPS, Version 1.0
Host Architecture:	NTT DIPS V20
Host OS and Version:	DIPS-UX (V0)
Target Architecture:	NTT DIPS V20
Target OS and Version:	DIPS-UX (V0)

Implementer's Declaration

I, the undersigned, representing SOFTECH, INC. have implemented no deliberate extensions to the Ada Language Standard ANSI/MIL-STD-1815A in the compiler(s) listed in this declaration. I declare that NIPPON TELEGRAPH and TELEPHONE CORPORATION is the owner of record of the Ada language compiler(s) listed above and, as such, is responsible for maintaining said compiler(s) in conformance to ANSI/MIL-STD-1815A. All certificates and registrations for the Ada language compiler(s) listed in this declaration shall be made only in the corporate name of the owner.

_____, _____
IMPLEMENTER COMPANY OFFICIAL, IMPLEMENTER OFFICIAL TITLE

Date: 31 Aug 1989

Owner's Declaration

I, the undersigned, representing NIPPON TELEGRAPH and TELEPHONE CORPORATION 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 declare that the Ada language compiler and its host/target performance is in compliance with the Ada Language Standard ANSI/MIL-STD-1815A.

S. Hamata, Executive Manager

OWNER COMPANY OFFICIAL, OWNER OFFICIAL TITLE

Date: 24 May 89

APPENDIX B

APPENDIX F OF THE Ada STANDARD

The only allowed implementation dependencies correspond to implementation-dependent pragmas, to certain machine-dependent conventions as mentioned in chapter 13 of the Ada Standard, and to certain allowed restrictions on representation clauses. The implementation-dependent characteristics of the AdaDIPS, Version 1.0, as described in this Appendix, are provided by SofTech, Inc.. Unless specifically noted otherwise, references in this appendix are to compiler documentation and not to this report. Implementation-specific portions of the package STANDARD, which are not a part of Appendix F, are:

```
package STANDARD is

...

 type INTEGER is range -32_768 .. 32_767;
 type SHORT_INTEGER is range -128 .. 127
 type LONG_INTEGER is range -2_147_483_648 .. 2_147_483_647;

 type FLOAT is digits 14 range
           -(16#FFFF_FFFF_FFFF#E63) .. (16#FFFF_FFFF_FFFF#E63);
 type SHORT_FLOAT is digits 6 range
           -(16#0.FFFF_F8#E63) .. (16#0.FFFF_F8#E63);

 type DURATION is delta 2.0 ** (14) range -86_400.0 .. 86_400.0;

...

end STANDARD;
```

APPENDIX F

NTT's AdaDIPS Pre-Validation Material

APPENDIX F of the Ada Language Reference Manual (ANSI/MIL-STD-1815A).

The implementation-dependent characteristics 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). One other section, package STANDARD is also included in this appendix.

---

(1) Implementation-Dependent Pragmas

The following are implementation-defined pragmas:

pragma TITLE :

Takes a character string as the single argument. This is a listing control pragma. The argument string will appear on the second line of each page of listing produced for the compilation. At most one pragma TITLE may appear for any compilation, and it must be the first statement in the declarative part of the unit (comments and other pragmas excepted).

pragma SUPPRESS_ALL :

Takes no arguments. Suppresses all checking for the compilation.

pragma TEXTUAL_ORDER :

Takes no arguments. Forces the compiler to lay out any record types defined after this pragma to be laid out in textual order. Otherwise, the compiler may rearrange components to reduce the amount of space required. It may only appear immediately within a declarative part or package specification before any basic declarative part.

pragma NO_TEXTUAL_ORDER :

Takes no arguments. Informs the compiler that it may rearrange components in record types defined after this pragma in order to reduce the amount of storage required. It may only appear immediately within a declarative part or package specification before any basic declarative items.

The following are effects of implementation-defined pragmas:

If a pragma appears before any compilation unit in a compilation, it will affect all following compilation units, as specified below, and in the Ada Reference Manual.

If a pragma appears inside a compilation unit, it will be associated with that compilation unit, and in listings associated with that compilation unit.

If a pragma follows a compilation unit, it will be associated with the preceding compilation unit, and the effects of the pragma will be found in the container of that compilation unit, and in listings associated with that container.

The following notes specify the language-required definitions of the predefined pragmas. Pragmas CONTROLLED, ELABORATE, INLINE, LIST, PACK, PAGE, and SHARED require no implementation-dependent notes.

pragma INTERFACE :

The following language names will be recognized and implemented:

DIPS_UNIX_ASSEMBLER and C.

This pragma indicates that the body of the specified subprogram will be supplied later, and conforms to DIPS standard calling conventions. The language name is ignored, except to check that the parameter formats correspond correctly.

pragma OPTIMIZE :

This pragma is effective only when the "OPTIMIZE" option has been given to the compiler. The argument is either TIME or SPACE. The pragma is always ignored for INLINE subprograms.

pragma PRIORITY :

The PRIORITY argument is an integer static expression value of predefined integer subtype PRIORITY. The pragma has no effect in a location other than a task (type) specification or outermost declaration part of a subprogram. If the pragma appears in the declaration part of a subprogram, it has no effect unless that subprogram is designated as the "main" subprogram at link time.

pragma SUPPRESS :

All CHECK names will have an effact, except DIVISION_CHECK and OVERFLOW_CHECK.

---

(2) Implementation-Dependent Attributes

The following are implementation defined attributes

attribute P'REFER :

For prefix P that denotes an object or label:

Yields an expression that can be used in a code statement as the address of the object or label. This allows objects and labels to be referenced symbolically by machine code statements. The value of this attribute is of the type symboladdress defined in package machinecode.

attribute P'REFERS :

For prefix P that denotes an object or label:

Yields an expression that can be used in a code statement as the address of the object or label. This allows objects and labels to be referenced symbolically by machine code statements. The value of this attribute is of the type symboladdrs defined in package machinecode.

The following notes augment the language-required definitions of the predefined attributes found in Appendix A of the Ada Reference Manual.

T'MACHINE_ROUNDS	is false.
T'MACHINE_RADIX	is 16.
T'MACHINE_MANTISSA	if the size of the base type of T is 32, MACHINE_MANTISSA is 24. Otherwise it is 56.
T'MACHINE_EMAX	is 63.
T'MACHINE_EMIN	is -64.
T'MACHINE_OVERFLOW	is true.

---

(3) Package SYSTEM

```
-- Ada Language System (ALS)
-- (COPYRIGHT (C) 1986, SOFTECH, INC. and Nippon Telegraph and Telephone)
-- The U.S. Government has unlimited rights to the ALS.

PACKAGE system IS

        TYPE address is NEW long_integer;

        TYPE name IS
            (DIPS, DIPS_UNIX, DIPS_EXEC);

        system_name: CONSTANT name := DIPS_UNIX;

        storage_unit: CONSTANT :=  8;
        memory_size:  CONSTANT :=  2**28 - 1;

        min_int:      CONSTANT := -(2**31);
        max_int:      CONSTANT :=  (2**31)-1;
        max_digits:   CONSTANT := 14;     -- Changed to correspond with standard.
        max_mantissa: CONSTANT := 31;
        fine_delta:   CONSTANT := 2.0**(-31);
```

```
        tick:          CONSTANT := 0.000_001;

        subtype priority is integer range 1..15;

        unresolved_reference: exception;
        system_error:         exception;

END system;
```

---

(4) Representation Clause Restrictions

Representation clauses specify how the types of the language are to be mapped onto the underlying machine. The following are restrictions on representation clauses.

Integer Types.

Integer types are specified with constraints of the form
range L..R

where
R ≤ SYSTEM.MAX_INT
L ≥ SYSTEM.MIN_INT

Length specifications of the form:

for T'SIZE use N;

may specify integer values N such that
N is in 2..32,

and such that
R ≤ 2**(N-1)-1
and L ≥ -2**(N-1);

or else such that
R ≤ (2**N)-1
L ≥ 0
and N is in 1..31.

For a stand-alone object of integer type, a SIZE of 8 is used when
R ≤ 127
and L≥ -128

A SIZE of 16 is used when

R ≤ 2**15-1
and L ≥ -2**15

Otherwise a SIZE of 32 is used.

For components of integer types within packed composite objects, the smaller of the default stand-alone SIZE or the SIZE from a length specification will be used.

Floating Types.

Floating types are specified with constraints of the form:

digits D

where
D is an integer value in 1..14.

Size specifications for floating points are not allowed.

Fixed Types.

Fixed types are specified with constraints of the form

delta D range L..R

where

$$\underset{\text{SMALL}}{\max}\ (\text{abs}(R), \text{abs}(L)) \leq 2^{**}31-1$$

SMALL defaults to the largest integral power of 2 less than or equal to the specified delta D. (This implies that fixed values are stored right-aligned.) For specifications of the form:

for T'SMALL use X;

X must be specified as an integral power of 2 such that
X ≤ D

Size specification for fixed types are not allowed.

Enumeration Types.

In the absence of a representation specification for an enumeration type T, the internal representation of T'FIRST = 0. The SIZE for a stand-alone object of enumeration type T will be the smallest of the values 8, 16, or 32, such that the internal representation of T'FIRST and T'LAST both fall within the range

```
 0..255          => 'SIZE = 8
-128..127        => 'SIZE = 8
-32768..32767    => 'SIZE = 16
```

```
            otherwise      => 'SIZE = 32
```

Length specifications of the form

```
    for T'SIZE use N;
```

and/or enumeration representations of the form

```
    for T use aggregate;
```

are permitted for N in 2 .. 32, provided the representation of T'FIRST $\geq$ -(2**(N-1)) and the representation of T'LAST $\leq$ 2**(N-1)-1, or else for N in 1 .. 31, provided that the internal representation of T'FIRST>=0 and the representation of T'LAST <= 2**(T'SIZE) - 1.

For components of enumeration types within packed composite objects, the smaller of the default stand-alone SIZE or the SIZE from a length specification will be used.

Enumeration representation on types derived from the predefined type BOOLEAN will not be accepted, but length specifications will be accepted. The SIZE for stand-alone objects of boolean types will be 32.

Access Types.

Representation specifications for STORAGE_SIZE for access types are not allowed. All access types have SIZE of 32. Size specifications for access types are not allowed.

Arrays and Records.

A length specification of the form

```
    for T'SIZE use N;
```

may cause arrays and records to be packed, if required, to accomodate the length specification. If the size specified is not large enough to contain any value of the type, a diagnostic message of severity ERROR is generated.

The PACK pragma may be used to minimize wasted space between components of arrays and records. The pragma causes the type representation to be chosen such that storage space requirements are minimized at the possible expense of data access time and code space.

A record type representation specification may be used to describe the allocation of components in a record. Bits are numbered 0..7 from the left. (Bit 8 starts at the left of the next higher-numbered byte.) Each location specification must allow at least X bits of range, where X is large enough to hold any value of the subtype of the component being allocated. Otherwise, a diagnostic message of severity ERROR is generated.

The alignment clause of the form

at mod N

may specify alignments of 1 (byte), 2 (halfword), 4 (word), 8 (double word), or 4096 (page).

If it is determinable at compilation time that the SIZE of a record or array type or subtype is outside the range of STANDARD.LONG_INTEGER, a diagnostic message of severity WARNING is generated. Declaration of an object of such a type or subtype would raise NUMERIC_ERROR when elaborated.

Other Length Specifications.

Length Specifications are described in Section 13.2 of the Ada Reference Manual.

T'STORAGE_SIZE for task type T - Specifies the number of bytes to be allocated for the runtime stack of each task object of type T.

Size specifications for task types are not allowed.

---

(5) Conventions

There are no conventions used for an implementation generated names denoting implementation-dependent components.

---

(6) Address Clauses

There are no conventions that define the interpretation of expressions that appear in address clauses, including those for interrupts.

---

(7) Unchecked Conversions

A program is erroneous if it performs UNCHECKEDCONVERSION when the source and target types have different sizes.

---

(8) Input-Output Packages

The following are implementation-dependent characteristics of the input-output packages.

SEQUENTIAL IO Package:

```
PACKAGE  sequential_io  IS

PRIVATE

    TYPE file_type IS RECORD
        real_file_type: io_support.file_type;
        buffer:         element_type;
    END RECORD;

END sequential_io;
```

DIRECT IO Package:

```
PACKAGE  direct_io  IS

    TYPE  count      IS  RANGE  0 .. io_support.count'last;

PRIVATE

   TYPE file_type IS RECORD
      real_file_type: io_support.file_type;
   END RECORD;

END DIRECT_IO;
```

TEXT IO Package:

```
********************************************************************************

      The Package TEXT_IO contains the following (implementation specific)
      definitions in addition to those specified in 14.3.10 of the LRM:

********************************************************************************


WITH ada_rsl, io_exceptions, als_kapse;

use als_kapse;

--* PURPOSE:
--%    This package provides input and output services for textual files
--%
--%    including creation,deletion,opening, and closing of said files.
--%    This package is as specified in the Ada Reference Manual (1983).
```

```
--%
--%    And here a word about primary and secondary routines.  A primary routine is
--%    always visible outside the package.  If it references a file, it will
--%    attempt to gain exclusive access to that file descriptor.  (The term
--%    "exclusive access" is used with regard to tasks.)  All modifications or
--%    tests on file descriptor FIELDs must be made only if the current task
--%    has exclusive access to that descriptor.  In every case where a primary
--%    routine gains exclusive access to a file descriptor, that routine must
--%    release the file descriptor beFORE exiting.  Primary routines may call
--%    primary or secondary routines.  Secondary routines are never visible
--%    outside the package.  If a secondary routine references a file descriptor,
--%    that routine assumes exclusive access for that descriptor.  Secondary
--%    routines may only call other secondary routines.  All calls to BASIC_IO
--%    for reading or writing are made by secondary routines.  All other
--%    BASIC_IO calls are made by primary routines.


   TYPE count IS RANGE 0 .. integer'LAST;

   SUBTYPE field IS integer RANGE 0 .. integer'LAST;

----------------------P R I V A T E  -----------------------------------

PRIVATE

   buffer_length : CONSTANT := 256;        -- restricted the line length to 255
                                           -- so each record(buffer) contains
                                           -- a line.

   TYPE char_buffer IS ARRAY ( io_defs.data_length_int
                               RANGE 1 .. buffer_length ) of character;

   TYPE file_rec IS -- common file state description; actual FILE_TYPE
      RECORD        -- declarations will be access types to this record.
         strm_ptr        : io_defs.stream_id_prv;
         external_name   : string_util.var_string_rec ;
         files_class     : io_defs.file_class_enu := io_defs.fc_text;
         f_mode          : file_mode;
         interactive     : boolean := false;
         end_info        : boolean := false;
         curr_col        : count := 1;
         curr_line       : count := 1;
         curr_page       : count := 1;
         eoln_found      : boolean := false;
         eop_found       : boolean := false;
         eof_found       : boolean := false;
         line_len        : count := unbounded;
         page_len        : count := unbounded;
         max_rec_length  : count := buffer_length;
         curr_rec_length : io_defs.data_length_int := 0;
         text_index      : io_defs.data_length_int := 0;
         text_buf        : char_buffer;
```

```
        next_rec_length : io_defs.data_length_int := 0;
        next_buf        : char_buffer;
      END RECORD;

max_line_length : CONSTANT := 255;
max_filename_length : CONSTANT := 20;

TYPE file_type IS ACCESS file_rec;

-----------------------------------------------------------------------

std_input   : file_type;         -- the standard and current file descriptors
std_output  : file_type;         -- should not be visible to the user except
curr_input  : file_type;         -- through the provided procedure (see above).
curr_output : file_type;

--   Define file marker values.
line_term   : CONSTANT character := ASCII.LF;
page_term   : CONSTANT character := ASCII.FF;    -- form feed     (ctrl-L) (16#0C#)
file_term   : CONSTANT character := ASCII.SUB;   --               (ctrl-Z) (16#1A#)
null_strm   : CONSTANT := 0;                     -- null stream pointer value

END text_io;
```

LOW LEVEL IO:

Low-level input-output is not provided.

---

(9) Package STANDARD

****

The Package STANDARD contains the following (implementation specific) definitions in addition to those specified in Annex C of the LRM:

****

```
-- Ada Language System (ALS)
-- (COPYRIGHT (C) 1986, SOFTECH, INC. and Nippon Telegraph and Telephone)
-- The U.S. Government has unlimited rights to the ALS.

PACKAGE standard IS

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

TYPE short_integer IS RANGE -128 .. 127;
```

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.

APPENDIX C

NTT's AdaDIPS Pre-Validation Material

List of values for the ".TST" tests.

```
-- MACRO.DFS                    -- ACVC_VERSION_1.10

-- THIS FILE CONTAINS THE MACRO DEFINITIONS USED IN THE ACVC TESTS.
-- THESE DEFINITIONS ARE USED BY THE ACVC TEST PRE-PROCESSOR,
-- MACROSUB. MACROSUB WILL CALCULATE VALUES FOR THOSE MACRO SYMBOLS
-- WHOSE DEFINITIONS DEPEND ON THE VALUE OF MAX_IN_LEN (NAMELY, THE
-- VALUES OF THE MACRO SYMBOLS BIG_ID1, BIG_ID2, BIG_ID3, BIG_ID4,
-- BIG_STRING1, BIG_STRING2, MAX_STRING_LITERAL, BIG_INT_LIT,
-- BIG_REAL_LIT, MAX_LEN_INT_BASED_LITERAL, MAX_LEN_REAL_BASED_LITERAL,
-- AND BLANKS).  THEREFORE, ANY VALUES GIVEN IN THIS FILE FOR THOSE
-- MACRO SYMBOLS WILL BE IGNORED BY MACROSUB.

-- NOTE: THE MACROSUB PROGRAM EXPECTS THE FIRST MACRO IN THIS FILE TO
--       BE MAX_IN_LEN.

-- EACH DEFINITION IS ACCORDING TO THE FOLLOWING FORMAT:

-- A. A NUMBER OF LINES PRECEDED BY THE ADA COMMENT DELIMITER, --.
--    THE FIRST OF THESE LINES CONTAINS THE MACRO SYMBOL AS IT APPEARS
--    IN THE TEST FILES (WITH THE DOLLAR SIGN).  THE NEXT FEW "COMMENT"
--    LINES CONTAIN A DESCRIPTION OF THE VALUE TO BE SUBSTITUTED.
--    THE REMAINING "COMMENT" LINES, THE FIRST OF WHICH BEGINS WITH THE
--    WORDS "USED IN: " (NO QUOTES), CONTAIN A LIST OF THE TEST FILES
--    (WITHOUT THE .TST EXTENSION) IN WHICH THE MACRO SYMBOL APPEARS.
--    EACH TEST FILE NAME IS PRECEDED BY ONE OR MORE BLANKS.
-- B. THE IDENTIFIER (WITHOUT THE DOLLAR SIGN) OF THE MACRO SYMBOL,
--    FOLLOWED BY A SPACE OR TAB, FOLLOWED BY THE VALUE TO BE
--    SUBSTITUTED.  IN THE DISTRIBUTION FILE, A SAMPLE VALUE IS
--    PROVIDED; THIS VALUE MUST BE REPLACED BY A VALUE APPROPRIATE TO
--    THE IMPLEMENTATION.

-- DEFINITIONS ARE SEPARATED BY ONE OR MORE EMPTY LINES.
-- THE LIST OF DEFINITIONS BEGINS AFTER THE FOLLOWING EMPTY LINE.

-- $MAX_IN_LEN
-- AN INTEGER LITERAL GIVING THE MAXIMUM LENGTH PERMITTED BY THE
-- COMPILER FOR A LINE OF ADA SOURCE CODE (NOT INCLUDING AN END-OF-LINE
-- CHARACTER).
-- USED IN:  A26007A
MAX_IN_LEN      120

-- $BIG_ID1
-- AN IDENTIFIER IN WHICH THE NUMBER OF CHARACTERS IS $MAX_IN_LEN.
-- USED IN:  C23003A  C23003B  C23003C  B23003D  B23003E  C23003G
--           C23003H  C23003I  C23003J  C35502D  C35502F
BIG_ID1 AAAAAAAAA_AAAAAAAAA_AAAAAAAAA_AAAAAAAAA_AAAAAAAAA_AAAAAAAAA_
AAAAAAAAA_AAAAAAAAA_AAAAAAAAA_AAAAAAAAA_AAAAAAAAA_AAAAAAAAA1
```

```
-- $BIG_ID2
-- AN IDENTIFIER IN WHICH THE NUMBER OF CHARACTERS IS $MAX_IN_LEN,
-- DIFFERING FROM $BIG_ID1 ONLY IN THE LAST CHARACTER.
-- USED IN:  C23003A  C23003B  C23003C  B23003F  C23003G  C23003H
--           C23003I  C23003J
BIG_ID2 AAAAAAAAA_AAAAAAAAA_AAAAAAAAA_AAAAAAAAA_AAAAAAAAA_AAAAAAAAA_
AAAAAAAAA_AAAAAAAAA_AAAAAAAAA_AAAAAAAAA_AAAAAAAAA_AAAAAAAAA2

-- $BIG_ID3
-- AN IDENTIFIER IN WHICH THE NUMBER OF CHARACTERS IS $MAX_IN_LEN.
-- USED IN:  C23003A  C23003B  C23003C  C23003G  C23003H  C23003I
--           C23003J
BIG_ID3 AAAAAAAAA_AAAAAAAAA_AAAAAAAAA_AAAAAAAAA_AAAAAAAAA_AAAAAAAAA3
AAAAAAAAA_AAAAAAAAA_AAAAAAAAA_AAAAAAAAA_AAAAAAAAA_AAAAAAAAAA

-- $BIG_ID4
-- AN IDENTIFIER IN WHICH THE NUMBER OF CHARACTERS IS $MAX_IN_LEN,
-- DIFFERING FROM $BIG_ID3 ONLY IN THE MIDDLE CHARACTER.
-- USED IN:  C23003A  C23003B  C23003C  C23003G  C23003H  C23003I
--           C23003J
BIG_ID4 AAAAAAAAA_AAAAAAAAA_AAAAAAAAA_AAAAAAAAA_AAAAAAAAA_AAAAAAAAA4
AAAAAAAAA_AAAAAAAAA_AAAAAAAAA_AAAAAAAAA_AAAAAAAAA_AAAAAAAAAA

-- $BIG_STRING1
-- A STRING LITERAL (WITH QUOTES) WHOSE CATENATION WITH $BIG_STRING2
-- ($BIG_STRING1 & $BIG_STRING2) PRODUCES THE IMAGE OF $BIG_ID1.
-- USED IN:  C35502D  C35502F
BIG_STRING1 "AAAAAAAAA_AAAAAAAAA_AAAAAAAAA_AAAAAAAAA_AAAAAAAAA_AAAAAAAAA_"

-- $BIG_STRING2
-- A STRING LITERAL (WITH QUOTES) WHOSE CATENATION WITH $BIG_STRING1
-- ($BIG_STRING1 & $BIG_STRING2) PRODUCES THE IMAGE OF $BIG_ID1.
-- USED IN:  C35502D  C35502F
BIG_STRING2 "AAAAAAAAA_AAAAAAAAA_AAAAAAAAA_AAAAAAAAA_AAAAAAAAA_AAAAAAAAA1"

-- $MAX_STRING_LITERAL
-- A STRING LITERAL CONSISTING OF $MAX_IN_LEN CHARACTERS (INCLUDING THE
-- QUOTE CHARACTERS).
-- USED IN:  A26007A
MAX_STRING_LITERAL "CCCCCCC10CCCCCCCC20CCCCCCCC30CCCCCCCC40CCCCCCCC50
CCCCCCCC60CCCCCCCC70CCCCCCCC80CCCCCCCC90CCCCCCCC00CCCCCCCC11CCCCCCC1"
```

```
-- $NEG_BASED_INT
-- A BASED_INTEGER LITERAL (PREFERABLY BASE 8 OR 16) WHOSE HIGHEST ORDER
-- NON-ZERO BIT WOULD FALL IN THE SIGN BIT POSITION OF THE
-- REPRESENTATION FOR SYSTEM.MAX_INT, I.E., AN ATTEMPT TO WRITE A
-- NEGATIVE VALUED LITERAL SUCH AS -2 BY TAKING ADVANTAGE OF THE
-- BIT REPRESENTATION.
-- USED IN:  E24201A
NEG_BASED_INT   8#37777777772#

$BIG_INT_LIT
-- AN INTEGER LITERAL WHOSE VALUE IS 298, BUT WHICH HAS
-- ($MAX_IN_LEN - 3) LEADING ZEROES.
-- USED IN:  C24003A
BIG_INT_LIT 000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000298

$BIG_REAL_LIT
-- A UNIVERSAL_REAL LITERAL WHOSE VALUE IS 690.0, BUT WHICH HAS
-- ($MAX_IN_LEN - 5) LEADING ZEROES.
-- USED IN:  C24003B  C24003C
BIG_REAL_LIT 000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000690.0

-- $MAX_LEN_INT_BASED_LITERAL
-- A BASED INTEGER LITERAL (USING COLONS) WHOSE VALUE IS 2:11:, HAVING
-- ($MAX_IN_LEN - 5) ZEROES BETWEEN THE FIRST COLON AND THE FIRST 1.
-- USED IN:  C2A009A
MAX_LEN_INT_BASED_LITERAL 2:000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000
00000000011:

-- $MAX_LEN_REAL_BASED_LITERAL
-- A BASED REAL LITERAL (USING COLONS) WHOSE VALUE IS 16:F.E:, HAVING
-- ($MAX_IN_LEN - 7) ZEROES BETWEEN THE FIRST COLON AND THE F.
-- USED IN:  C2A009A
MAX_LEN_REAL_BASED_LITERAL 16:0000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000
000000000F.E:

-- $BLANKS
-- A SEQUENCE OF ($MAX_IN_LEN - 20) BLANKS.
-- USED IN:  B22001A  B22001B  B22001C  B22001D  B22001E  B22001F
--           B22001G  B22001I  B22001J  B22001K  B22001L  B22001M
--           B22001N
--      <       LIMITS OF SAMPLE SHOWN BY ANGLE BRACKETS
BLANKS
```

```
-- $MAX_DIGITS
-- AN INTEGER LITERAL WHOSE VALUE IS SYSTEM.MAX_DIGITS.
-- USED IN:  B35701A  CD7102B
MAX_DIGITS      14

-- $NAME
-- THE NAME OF A PREDEFINED INTEGER TYPE OTHER THAN INTEGER,
-- SHORT_INTEGER, OR LONG_INTEGER.
-- (IMPLEMENTATIONS WHICH HAVE NO SUCH TYPES SHOULD USE AN UNDEFINED
-- IDENTIFIER SUCH AS   NO_SUCH_TYPE_AVAILABLE.)
-- USED IN:  AVAT007  C45231D  B86001X CD7101G
NAME    SHORT_SHORT_INTEGER

-- $FLOAT_NAME
-- THE NAME OF A PREDEFINED FLOATING POINT TYPE OTHER THAN FLOAT,
-- SHORT_FLOAT, OR LONG_FLOAT.  (IMPLEMENTATIONS WHICH HAVE NO SUCH
-- TYPES SHOULD USE AN UNDEFINED IDENTIFIER SUCH AS NO_SUCH_TYPE.)
-- USED IN:  AVAT013  B86001Z
FLOAT_NAME      SHORT_SHORT_FLOAT

-- $FIXED_NAME
-- THE NAME OF A PREDEFINED FIXED POINT TYPE OTHER THAN DURATION.
-- (IMPLEMENTATIONS WHICH HAVE NO SUCH TYPES SHOULD USE AN UNDEFINED
-- IDENTIFIER SUCH AS NO_SUCH_TYPE.)
-- USED IN:  AVAT030  AVAT015  B86001Y
FIXED_NAME      NO_SUCH_FIXED_TYPE

-- $INTEGER_FIRST
-- AN INTEGER LITERAL, WITH SIGN, WHOSE VALUE IS INTEGER'FIRST.
-- USED IN:  C35503F  B54B01B
INTEGER_FIRST   -32768

-- $INTEGER_LAST
-- AN INTEGER LITERAL WHOSE VALUE IS INTEGER'LAST.
-- USED IN:  C35503F  B45232A  B45B01B
INTEGER_LAST    32767

-- $INTEGER_LAST_PLUS_1
-- AN INTEGER LITERAL WHOSE VALUE IS INTEGER'LAST + 1.
-- USED IN:  C45232A
INTEGER_LAST_PLUS_1     32_768

-- $MIN_INT
-- AN INTEGER LITERAL, WITH SIGN, WHOSE VALUE IS SYSTEM.MIN_INT.
-- THE LITERAL MUST NOT CONTAIN UNDERSCORES OR LEADING OR TRAILING
-- BLANKS.
-- USED IN:  C35503D  C35503F  CD7101B
MIN_INT -2147483648
```

```
-- $MAX_INT
-- AN INTEGER LITERAL WHOSE VALUE IS SYSTEM.MAX_INT.
-- THE LITERAL MUST NOT INCLUDE UNDERSCORES OR LEADING OR TRAILING
-- BLANKS.
-- USED IN:  C35503D  C35503F  C4A007A  CD7101B
MAX_INT 2147483647

-- $MAX_INT_PLUS_1
-- AN INTEGER LITERAL WHOSE VALUE IS SYSTEM.MAX_INT + 1.
-- USED IN:  C45232A
MAX_INT_PLUS_1  2_147_483_648

-- $LESS_THAN_DURATION
-- A REAL LITERAL (WITH SIGN) WHOSE VALUE (NOT SUBJECT TO
-- ROUND-OFF ERROR IF POSSIBLE) LIES BETWEEN DURATION'BASE'FIRST AND
-- DURATION'FIRST.  IF NO SUCH VALUES EXIST, USE A VALUE IN
-- DURATION'RANGE.
-- USED IN:  C96005B
LESS_THAN_DURATION      -100_000.0

-- $GREATER_THAN_DURATION
-- A REAL LITERAL WHOSE VALUE (NOT SUBJECT TO ROUND-OFF ERROR
-- IF POSSIBLE) LIES BETWEEN DURATION'BASE'LAST AND DURATION'LAST.  IF
-- NO SUCH VALUES EXIST, USE A VALUE IN DURATION'RANGE.
-- USED IN:  C96005B
GREATER_THAN_DURATION   100_000.0

-- $LESS_THAN_DURATION_BASE_FIRST
-- A REAL LITERAL (WITH SIGN) WHOSE VALUE IS LESS THAN
-- DURATION'BASE'FIRST.
-- USED IN:  C96005C
LESS_THAN_DURATION_BASE_FIRST   -10_000_000.0

-- $GREATER_THAN_DURATION_BASE_LAST
-- A REAL LITERAL WHOSE VALUE IS GREATER THAN DURATION'BASE'LAST.
-- USED IN: C96005C
GREATER_THAN_DURATION_BASE_LAST 10_000_000.0

-- $COUNT_LAST
-- AN INTEGER LITERAL WHOSE VALUE IS TEXT_IO.COUNT'LAST.
-- USED IN:  CE3002B
COUNT_LAST      32_767

-- $FIELD_LAST
-- AN INTEGER LITERAL WHOSE VALUE IS TEXT_IO.FIELD'LAST.
-- USED IN:  CE3002C
FIELD_LAST      32_767
```

```
-- $ILLEGAL_EXTERNAL_FILE_NAME1
-- AN ILLEGAL EXTERNAL FILE NAME (E.G., TOO LONG, CONTAINING INVALID
-- CHARACTERS, CONTAINING WILD-CARD CHARACTERS, OR SPECIFYING A
-- NONEXISTENT DIRECTORY).
-- USED IN:  CE2102C  CE2102H  CE2103A  CE2103B  CE3102B  CE3107A
ILLEGAL_EXTERNAL_FILE_NAME1     /nonexist/file

-- $ILLEGAL_EXTERNAL_FILE_NAME2
-- AN ILLEGAL EXTERNAL FILE NAME, DIFFERENT FROM $EXTERNAL_FILE_NAME1.
-- USED IN:  CE2102C  CE2102H  CE2103A  CE2103B  CE3102B
ILLEGAL_EXTERNAL_FILE_NAME2     /nonexist/directory

-- $ACC_SIZE
-- AN INTEGER LITERAL WHOSE VALUE IS THE MINIMUM NUMBER OF BITS
-- SUFFICIENT TO HOLD ANY VALUE OF AN ACCESS TYPE.
-- USED IN:  CD1C03C  CD2A81A  CD2A81B  CD2A81C  CD2A81D  CD2A81E
--           CD2A81F  CD2A81G  CD2A83A  CD2A83B  CD2A83C  CD2A83E
--           CD2A83F  CD2A83G  ED2A86A  CD2A87A  AVAT135
ACC_SIZE        32

-- $TASK_SIZE
-- AN INTEGER LITERAL WHOSE VALUE IS THE NUMBER OF BITS REQUIRED TO
-- HOLD A TASK OBJECT WHICH HAS A SINGLE ENTRY WITH ONE INOUT PARAMETER.
-- USED IN:  CD2A91A  CD2A91B  CD2A91C  CD2A91D  CD2A91E  AVAT137
TASK_SIZE       128

-- $MIN_TASK_SIZE
-- AN INTEGER LITERAL WHOSE VALUE IS THE NUMBER OF BITS REQUIRED TO
-- HOLD A TASK OBJECT WHICH HAS NO ENTRIES, NO DECLARATIONS, AND "NULL;"
-- AS THE ONLY STATEMENT IN ITS BODY.
-- USED IN:  CD2A95A
MIN_TASK_SIZE   1504

-- $NAME_LIST
-- A LIST OF THE ENUMERATION LITERALS IN THE TYPE SYSTEM.NAME, SEPARATED
-- BY COMMAS.
-- USED IN:  CD7003A
NAME_LIST       DIPS,DIPS_UNIX,DIPS_EXEC

-- $DEFAULT_SYS_NAME
-- THE VALUE OF THE CONSTANT SYSTEM.SYSTEM_NAME.
-- USED IN:  CD7004A  CD7004C  CD7004D
DEFAULT_SYS_NAME        DIPS_UNIX
```

```
-- $NEW_SYS_NAME
-- A VALUE OF THE TYPE SYSTEM.NAME, OTHER THAN $DEFAULT_SYS_NAME.  IF
-- THERE IS ONLY ONE VALUE OF THE TYPE, THEN USE THAT VALUE.
-- NOTE: IF THERE ARE MORE THAN TWO VALUES OF THE TYPE, THEN THE
--       PERTINENT TESTS ARE TO BE RUN ONCE FOR EACH ALTERNATIVE.
-- USED IN:  ED7004B1
NEW_SYS_NAME    DIPS_UNIX

-- $DEFAULT_STOR_UNIT
-- AN INTEGER LITERAL WHOSE VALUE IS SYSTEM.STORAGE_UNIT.
-- USED IN:  CD7005B  ED7005D3M  CD7005E
DEFAULT_STOR_UNIT       8

-- $NEW_STOR_UNIT
-- AN INTEGER LITERAL WHOSE VALUE IS A PERMITTED ARGUMENT FOR
-- PRAGMA STORAGE_UNIT, OTHER THAN $DEFAULT_STOR_UNIT.  IF THERE
-- IS NO OTHER PERMITTED VALUE, THEN USE THE VALUE OF
-- $SYSTEM.STORAGE_UNIT.  IF THERE IS MORE THAN ONE ALTERNATIVE,
-- THEN THE PERTINENT TESTS SHOULD BE RUN ONCE FOR EACH ALTERNATIVE.
-- USED IN:  ED7005C1  ED7005D1  CD7005E
NEW_STOR_UNIT   8

-- $DEFAULT_MEM_SIZE
-- AN INTEGER LITERAL WHOSE VALUE IS SYSTEM.MEMORY_SIZE.
-- USED IN:  CD7006B  ED7006D3M  CD7006E
DEFAULT_MEM_SIZE        268_435_455

-- $NEW_MEM_SIZE
-- AN INTEGER LITERAL WHOSE VALUE IS A PERMITTED ARGUMENT FOR
-- PRAGMA MEMORY_SIZE, OTHER THAN $DEFAULT_MEM_SIZE.  IF THERE IS NO
-- OTHER VALUE, THEN USE $DEFAULT_MEM_SIZE.  IF THERE IS MORE THAN
-- ONE ALTERNATIVE, THEN THE PERTINENT TESTS SHOULD BE RUN ONCE FOR
-- EACH ALTERNATIVE.  IF THE NUMBER OF PERMITTED VALUES IS LARGE, THEN
-- SEVERAL VALUES SHOULD BE USED, COVERING A WIDE RANGE OF
-- POSSIBILITIES.
-- USED IN:  ED7006C1  ED7006D1  CD7006E
NEW_MEM_SIZE    248_435_455

-- $LOW_PRIORITY
-- AN INTEGER LITERAL WHOSE VALUE IS THE LOWER BOUND OF THE RANGE
-- FOR THE SUBTYPE SYSTEM.PRIORITY.
-- USED IN:  CD7007C
LOW_PRIORITY    1

-- $HIGH_PRIORITY
-- AN INTEGER LITERAL WHOSE VALUE IS THE UPPER BOUND OF THE RANGE
-- FOR THE SUBTYPE SYSTEM.PRIORITY.
-- USED IN:  CD7007C
HIGH_PRIORITY   15
```

```
-- $MANTISSA_DOC
-- AN INTEGER LITERAL WHOSE VALUE IS SYSTEM.MAX_MANTISSA AS SPECIFIED
-- IN THE IMPLEMENTOR'S DOCUMENTATION.
-- USED IN:  CD7013B
MANTISSA_DOC    31

-- $DELTA_DOC
-- A REAL LITERAL WHOSE VALUE IS SYSTEM.FINE_DELTA AS SPECIFIED IN THE
-- IMPLEMENTOR'S DOCUMENTATION.
-- USED IN:  CD7013D
DELTA_DOC       0.000_000_000_465_661_287_307_739_257_812_5

-- $TICK
-- A REAL LITERAL WHOSE VALUE IS SYSTEM.TICK AS SPECIFIED IN THE
-- IMPLEMENTOR'S DOCUMENTATION.
-- USED IN:  CD7104B
TICK    0.000001
```

APPENDIX D

WITHDRAWN TESTS

Some tests are withdrawn from the ACVC because they do not conform to the Ada Standard. The following 44 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.

A39005G
This test unreasonably expects a component clause to pack an array component into a minimum size (line 30).

B97102E
This test contains an unintended illegality: a select statement contains a null statement at the place of a selective wait alternative (line 31).

C97116A
This test contains race conditions, and it assumes that guards are evaluated indivisibly. A conforming implementation may use interleaved execution in such a way that the evaluation of the guards at lines 50 & 54 and the execution of task CHANGING_OF_THE_GUARD results in a call to REPORT.FAILED at one of lines 52 or 56.

BC3009B
This test wrongly expects that circular instantiations will be detected in several compilation units even though none of the units is illegal with respect to the units it depends on; by AI-00256, the illegality need not be detected until execution is attempted (line 95).

CD2A62D
This test wrongly requires that an array object's size be no greater than 10 although its subtype's size was specified to be 40 (line 137).

CD2A63A..D, CD2A66A..D, CD2A73A..D, CD2A76A..D [16 tests]
These tests wrongly attempt to check the size of objects of a derived type (for which a 'SIZE length clause is given) by passing them to a derived subprogram (which implicitly converts them to the parent type (Ada standard 3.4:14)). Additionally, they use the 'SIZE length clause and attribute, whose interpretation is considered problematic by the WG9 ARG.

CD2A81G, CD2A83G, CD2A84M & N, & CD50110
These tests assume that dependent tasks will terminate while the main program executes a loop that simply tests for task termination; this is not the case, and the main program may loop indefinitely (lines 74, 85, 86 & 96, 86 & 96, and 58, resp.).

CD2B15C & CD7205C
These tests expect that a 'STORAGE_SIZE length clause provides precise control over the number of designated objects in a collection; the Ada standard 13.2:15 allows that such control must not be expected.

CD2D11B
This test gives a SMALL representation clause for a derived fixed-point type (at line 30) that defines a set of model numbers that are not necessarily represented in the parent type; by Commentary AI-00099, all model numbers of a derived fixed-point type must be representable values of the parent type.

CD5007B
This test wrongly expects an implicitly declared subprogram to be at the address that is specified for an unrelated subprogram (line 303).

ED7004B, ED7005C & D, ED7006C & D [5 tests]
These tests check various aspects of the use of the three SYSTEM pragmas; the AVO withdraws these tests as being inappropriate for validation.

CD7105A
This test requires that successive calls to CALENDAR.CLOCK change by at least SYSTEM.TICK; however, by Commentary AI-00201, it is only the expected frequency of change that must be at least SYSTEM.TICK -- particular instances of change may be less (line 29).

CD7203B, & CD7204B
These tests use the 'SIZE length clause and attribute, whose interpretation is considered problematic by the WG9 ARG.

CD7205D
This test checks an invalid test objective: it treats the specification of storage to be reserved for a task's activation as though it were like the specification of storage for a collection.

CE2107I
This test requires that objects of two similar scalar types be distinguished when read from a file--DATA_ERROR is expected to be raised by an attempt to read one object as of the other type. However, it is not clear exactly how the Ada standard 14.2.4:4 is to be interpreted; thus, this test objective is not considered valid. (line 90)

CE3111C
This test requires certain behavior, when two files are associated with the same external file, that is not required by the Ada standard.

CE3301A
This test contains several calls to END_OF_LINE & END_OF_PAGE that have no parameter: these calls were intended to specify a file, not to refer to STANDARD_INPUT (lines 103, 107, 118, 132, & 136).

CE3411B
This test requires that a text file's column number be set to COUNT'LAST in order to check that LAYOUT_ERROR is raised by a subsequent PUT operation. But the former operation will generally raise an exception due to a lack of available disk space, and the test would thus encumber validation testing.

E28005C
This test expects that the string "-- TOP OF PAGE. --63" of line 204 will appear at the top of the listing page due to a pragma PAGE in line 203; but line 203 contains text that follows the pragma, and it is this that must appear at the top of the page.

APPENDIX E

COMPILER OPTIONS AS SUPPLIED BY

SofTech, Inc.

Compiler: AdaDIPS, Version 1.0

ACVC Version: 1.10

APPENDIX E

NTT's AdaDIPS Pre-Validation Material

AdaDIPS compiler options list
(with the options used highlighted).

The following options may be specified to the compiler. Default options have a (D) after the option name. The options in capitol letters are the options used at pre-validation.

Listing Control Options:

PRIVATE (D)	no_private
notes	NO_NOTES (D)
attribute	NO_ATTRIBUTE (D)
xref	NO_XREF (D)
machine	NO_MACHINE (D)
read_only_data	NO_READ_ONLY_DATA (D)
read_write_data	NO_READ_WRITE_DATA (D)
stack_data	NO_STACK_DATA (D)
diagnostics	NO_DIAGNOSTICS (D)
summary	NO_SUMMARY (D)
SOURCE	NO_SOURCE (D)

(NOTE: SOURCE is used for E tests, B tests, and not-applicable tests. All others use NO_SOURCE.)

Maintenance Aid Options:

compiler_maint	NO_COMPILER_MAINT (D)
stop_container_nn	
unlimited_errors	
flags_36_stp	

Special Compilation Options:

standard_compile	NO_STANDARD_COMPILE (D)
system_compile	NO_SYSTEM_COMPILE (D)
rsl_compile	NO_RSL_COMPILE (D)
bis_compile	NO_BIS_COMPILE (D)

Other Options:

CHECKS (D)	no_checks
CODE_ON_WARNING (D)	no_code_on_warning
CONTAINER_GENERATION (D)	no_container_generation
frequency	NO_FREQUENCY (D)
kana	NO_KANA (D)
object_only	NO_OBJECT_ONLY (D)
optimize (D)	NO_OPTIMIZE
trace_back (D)	NO_TRACE_BACK
sfd_opt	NO_SFD_OPT (D)

Listing Control Options:

PRIVATE	List text in the private part of a package specifier (if there is a source listing) subject to requirements of LIST pragmas.
NOTES	Include diagnostics of severity NOTE in the source listings and in the diagnostic summary listing.
ATTRIBUTE	Produce a symbol attribute listing.
XREF	Produce a cross-reference listing.
MACHINE	Produce a machine code listing if code is generated.
READ_ONLY_DATA	Produce a data map listing of read only data generated.
READ_WRITE_DATA	Produce a data map listing of read/write data if code is generated.
STACK_DATA	Produce a data map listing of stack data if code is generated.
DIAGNOSTICS	Produce a diagnostic summary listing.
SUMMARY	Produce a display CPU time used, total diagnostics, and options used for compilation. Regardless of the option setting, the compiler will always provide this information if errors or recompilation advisories were detected.
SOURCE	Produce a listing of the source text.

Maintainance Aid Options:

COMPILER_MAINT	Permit the other maintenance aid options to have an effect.
STOP_CONTAINER_nn	Save the state of the Container into the current program library and halt the compilation process after the "nn" portion.
UNLIMITED_ERRORS	Allow the compiler to continue the compilation process after the total errors generated reaches the predefined limit of 200 errors.
FLAGS_36_STP	Control function. S: show resources used between phases T: Display diagnostic information at terminal P: Turn on PLM maintenance flags

Special Compilation Options:

STANDARD_COMPILE	Compile a new version of the predefined package STANDARD.
SYSTEM_COMPILE	Compile a new version of the specification for the predefined library pakage SYSTEM.
RSL_COMPILE	Compile a new version of the specification for the package ADA_RSL.
BIS_COMPILE	Compile specific generic subprogram specifications or specific packages containing only generic subprogram specifications, for which the compiler will later recognize instantiations as implementation-defined built-in subprograms.

Other Options:

CHECKS	Provide run-time error checking.
CODE_ON_WARNING	Generated code (and, if requested, a machine code listing) when there are diagnostics of severity level WARNING, provided there are no FATAL, SYSTEM, or ERROR diagnostics.
CONTAINER_GENERATION	Produce a Container if diagnostic severity permits.
FREQUENCY	Permit generation of code to monitor execution frequency at the basic block level.
KANA	Allow the extended character set to be used in comments and string literals without generating errors.
OBJECT_ONLY	Include in compiler output only that information needed to link, export, and execute the current compilation.
OPTIMIZE	Permit optimization in accordance with the OPTIMIZE pragmas that appear in the text.
TRACE_BACK	Provide the user with calling sequence traceback information when the program aborts because of an unhandled exception.
SFD_OPT	Allows the user to not maintain the Static Frame Descriptor for subprograms in the compilation unit.