AD-A242 277

NTATION PAGE

Form Approved
OPM No. 0704-0188

hour per response, including the time for reviewing instructions, searching existing data sources gathering and maintaining the data is burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Information and Regulatory Affairs, Office of

1. AGENCY USE ONLY (Leave Blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
		Final: 03 Apr 1991 to 01 Jun 1993

4. TITLE AND SUBTITLE
Texas Instruments, TI Ada, Version 1.0, MicroVAX 3400 (Host)to TI DP32 R3000 Processor (Target), 910403W1.11135

5. FUNDING NUMBERS

6. AUTHOR(S)
Wright-Patterson AFB, Dayton, OH
USA

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Ada Validation Facility, Language Control Facility ASD/SCEL
Bldg. 676, Rm 135
Wright-Patterson AFB, Dayton, OH 45433

8. PERFORMING ORGANIZATION REPORT NUMBER
AVF-VSR-459.0891

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Ada Joint Program Office
United States Department of Defense
Pentagon, Rm 3E114
Washington, D.C. 20301-3081

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES
No software available for distribution per Michele Kee, AJPO 11/4/91 telecon Mgm

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*
Texas Instruments, TI Ada, Version 1.0, Wright-Patterson AFB, OH, MicroVAX 3400 (Host)to TI DP32 R3000 Processor (Target), AVCV 1.11.

91-15052

14. SUBJECT TERMS
Ada programming language, Ada Compiler Val. Summary Report, Ada Compiler Val. Capability, Val. Testing, Ada Val. Office, Ada Val. Facility, ANSI/MIL-STD-1815A, AJPO.

15. NUMBER OF PAGES

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFED	UNCLASSIFIED	

NSN 7540-01-280-550

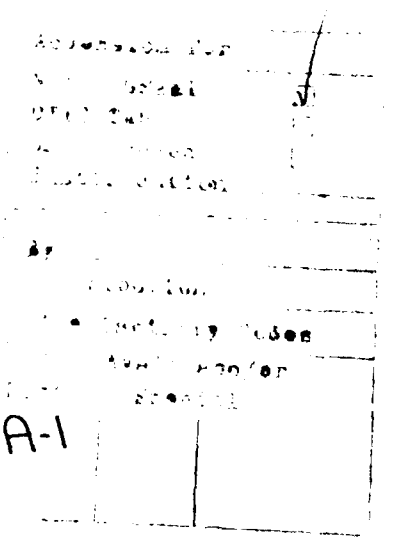
Standard Form 298, (Rev. 2-89)
Prescribed by ANSI Std. 239-128

91 1104 123

AVF Control Number: AVF-VSR-459.0891
23 August 1991
91-01-10-TEX

Ada COMPILER
VALIDATION SUMMARY REPORT:
Certificate Number: 910403W1.11135
Texas Instruments
TI Ada, Version 1.0
MicroVAX 3400 => TI DP32 R3000 Processor

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



Certificate Information

The following Ada implementation was tested and determined to pass ACVC 1.11. Testing was completed on 03 April 1991.

Compiler Name and Version: TI Ada, Version 1.0

Host Computer System: MicroVAX 3400, running VMS 5.3-1

Target Computer System: TI DP32 R3000 Processor, running TI Executive and Runtime Services (EARS) Version 1.0

Customer Agreement Number: 91-01-10-TEX

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

As a result of this validation effort, Validation Certificate 910403W1.11135 is awarded to Texas Instruments. This certificate expires on 1 June 1993.

This report has been reviewed and is approved.

Ada Validation Facility
Steven P. Wilson
Technical Director
ASD/SCEL
Wright-Patterson AFB OH 45433-6503

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

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

DECLARATION OF CONFORMANCE

Customer: Texas Instruments

Ada Validation Facility: ASD/SCEL, Wright-Patterson AFB OH 45433-6503

ACVC Version: 1.11

Ada Implementation:

Compiler Name and Version: TI Ada , Version 1.0
Host Computer System: microVAX 3400 running VMS 5.3-1
Target Computer System: TI DP32 R3000 Processor, running TI Executive and Runtime Services (EARS) Version 1.0

Customer's Declaration

I, the undersigned, representing Texas Instruments, declare that Texas Instruments has no knowledge of deliberate deviations from the Ada Language Standard ANSI/MIL-STD-1815A in the implementation listed in this declaration.

Stewart L French

Date: 4/15/1991

Texas Instruments
Stewart L. French
Member, Group Technical Staff

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

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

1.1 USE OF THIS VALIDATION SUMMARY REPORT

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

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

National Technical Information Service
5285 Port Royal Road
Springfield VA 22161

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

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

1.2 REFERENCES

[Ada83] Reference Manual for the Ada Programming Language, ANSI/MIL-STD-1815A, February 1983 and ISO 8652-1987.

[Pro90] Ada Compiler Validation Procedures, Version 2.1, Ada Joint Program Office, August 1990.

[UG89] Ada Compiler Validation Capability User's Guide, 21 June 1989.

1.3 ACVC TEST CLASSES

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

The executable tests are written in a self-checking manner and produce a PASSED, FAILED, or NOT APPLICABLE message indicating the result when they are executed. Three Ada library units, the packages REPORT and SPPRT13, and the procedure CHECK_FILE are used for this purpose. The package REPORT also provides a set of identity functions used to defeat some compiler optimizations allowed by the Ada Standard that would circumvent a test objective. The package SPPRT13 is used by many tests for Chapter 13 of the Ada Standard. The procedure CHECK_FILE is used to check the contents of text files written by some of the Class C tests for Chapter 14 of the Ada Standard. The operation of REPORT and CHECK_FILE is checked by a set of executable tests. If these units are not operating correctly, validation testing is discontinued.

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

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

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

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

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

1.4 DEFINITION OF TERMS

Ada Compiler	The software and any needed hardware that have to be added to a given host and target computer system to allow transformation of Ada programs into executable form and execution thereof.
Ada Compiler Validation Capability (ACVC)	The means for testing compliance of Ada implementations, consisting of the test suite, the support programs, the ACVC user's guide and the template for the validation summary report.
Ada Implementation	An Ada compiler with its host computer system and its target computer system.
Ada Joint Program Office (AJPO)	The part of the certification body which provides policy and guidance for the Ada certification system.
Ada Validation Facility (AVF)	The part of the certification body which carries out the procedures required to establish the compliance of an Ada implementation.
Ada Validation Organization (AVO)	The part of the certification body that provides technical guidance for operations of the Ada certification system.
Compliance of an Ada Implementation	The ability of the implementation to pass an ACVC version.
Computer System	A functional unit, consisting of one or more computers and associated software, that uses common storage for all or part of a program and also for all or part of the data necessary for the execution of the program; executes user-written or user-designated programs; performs user-designated data manipulation, including arithmetic operations and logic operations; and that can execute programs that modify themselves during execution. A computer system may be a stand-alone unit or may consist of several inter-connected units.

Conformity	Fulfillment by a product, process or service of all requirements specified.
Customer	An individual or corporate entity who enters into an agreement with an AVF which specifies the terms and conditions for AVF services (of any kind) to be performed.
Declaration of Conformance	A formal statement from a customer assuring that conformity is realized or attainable on the Ada implementation for which validation status is realized.
Host Computer System	A computer system where Ada source programs are transformed into executable form.
Inapplicable test	A test that contains one or more test objectives found to be irrelevant for the given Ada implementation.
ISO	International Organization for Standardization.
Operating System	Software that controls the execution of programs and that provides services such as resource allocation, scheduling, input/output control, and data management. Usually, operating systems are predominantly software, but partial or complete hardware implementations are possible.
Target Computer System	A computer system where the executable form of Ada programs are executed.
Validated Ada Compiler	The compiler of a validated Ada implementation.
Validated Ada Implementation	An Ada implementation that has been validated successfully either by AVF testing or by registration [Pro90].
Validation	The process of checking the conformity of an Ada compiler to the Ada programming language and of issuing a certificate for this implementation.
Withdrawn test	A test found to be incorrect and not used in conformity testing. A test may be incorrect because it has an invalid test objective, fails to meet its test objective, or contains erroneous or illegal use of the Ada programming language.

CHAPTER 2

IMPLEMENTATION DEPENDENCIES

2.1 WITHDRAWN TESTS

The following tests have been withdrawn by the AVO. The rationale for withdrawing each test is available from either the AVO or the AVF. The publication date for this list of withdrawn tests is 14 March 1991.

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

2.2 INAPPLICABLE TESTS

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

IMPLEMENTATION DEPENDENCIES

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

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

The following 20 tests check for the predefined type LONG_INTEGER:

C35404C	C45231C	C45304C	C45411C	C45412C
C45502C	C45503C	C45504C	C45504F	C45611C
C45613C	C45614C	C45631C	C45632C	B52004D
C55B07A	B55B09C	B86001W	C86006C	CD7101F

C35713B, C45423B, B86001T, and C86006H check for the predefined type SHORT_FLOAT.

C35713D and B86001Z check for a predefined floating-point type with a name other than FLOAT, LONG_FLOAT, or SHORT_FLOAT.

C45531M..P (4 tests) and C45532M..P (4 tests) check fixed-point operations for types that require a SYSTEM.MAX_MANTISSA of 47 or greater.

C45624A..B (2 tests) check that the proper exception is raised if MACHINE_OVERFLOWS is FALSE for floating point types; for this implementation, MACHINE_OVERFLOWS is TRUE.

B86001Y checks for a predefined fixed-point type other than DURATION.

C96005B checks for values of type DURATION'BASE that are outside the range of DURATION. There are no such values for this implementation.

CD1009C uses a representation clause specifying a non-default size for a floating-point type.

CD2A84A, CD2A84E, CD2A84I..J (2 tests), and CD2A840 use representation clauses specifying non-default sizes for access types.

The following 264 tests check for sequential, text, and direct access files:

CE2102A..C (3)	CE2102G..H (2)	CE2102K	CE2102N..Y (12)
CE2103C..D (2)	CE2104A..D (4)	CE2105A..B (2)	CE2106A..B (2)
CE2107A..H (8)	CE2107L	CE2108A..H (8)	CE2109A..C (3)
CE2110A..D (4)	CE2111A..I (9)	CE2115A..B (2)	CE2120A..B (2)
CE2201A..C (3)	EE2201D..E (2)	CE2201F..N (9)	CE2203A
CE2204A..D (4)	CE2205A	CE2206A	CE2208B
CE2401A..C (3)	EE2401D	CE2401E..F (2)	EE2401G

CE2401H..L (5)	CE2403A	CE2404A..B (2)	CE2405B
CE2406A	CE2407A..B (2)	CE2408A..B (2)	CE2409A..B (2)
CE2410A..B (2)	CE2411A	CE3102A..C (3)	CE3102F..H (3)
CE3102J..K (2)	CE3103A	CE3104A..C (3)	CE3106A..B (2)
CE3107B	CE3108A..B (2)	CE3109A	CE3110A
CE3111A..B (2)	CE3111D..E (2)	CE3112A..D (4)	CE3114A..B (2)
CE3115A	CE3119A	EE3203A	EE3204A
CE3207A	CE3208A	CE3301A	EE3301B
CE3302A	CE3304A	CE3305A	CE3401A
CE3402A	EE3402B	CE3402C..D (2)	CE3403A..C (3)
CE3403E..F (2)	CE3404B..D (3)	CE3405A	EE3405B
CE3405C..D (2)	CE3406A..D (4)	CE3407A..C (3)	CE3408A..C (3)
CE3409A	CE3409C..E (3)	EE3409F	CE3410A
CE3410C..E (3)	EE3410F	CE3411A	CE3411C
CE3412A	EE3412C	CE3413A..C (3)	CE3414A
CE3602A..D (4)	CE3603A	CE3604A..B (2)	CE3605A..E (5)
CE3606A..B (2)	CE3704A..F (6)	CE3704M..O (3)	CE3705A..E (5)
CE3706D	CE3706F..G (2)	CE3804A..P (16)	CE3805A..B (2)
CE3806A..B (2)	CE3806D..E (2)	CE3806G..H (2)	CE3904A..B (2)
CE3905A..C (3)	CE3905L	CE3906A..C (3)	CE3906E..F (2)

CE2103A, CE2103B, and CE3107A use an illegal file name in an attempt to create a file and expect NAME_ERROR to be raised; this implementation does not support external files and so raises USE_ERROR. (See section 2.3.)

2.3 TEST MODIFICATIONS

Modifications (see section 1.3) were required for 16 tests.

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

B24009A	B33301B	B38003A	B38003B	B38009A	B38009B
B85008G	B85008H	BC1303F	BC3005B	BD2B03A	BD2D03A
BD4003A					

CE2103A, CE2103B, and CE3107A were graded inapplicable by Evaluation Modification as directed by the AVO. The tests abort with an unhandled exception when USE_ERROR is raised on the attempt to create an external file. This is acceptable behavior because this implementation does not support external files (cf. AI-00332).

CHAPTER 3

PROCESSING INFORMATION

3.1 TESTING ENVIRONMENT

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

For a point of contact for technical information about this Ada implementation system, see:

Stewart L. French
Texas Instruments
6550 Chase Oaks Boulevard
Plano TX 75023

(214) 575-6202

For a point of contact for sales information about this Ada implementation system, see:

Stewart L. French
Texas Instruments
6550 Chase Oaks Boulevard
Plano TX 75023

(214) 575-6202

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

3.2 SUMMARY OF TEST RESULTS

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

PROCESSING INFORMATION

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

a) Total Number of Applicable Tests	3565
b) Total Number of Withdrawn Tests	93
c) Processed Inapplicable Tests	47
d) Non-Processed I/O Tests	264
e) Non-Processed Floating-Point Precision Tests	201
f) Total Number of Inapplicable Tests	512
g) Total Number of Tests for ACVC 1.11	4170

The above number of I/O tests were not processed because this implementation does not support a file system. The above number of floating-point tests were not processed because they used floating-point precision exceeding that supported by the implementation. When this compiler was tested, the tests listed in section 2.1 had been withdrawn because of test errors.

3.3 TEST EXECUTION

Version 1.11 of the ACVC comprises 4170 tests. When this compiler was tested, the tests listed in section 2.1 had been withdrawn because of test errors. The AVF determined that 512 tests were inapplicable to this implementation. All inapplicable tests were processed during validation testing except for 201 executable tests that use floating-point precision exceeding that supported by the implementation and 264 executable tests that use file operations not supported by the implementation. In addition, the modified tests mentioned in section 2.3 were also processed.

A magnetic tape containing the customized test suite (see section 1.3) was taken on-site by the validation team for processing. The contents of the magnetic tape were loaded onto the TEMP: disk of the clustered VAX system. The files were copied into the existing directories from which the ACVC tests were run.

Two queues had been set up. One, called ACVC_COMPILE, ran on the MicroVAX 3400 and performed the tasks of compiling and linking to produce the executable load module for the target hardware. The other queue, ACVC_RUN, was used to transfer the load module from the disk to the target. This queue ran on the VAX 6000 which supported the required Ethernet connection.

The files in the support directory were compiled, and the CZ tests were compiled, linked, and run. The rest of the tests were then compiled, linked, and run (as appropriate). Running a test consisted of transferring the executable image to an intermediate host computer, the M2000, on which the target TI DP32 R3000 Processor is directly connected via RS232 lines. The programs were downloaded and run. The results were retrieved onto the

M2000 host computer and transferred back to the VAX cluster. The results were transferred to the Micro-VAX II for printing manually.

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

/LIB=[] - indicates that the program library is this default directory.

Test output, compiler and linker 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.

APPENDIX A

MACRO PARAMETERS

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

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

MACRO PARAMETERS

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

Macro Parameter	Macro Value
$MAX_IN_LEN	499
$ACC_SIZE	32
$ALIGNMENT	4
$COUNT_LAST	2147483647
$DEFAULT_MEM_SIZE	16777216
$DEFAULT_STOR_UNIT	8
$DEFAULT_SYS_NAME	DPOS_II
$DELTA_DOC	0.000000000465661287307739257812 5
$ENTRY_ADDRESS	SYSTEM."+"(16)
$ENTRY_ADDRESS1	SYSTEM."+"(17)
$ENTRY_ADDRESS2	SYSTEM."+"(2)
$FIELD_LAST	2147483647
$FILE_TERMINATOR	' '
$FIXED_NAME	NO_SUCH_FIXED_TYPE
$FLOAT_NAME	NO_SUCH_TYPE
$FORM_STRING	""
$FORM_STRING2	"CANNOT_RESTRICT_FILE_CAPACITY"
$GREATER_THAN_DURATION	100000.0
$GREATER_THAN_DURATION_BASE_LAST	10000000.0
$GREATER_THAN_FLOAT_BASE_LAST	1.8E+308
$GREATER_THAN_FLOAT_SAFE_LARGE	5.0E307

```
$GREATER_THAN_SHORT_FLOAT_SAFE_LARGE
                        9.0E37

$HIGH_PRIORITY          99

$ILLEGAL_EXTERNAL_FILE_NAME1
                        "/illegal/file_name/2{]$%2102C.DAT"

$ILLEGAL_EXTERNAL_FILE_NAME2
                        "/illegal/file_name/CE2102C*.DAT"

$INAPPROPRIATE_LINE_LENGTH
                        -1

$INAPPROPRIATE_PAGE_LENGTH
                        -1

$INCLUDE_PRAGMA1        PRAGMA INCLUDE ("A28006D1.TST")

$INCLUDE_PRAGMA2        PRAGMA INCLUDE ("B28006F1.TST")

$INTEGER_FIRST          -2147483648

$INTEGER_LAST           2147483647

$INTEGER_LAST_PLUS_1    2147483648

$INTERFACE_LANGUAGE     C

$LESS_THAN_DURATION     -100000.0

$LESS_THAN_DURATION_BASE_FIRST
                        -10000000.0

$LINE_TERMINATOR        ASCII.LF

$LOW_PRIORITY           0

$MACHINE_CODE_STATEMENT
                        CODE_0'(OP=>NOP);

$MACHINE_CODE_TYPE      CODE_0

$MANTISSA_DOC           31

$MAX_DIGITS             15

$MAX_INT                2147483647

$MAX_INT_PLUS_1         2147483648

$MIN_INT                -2147483648
```

MACRO PARAMETERS

$NAME	TINY_INTEGER
$NAME_LIST	DPOS_II
$NAME_SPECIFICATION1	[ACVC1_11.C.E]X2120A
$NAME_SPECIFICATION2	[ACVC1_11.C.E]X2121B
$NAME_SPECIFICATION3	[ACVC1_11.C.E]X3119A
$NEG_BASED_INT	16#FFFFFFFD#
$NEW_MEM_SIZE	16777216
$NEW_STOR_UNIT	8
$NEW_SYS_NAME	DPOS_II
$PAGE_TERMINATOR	ASCII.LF & ASCII.FF
$RECORD_DEFINITION	RECORD NULL; END RECORD;
$RECORD_NAME	NO_SUCH_MACHINE_CODE_TYPE
$TASK_SIZE	32
$TASK_STORAGE_SIZE	1024
$TICK	0.01
$VARIABLE_ADDRESS	VAR_1'ADDRESS
$VARIABLE_ADDRESS1	VAR_2'ADDRESS
$VARIABLE_ADDRESS2	VAR_3'ADDRESS
$YOUR_PRAGMA	PASSIVE

APPENDIX B

COMPILATION SYSTEM OPTIONS

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

Compile and Link Options

```
VADS ADA source_file [, ...]

Qualifiers

/APPEND             Must be used with /OUTPUT.  It appends output to
                    file_name.

/CG_OPTIONS=("option" [, ...])
                    Pass command qualifiers and parameters to the code
                    generator.

/DEBUG=G            Write out the gnrx.lib file in ASCII.

/DEFINE="identifier:type=value", ...)
                    Define identifier of a specified type and value.

/DEPENDENCIES       Analyze for dependencies only; no link will be
                    performed if this option is given.

/ERRORS[=(option [, ...])]
                    Process compilation error messages using the ERROR
                    tool and direct the output to SYS$OUTPUT.

                    OPTIONS:

                    LISTING         List entire input file.
                    EDITOR="editor"
                                    Insert error messages into the source
                                    file and call a text editor.
```

OUTPUT[=file_name]
Direct error processed output to the specified file.
BRIEF list only the affected lines

/KEEP_IL Keep the Intermediate Language (IL) file produced by the compiler front end.

/LIBRARY=library_name
Operate in VADS library "library_name".

/MAIN[=unit_name]
Produce an executable program using the named unit as the main program.

/NOOPTIMIZE Do not optimize.

/WARNINGS Print warning diagnostics.

/OPTIMIZE[=number]
Invoke the code optimizer (OPTIM3). An optional digit provides the level of optimization. The range is 0..9 with 9 being the maximum.

/OUTPUT=file_name
Direct the output to file_name.

/PRE_PROCESS Invoke the Ada Preprocessor.

/RECOMPILE_LIBRARY=VADS_library
Force analysis of all generic instantiations causing reinstantiation of any that are out of date.

/SHOW Show the name of the tool executable but do not execute it.

/SHOW_ALL Print the name of the front end, code generator, optimizer, linker and list the tools that will be invoked.

/SUPPRESS Apply PRAGMA SUPPRESS for all checks to the entire compilation.

/TIMING Print timing information for the compilation.

/VERBOSE Print information for the compilation.

APPENDIX C

APPENDIX F OF THE Ada STANDARD

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

```
package STANDARD is
    ..........

   type INTEGER is range -2147483648 .. 2147483647;
   type SHORT_INTEGER is range -32768 .. 32767;
   type TINY_INTEGER is range -128 .. 127;

   type FLOAT is digits 6 range -3.40282E+38 .. 3.40282E+38;
   type LONG_FLOAT is digits 15 range -8.988465674312E+307 ..
                                       8.988465674312E+308;

   type DURATION is delta 0.001 range -2147483.648 ..
                                           2147483.647;
    ..........

end STANDARD;
```

ATTACHMENT I

APPENDIX F. Implementation-Dependent Characteristics

1. Implementation-Dependent Pragmas

1.1 INLINE ONLY Pragma

The INLINE_ONLY pragma, when used in the same way as pragma INLINE, indicates to the compiler that the subprogram must always be inlined. This pragma also suppresses the generation of a callable version of the routine which save code space.

1.2. BUILT IN Pragma

The BUILT_IN pragma is used in the implementation of some predefined Ada packages, but provides no user access. It is used only to implement code bodies for which no actual Ada body can be provided.

1.3. SHARE CODE Pragma

The SHARE_CODE pragma takes the name of a generic instantiation or a generic unit as the first argument and one of the identifiers TRUE or FALSE as the second argument. This pragma is only allowed immediately at the place of a declarative item in a declarative part or package specification, or after a library unit in a compilation, but before any subsequent compilation unit.

When the first argument is a generic unit, the pragma applies to all instantiations of that generic. When the first argument is the name of a generic instantiation, the pragma applies only to the specified instantiation, or overloaded instantiations.

If the second argument is TRUE, the compiler will try to share code generated for a generic instantiation with code generated for other instantiations of the same generic. When the second argument is FALSE, each instantiation will get a unique copy of the generated code. The extent to which code is shared between instantiations depends on this pragma and the kind of generic formal parameters declared for the generic unit.

1.4. NO IMAGE Pragma

The pragma suppresses the generation of the image array used for the IMAGE attribute of enumeration types. This eliminates the overhead required to store the array in the executable image. An attempt to use the IMAGE attribute will result in a compile-time warning and PROGRAM_ERROR raised at runtime.

1.5. EXTERNAL NAME Pragma

The EXTERNAL_NAME pragma takes the name of a subprogram or variable defined in Ada and allows the user to specify a different external name that may be used to reference the entity from other languages. The pragma is allowed at the place of a declarative item in a package specification and must apply to an object declared earlier in the same package specification.

1.6. INTERFACE NAME Pragma

The INTERFACE_NAME pragma takes the name of a a variable or a subprogram defined in another language and allows it to be referenced directly in Ada. The pragma will replace all occurrences of the variable name or the subprogram name with an external reference to the second, link_argument. The pragma is allowed at the place of a declarative item in a package specification and must apply to an object declared earlier in the same package specification. The object must be declared as a scalar or an access type. The object cannot be any of the following:

a loop variable,
a constant,
an initialized variable,
an array, or
a record.

1.7. IMPLICIT CODE Pragma

Takes one of the identifiers ON or OFF as the single argument. This pragma is only allowed within a machine code procedure. It specifies that implicit code generated by the compiler be allowed or disallowed. A warning is issued if OFF is used and any implicit code needs to be generated. The default is ON.

1.3. LINK WITH Pragma

LINK_WITH pragma can be used to pass arguments to the target linker. It may appear in any declarative part and have only

one argument, a constant string expression. This argument is passed to the target linker whenever the unit containing the pragma is included in a link.

1.9. NON REENTRANT Pragma

NON-REENTRANT pragma takes one argument which can be the name of a library subprogram or a subprogram declared immediately within a library spec or body. This pragma indicates to the compiler that the subprogram will not be called recursively allowing the compiler to perform specific optimizations.

1.10. PASSIVE Pragma

PASSIVE pragma can be applied to a task or a task type declared immediately within a library package spec or body. It directs the compiler to optimize certain tasking optimizations.

1.11. VOLATILE Pragma

VOLATILE pragma guarantees that loads and stores to the named objects will be performed as expected after optimization.

2. Implementation of Predefined Pragmas

2.1. CONTROLLED

This pragma is recognized by the implementation but has no effect.

2.2. ELABORATE

This pragma is implemented as described in Appendix B of the Ada RM.

2.3. INLINE

This pragma is implemented as described in Appendix B of the Ada RM.

2.4. INTERFACE

This pragma supports calls to 'C' and FORTRAN functions. The Ada subprograms can be either functions or procedures. The types of parameters and the result type for functions must be scalar, access, or the predefined type ADDRESS in SYSTEM. All parameters must have mode IN. Record and array objects can be passed by reference using the ADDRESS attribute.

2.5. LIST

This pragma is implemented as described in Appendix B of the Ada RM.

2.6. MEMORY SIZE

This pragma is recognized by the implementation. The implementation does not allow SYSTEM to be modified by means of pragmas; the SYSTEM package must be recompiled.

2.7. NOT ELABORATED

This pragma can only appear in a library package specification. It indicates that the package will not be elaborated because it is either part of the RTS, a configuration package, or an Ada package that is referenced from a language other than Ada. The presence of this pragma suppresses the generation of elaboration code and issues warnings if elaboration code is required.

2.8. OPTIMIZE

This pragma is recognized by the implementation but has no effect.

2.9. PACK

This pragma will cause the compiler to choose a non-aligned representation for composite types. It will not cause objects to be packed at the bit level.

2.10. PAGE

This pragma is implemented as described in Appendix B of the Ada RM.

2.11. PRIORITY

This pragma is implemented as described in Appendix B of the Ada RM.

2.12. SHARED

This pragma is recognized by the implementation but has no effect.

2.13. STORAGE_UNIT

This pragma is recognized by the implementation. The implementation does not allow SYSTEM to be modified by means of pragmas, the SYSTEM package must be recompiled.

2.14. SUPPRESS

This pragma is implemented as described, except that RANGE_CHECK and DIVISION_CHECK cannot be suppressed.

2.15. SYSTEM_NAME

This pragma is recognized by the implementation. The implementation does not allow SYSTEM to be modified by means of pragmas, the SYSTEM package must be recompiled.

3. Implementation-Dependent Attributes

3.1. P'REF

This attribute can be used to convert an integer to an address.

3.2. 'TASK_ID

For a task object T, T'TASK_ID yields the unique task ID associated with the task.

3.3. Restrictions on 'Main' programs

TI Ada requires that a 'Main' program must be a non-generic subprogram that is either a procedure or a function returning an Ada STANDARD.INTEGER.

4. Specification of Package SYSTEM

```
with UNSIGNED_TYPES;

package SYSTEM is

        pragma suppress (ALL_CHECKS);
        pragma suppress (EXCEPTION_TABLES);
        pragma not_elaborated;

        type NAME is (DPOS_II);

        SYSTEM_NAME : constant NAME := DPOS_II;

        STORAGE_UNIT : constant := 8;
        MEMORY_SIZE : constant := 16_777_216;

        -- System-Dependent Named Numbers

        MIN_INT          : constant := -2_147_483_648;
        MAX_INT          : constant := 2_147_483_647;
        MAX_DIGITS       : constant := 15;
        MAX_MANTISSA     : constant := 31;
        FINE_DELTA       : constant := 2.0**(-31);
        TICK             : constant := 0.01;

        -- Other System-dependent Declarations

        subtype PRIORITY is INTEGER range 1 .. 245;

        MAX_REC_SIZE : INTEGER := 64*1024;

        type ADDRESS is private;

        NO_ADDR : constant ADDRESS;


        function ">" (A: ADDRESS; B: ADDRESS) return BOOLEAN;
        function "<" (A: ADDRESS; B: ADDRESS) return BOOLEAN;
        function ">="(A: ADDRESS; B: ADDRESS) return BOOLEAN;
        function "<="(A: ADDRESS; B: ADDRESS) return BOOLEAN;
        function "-" (A: ADDRESS; B: ADDRESS) return INTEGER;
        function "+" (A: ADDRESS; I: INTEGER) return ADDRESS;
        function "-" (A: ADDRESS; I: INTEGER) return ADDRESS;

        function "+" (I: UNSIGNED_TYPES.UNSIGNED_INTEGER)
                     return ADDRESS;
```

```
        function PHYSICAL_ADDRESS
                (I: UNSIGNED_TYPES.UNSIGNED_INTEGER)
                     return ADDRESS renames "+";

        type TASK_ID is private;
        NO_TASK_ID : constant TASK_ID;

private

        type ADDRESS is new UNSIGNED_TYPES.UNSIGNED_INTEGER;

        NO_ADDR : constant ADDRESS := 0;

        pragma BUILT_IN (">");
        pragma BUILT_IN ("<");
        pragma BUILT_IN (">=");
        pragma BUILT_IN ("<=");
        pragma BUILT_IN ("-");
        pragma BUILT_IN ("+");

        type TASK_ID is new UNSIGNED_TYPES.UNSIGNED_INTEGER;
        NO_TASK_ID : constant TASK_ID := 0;

end SYSTEM;
```

5. Restrictions On Representation Clauses

5.1. Pragma PACK

In the absence of pragma PACK, record components are padded so as to provide for efficient access by the target hardware; pragma PACK applied to a record eliminates the padding where possible. Pragma PACK has no other effect on the storage allocated for record components a record representation is required.

5.2. Record Representation Clauses

For scalar types, a representation clause will pack to the number of bits required to represent the range of the subtype. A record representation applied to a composite type will not cause the object to be packed to fit in the space required. An explicit representation clause must be given for the component type. An error will be issued if there is insufficient space allocated.

5.3. Address Clauses

Address clauses are supported for variables and constants.

5.4. Interrupts

Interrupt entries are not supported.

5.5. Representation Attributes

The ADDRESS attribute is supported for the following entities, but a meaningless value is returned.

Packages
Tasks
Entries

6. Conventions for Implementation-generated Names

There are no implementation-generated names.

7. Interpretation of Expressions in Address Clauses

Address clauses are supported for constants and variables.

8. Restrictions on Unchecked Conversions

None.

9. Restrictions on Unchecked Deallocations

None.

10. Implementation Characteristics of I/O Packages

Instantiations of DIRECT_IO use the value MAX_REC_SIZE as the record size (expressed in STORAGE_UNITS) when the size of ELEMENT_TYPE exceeds that value. For example, for unconstrained arrays such as string where ELEMENT_TYPE'SIZE is very large, MAX_REC_SIZE is used instead. MAX_RECORD_SIZE is defined in SYSTEM and can be changed by a program before instantiating DIRECT_IO to provide an upper limit on the record size. In any case, the maximum size supported is 1024 x 1024 x STORAGE_UNIT bits. DIRECT_IO will raise USE_ERROR if MAX_REC_SIZE exceeds this absolute limit.

Instantiations of SEQUENTIAL_IO use the value MAX_REC_SIZE as the record size (expressed in STORAGE_UNITS) when the size of ELEMENT_TYPE exceeds that value. For example for unconstrained arrays, such as string where ELEMENT_TYPE'SIZE is very large, MAX_REC_SIZE is used instead. MAX_RECORD_SIZE is defined in SYSTEM and can be changed by a program before instantiating INTEGER_IO to provide an upper limit on the record size. SEQUENTIAL_IO imposes no limit on MAX_REC_SIZE.

The DP32 target system does not support text files. Calls to I/O services which operate on files will raise appropriate exceptions, except for PUT to STANDARD_OUTPUT.

11. Implementation Limits

The following limits are actually enforced by the implementation. It is not intended to imply that resources up to or even near these limits are available to every program.

11.1. Line Length

The implementation supports a maximum line length of 500 characters including the end of line character.

11.2. Record and Array Sizes

The maximum size of a statically sized array type is 24,000,000 x STORAGE_UNITS. The maximum size of a statically sized record type is 24,000,000 x STORAGE_UNITS. A record type or array type declaration that exceeds these limits will generate a warning message.

11.3. Default Stack Size for Tasks

In the absence of an explicit STORAGE_SIZE length specification, every task except the main program is allocated a fixed size stack of 16,384 STORAGE_UNITS. This is the value returned by T'STORAGE_SIZE for a task type T.

11.4. Default Collection Size

In the absence of an explicit STORAGE_SIZE length attribute, the default collection size for an access type is 100 times the size of the designated type. This is the value returned by T'STORAGE_SIZE for an access type T.

11.5. Limit on Declared Objects

There is an absolute limit of 6,000,000 x STORAGE_UNITS for objects declared statically within a compilation unit. If this value is exceeded, the compiler will terminate the compilation of the unit with a FATAL error message.