REPORT DOCUMENTATION PAGE

Form Approved
OPM No. 0704-0188

AD-A248 213

Pu... ne... He... Mi... ...our per response, including the time for reviewing instructions, searching existing data sources gathering and maintaining the data ...s 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.	...ORT DATE	3. REPORT TYPE AND DATES COVERED
		Final: 01 Jun 1991 to 01 Jun 1993

4. TITLE AND SUBTITLE
Validation Summary Report: Intermetrics, Inc., Intermetrics MVS Ada, Version 7.0, Amdahl 5890/180E, MVS/XA Release 2.2 (Host & Target) 910622W1.11170

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

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

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

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*
Intermetrics, Inc., Intermetrics MVS Ada, Version 7.0, Wright-Patterson AFB, OH,Amdahl 5890/180E, MVS/XA Release 2.2 (Host & Target), ACVC 1.11.

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

Certificate Information

The following Ada implementation was tested and determined to pass ACVC 1.11. Testing was completed on 22 June 1991.

Compiler Name and Version: Intermetrics MVS Ada, Version 7.0

Host Computer System: Amdahl 5890/180E, MVS/XA Release 2.2

Target Computer System: Amdahl 5890/180E, MVS/XA Release 2.2

Customer Agreement Number: 91-05-20-INT

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

As a result of this validation effort, Validation Certificate 910622W1.11170 is awarded to Intermetrics, Inc. This certificate expires on 1 March 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 and Software Engineering Division
Institute for Defense Analyses
Alexandria VA 22311

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

92 3 18 027

92-06993

AVF Control Number: AVF-VSR-478.0691
4 February 1992
91-05-20-INT

Ada COMPILER
VALIDATION SUMMARY REPORT:
Certificate Number: 910622W1.11170
Intermetrics, Inc.
Intermetrics MVS Ada, Version 7.0
Amdahl 5890/180E => Amdahl 5890/180E

Prepared By:
Ada_Validation_Facility
ASD/SCEL
Wright-Patterson AFB OH 45433-6503

Accesion For		
NTIS CRA&I		☑
DTIC TAB		☐
Unannounced		☐
Justification		
By		
Distribution /		
Availability Codes		
Dist	Avail and / or Special	
A-1		

Certificate Information

The following Ada implementation was tested and determined to pass ACVC 1.11. Testing was completed on 22 June 1991.

Compiler Name and Version: Intermetrics MVS Ada, Version 7.0

Host Computer System: Amdahl 5890/180E, MVS/XA Release 2.2

Target Computer System: Amdahl 5890/180E, MVS/XA Release 2.2

Customer Agreement Number: 91-05-20-INT

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

As a result of this validation effort, Validation Certificate 910622W1.11170 is awarded to Intermetrics, Inc. This certificate expires on 1 March 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

for Ada Validation Organization
Director, Computer and 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: Intermetrics, Inc., Cambridge, MA

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

ACVC Version: 1.11

Ada Implementation

Compiler Name and Version: Intermetrics MVS Ada Compiler, Version 7.0
Host Computer System: Amdahl 5890/180E, MVS/XA Release 2.2
Target Computer System: same

Customer's Declaration

I, the undersigned, representing Intermetrics, Inc., declare that Intermetrics, Inc. has no knowledge of deliberate deviations from the Ada Language Standard ANSI/MIL-STD-1815A in the implementation listed in this declaration. I declare that Intermetrics, Inc. is the owner of record of the above implementation and the certificates shall be awarded in Intermetrics' corporate name.

[signature] Date: 5/30/91

Dennis Struble, Deputy General Manager,
Development Systems Group, Intermetrics, Inc.

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
Computer and Software Engineering Division
Institute for Defense Analyses
1801 North Beauregard Street
Alexandria VA 22311-1772

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 removing 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.
LRM	The Ada standard, or Language Reference Manual, published as ANSI/MIL-STD-1815A-1983 and ISO 8652-1987. Citations from the LRM take the form "<section>.<subsection>:<paragraph>."
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 3 May 1991.

E28005C	B28006C	C34006D	C35508I	C35508J	C35508M
C35508N	C35702A	C35702B	B41308B	C43004A	C45114A
C45346A	C45612A	C45612B	C45612C	C45651A	C46022A
B49008A	B49008B	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 the 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 Ada Commentaries are included as appropriate.

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 21 tests check for the predefined type SHORT_INTEGER; for this implementation, there is no such type:

C35404B	B36105C	C45231B	C45304B	C45411B
C45412B	C45502B	C45503B	C45504B	C45504E
C45611B	C45613B	C45614B	C45631B	C45632B
B52004E	C55B07B	B55B09D	B86001V	C86006D
CD7101E				

The following 20 tests check for the predefined type LONG_INTEGER; for this implementation, there is no such type:

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

C35404D, C45231D, B86001X, C86006E, and CD7101G check for a predefined integer type with a name other than INTEGER, LONG_INTEGER, or SHORT_INTEGER; for this implementation, there is no such type.

C35713C, B86001U, and C86006G check for the predefined type LONG_FLOAT; for this implementation, there is no such type.

C35713D and B86001Z check for a predefined floating-point type with a name other than FLOAT, LONG_FLOAT, or SHORT_FLOAT; for this implementation, there is no such type.

C45423A..B (2 tests), C45523A, and C45622A check that the proper exception is raised if MACHINE_OVERFLOWS is TRUE and the results of various floating-point operations lie outside the range of the base type; for this implementation, MACHINE_OVERFLOWS is FALSE.

C45531M..P and C45532M..P (8 tests) check fixed-point operations for types that require a SYSTEM.MAX_MANTISSA of 47 or greater; for this implementation, MAX_MANTISSA is less than 47.

C46013B, C46031B, C46033B, and C46034B contain length clauses that specify values for 'SMALL that are not powers of two or ten; this implementation does not support such values for 'SMALL.

D55A03E..H (4 tests) uses 31 levels of loop nesting; this level of loop nesting exceeds the capacity of the compiler.

D56001B uses 65 levels of block nesting; this level of block nesting exceeds the capacity of the compiler.

B86001Y uses the name of a predefined fixed-point type other than type DURATION; for this implementation, there is no such type.

CA2009C and CA2009F check whether a generic unit can be instantiated before the separate compilation of its body (and any of its subunits); this implementation requires that the body and subunits of generic be in the same compilation as the specification if instantiations precede them. (See section 2.3.)

CD1009C checks whether a length clause can specify a non-default size for a floating-point type; this implementation does not support a floating-point representation of other than 32 or 64 bits.

CD2A84A, CD2A84E, CD2A84I..J (2 tests), and CD2A84O use length clauses to specify non-default sizes for access types; this implementation does not support such sizes.

BD8001A, BD8003A, BD8004A..B (2 tests), and AD8011A use machine code insertions; this implementation provides no package MACHINE_CODE.

AE2101C and EE2201D..E (2 tests) use instantiations of package SEQUENTIAL_IO with unconstrained array types and record types with discriminants without defaults; these instantiations are rejected by this compiler.

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.

The tests listed in the following table check that USE_ERROR is raised if the given file operations are not supported for the given combination of mode and access method; this implementation supports these operations.

Test	File Operation	Mode	File Access Method
CE2102D	CREATE	IN_FILE	SEQUENTIAL_IO
CE2102E	CREATE	OUT_FILE	SEQUENTIAL_IO
CE2102F	CREATE	INOUT_FILE	DIRECT_IO
CE2102I	CREATE	IN_FILE	DIRECT_IO
CE2102J	CREATE	OUT_FILE	DIRECT_IO
CE2102N	OPEN	IN_FILE	SEQUENTIAL_IO
CE2102O	RESET	IN_FILE	SEQUENTIAL_IO
CE2102P	OPEN	OUT_FILE	SEQUENTIAL_IO
CE2102Q	RESET	OUT_FILE	SEQUENTIAL_IO
CE2102R	OPEN	INOUT_FILE	DIRECT_IO

CE2102S	RESET	INOUT_FILE	DIRECT_IO
CE2102T	OPEN	IN_FILE	DIRECT_IO
CE2102U	RESET	IN_FILE	DIRECT_IO
CE2102V	OPEN	OUT_FILE	DIRECT_IO
CE2102W	RESET	OUT_FILE	DIRECT_IO
CE3102E	CREATE	IN_FILE	TEXT_IO
CE3102F	RESET	Any Mode	TEXT_IO
CE3102G	DELETE	--------	TEXT_IO
CE3102I	CREATE	OUT_FILE	TEXT_IO
CE3102J	OPEN	IN_FILE	TEXT_IO
CE3102K	OPEN	OUT_FILE	TEXT_IO

CE2107C..D (2 tests) apply function NAME to temporary sequential files in an attempt to associate multiple internal files with the same external file; for this implementation, temporary files have no name and so USE_ERROR is raised by NAME.

CE2107E checks operations on direct and sequential temporary files when files of both kinds are associated with the same external file; this implementation does not support simultaneous association of an external file to files of different file types. (See section 2.3.)

CE2107L checks operations on direct and sequential files when files of both kinds are associated with the same external file; USE_ERROR is raised when this association is attempted.

CE2108B and CE3112B use the names of temporary sequential and text files that were created in other tests in order to check that the temporary files are not accessible after the completion of those tests; for this implementation, temporary sequential and text files have no name.

CE2203A checks that WRITE raises USE_ERROR if the capacity of an external sequential file is exceeded; this implementation cannot restrict file capacity.

CE2403A checks that WRITE raises USE_ERROR if the capacity of an external direct file is exceeded; this implementation cannot restrict file capacity.

CE3111B and CE3115A associate multiple internal text files with the same external file and attempt to read from one file what was written to the other, which is assumed to be immediately available; this implementation buffers output. (See section 2.3.)

CE3304A checks that SET_LINE_LENGTH and SET_PAGE_LENGTH raise USE_ERROR if they specify an inappropriate value for the external file; there are no inappropriate values for this implementation.

CE3413B checks that PAGE raises LAYOUT_ERROR when the value of the page number exceeds COUNT'LAST; for this implementation, the value of COUNT'LAST is greater than 150000, making the checking of this objective impractical.

2.3 TEST MODIFICATIONS

Modifications (see section 1.3) were required for seven 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.

BA1101C   BC3205D

CA2009C and CA2009F were graded inapplicable by Evaluation Modification as directed by the AVO. These tests contain instantiations of a generic unit prior to the separate compilation of that unit's body; as allowed by AI-257, this implementation requires that the bodies of a generic unit be in the same compilation if instantiations of that unit precede the bodies. The instantiations were rejected at compile time.

CE2107E was graded inapplicable based on the Report.Result output, without critical consideration to the intermediate, Report.Not_Applicable output which was generated by line 56 ("NAME_ERROR RAISED; SEQUENTIAL CREATE WITH OUT_FILE MODE"). After validation testing was completed, AVO and AVF analysis led to the conclusion that the particular Report.Not_Applicable output was an unexpected consequence, possibly due to the existence of a file named "X2107E" in the working directory. Upon subsequent consultation the customer confirmed that the expected Report.Not_Applicable output, from line 76 ("UNABLE TO ASSOCIATE A SEQUENTIAL FILE AND A DIRECT FILE TO THE SAME EXTERNAL FILE"), is generated by processing the test.

CE3111B and CE3115A were graded inapplicable by Evaluation Modification as directed by the AVO. The tests assume that output from one internal file is unbuffered and may be immediately read by another file that shares the same external file. This implementation raises END_ERROR on the attempts to read at lines 87 and 101, respectively.

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 technical and sales information about this Ada implementation, contact:

Mr. Mike Ryer
Intermetrics, Inc.
733 Concord Avenue
Cambridge, MA 02138-1002

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].

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

The list of items below gives the number of ACVC tests in various categories. All tests were processed, except those that were withdrawn because of test errors (item b; see section 2.1), those that require a floating-point precision that exceeds the implementation's maximum precision (item e; see section 2.2), and those that depend on the support of a file system — if none is supported (item d). All tests passed, except those that are listed in sections 2.1 and 2.2 (counted in items b and f, below).

PROCESSING INFORMATION

a)	Total Number of Applicable Tests	3749	
b)	Total Number of Withdrawn Tests	94	
c)	Processed Inapplicable Tests	125	
d)	Non-Processed I/O Tests	0	
e)	Non-Processed Floating-Point Precision Tests	201	
f)	Total Number of Inapplicable Tests	326	(c+d+e)
g)	Total Number of Tests for ACVC 1.11	4170	(a+b+f)

3.3 TEST EXECUTION

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 directly onto the host computer.

After the test files were loaded onto the host computer, the full set of tests was processed by the Ada implementation.

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 default options were invoked implicitly for validation testing during this test.

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
$MAX_IN_LEN	255 — Value of V
$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') & '"'

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

Macro Parameter	Macro Value
$ACC_SIZE	32
$ALIGNMENT	4
$COUNT_LAST	2_147_483_647
$DEFAULT_MEM_SIZE	2**31
$DEFAULT_STOR_UNIT	8
$DEFAULT_SYS_NAME	UTS
$DELTA_DOC	2.0**(-31)
$ENTRY_ADDRESS	SYSTEM.MAKE_ADDRESS(16#40#)
$ENTRY_ADDRESS1	SYSTEM.MAKE_ADDRESS(16#80#)
$ENTRY_ADDRESS2	SYSTEM.MAKE_ADDRESS(16#100#)
$FIELD_LAST	2_147_483_647
$FILE_TERMINATOR	TEST_WITHDRAWN
$FIXED_NAME	NO_SUCH_FIXED_TYPE
$FLOAT_NAME	NO_SUCH_FLOAT_TYPE
$FORM_STRING	""
$FORM_STRING2	CANNOT_RESTRICT_FILE_CAPACITY
$GREATER_THAN_DURATION	90_000.0
$GREATER_THAN_DURATION_BASE_LAST	10_000_000.0
$GREATER_THAN_FLOAT_BASE_LAST	1.0E+63
$GREATER_THAN_FLOAT_SAFE_LARGE	16#0.FFFFFFFFFFFE1#E+63

$GREATER_THAN_SHORT_FLOAT_SAFE_LARGE	16#0.FFFFF9#E+63
$HIGH_PRIORITY	127
$ILLEGAL_EXTERNAL_FILE_NAME1	BAD-CHARAC/TER
$ILLEGAL_EXTERNAL_FILE_NAME2	NO/MUCH-TOO-LONG-NAME-FOR-A-FILE
$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	2_147_483_648
$INTERFACE_LANGUAGE	AIE_ASSEMBLER
$LESS_THAN_DURATION	-90_000.0
$LESS_THAN_DURATION_BASE_FIRST	-10_000_000.0
$LINE_TERMINATOR	ASCII.LF
$LOW_PRIORITY	-127
$MACHINE_CODE_STATEMENT	NULL;
$MACHINE_CODE_TYPE	NO_SUCH_TYPE
$MANTISSA_DOC	31
$MAX_DIGITS	15
$MAX_INT	2147483647
$MAX_INT_PLUS_1	2147483648
$MIN_INT	-2147483648
$NAME	NO_SUCH_INTEGER_TYPE

$NAME_LIST	UTS,MVS,CMS,PRIME50,SPERRY1100,MIL_STD_1750A
$NAME_SPECIFICATION1	=BIGGA.X2120A
$NAME_SPECIFICATION2	=BIGGA.X2120B
$NAME_SPECIFICATION3	=BIGGA.X3119A
$NEG_BASED_INT	16#FFFFFFFE#
$NEW_MEM_SIZE	TEST_WITHDRAWN
$NEW_STOR_UNIT	8
$NEW_SYS_NAME	TEST_WITHDRAWN
$PAGE_TERMINATOR	TEST_WITHDRAWN
$RECORD_DEFINITION	TEST_WITHDRAWN
$RECORD_NAME	TEST_WITHDRAWN
$TASK_SIZE	96
$TASK_STORAGE_SIZE	1024
$TICK	1.0E-3
$VARIABLE_ADDRESS	FCNDECL.VARIABLE_ADDRESS;
$VARIABLE_ADDRESS1	FCNDECL.VARIABLE_ADDRESS1;
$VARIABLE_ADDRESS2	FCNDECL.VARIABLE_ADDRESS2;
$YOUR_PRAGMA	TEST_WITHDRAWN

APPENDIX B

COMPILATION SYSTEM OPTIONS

The compiler 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.

The compilation defaults are as follows:

SRCLIB='I2ADA.ACVC111.ADA'
HLI=I2ADA
VERSION='WORK'
TSOID=I2ADA
SIZE='6000K'
AMSLIST='DUMMY'
ERRDCB='(RECFM=FB,LRECL=120,BLKSIZE=120)'
LISTDCB='(RECFM=VB,LRECL=136,BLKSIZE=140)'
SYSOUT='*' (WHICH DEFAULTS TO MSGCLASS)

For more information please see the compilation options listing that follows.

COMPILER OPTIONS

Note that the compiler can either take a source file or to increase performance, a "script" file containing a list of files and options.

```
//*
//* Ada Compile (single phase compiler).
//*
//ADA1C    PROC SRCLIB='I2ADA.ACVC111.ADA',
//              SRCMEM=,            PDS MEMBER CONTAINING ADA SOURCE
//              HLI=I2ADA,          HIGH LEVEL INDEX,
//              VERSION='WORK',     LEVEL OF THE COMPILER
//              PTN=,               PARTITION NAME
//              TSOID=I2ADA,        HIGH LEVEL QUALIFIER FOR PGMLIB
//              MONO=,              COMPILER OPTIONS
//              SIZE='6000K',       REGION SIZE
//              AMSLIST='DUMMY',    AMS LISTING FILE
//              ERRDCB='(RECFM=FB,LRECL=120,BLKSIZE=120)',
//              LISTDCB='(RECFM=VB,LRECL=136,BLKSIZE=140)',
//              SYSOUT='*'          OUTPUT CLASS (DEFAULTS TO MSGCLASS)
//*
```

```
//*
//* Ada Compile (single phase compiler).
//*
//*  Compiles a chapter of ACVC tests using a script.
//*
//ADA1C    PROC HLI=I2ADA,          HIGH LEVEL INDEX,
//              VERSION='WORK',     LEVEL OF THE COMPILER
//              CHAPTER='',         CHAPTER OF ACVC SCRIPT TO COMPILE
//              PTN=,               PARTITION NAME
//              SIZE='6000K',       REGION SIZE
//              AMSLIST='DUMMY',    AMS LISTING FILE
//              ERRDCB='(RECFM=FB,LRECL=120,BLKSIZE=120)',
//              LISTDCB='(RECFM=VB,LRECL=136,BLKSIZE=140)',
//              SYSOUT='*'          OUTPUT CLASS (DEFAULTS TO MSGCLASS)
//*
```

LINKER OPTIONS

The 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 linker documentation and not to this report.

The linker defaults are as follows:

```
LOAD=ACVC1
TSOID=I2ADA
USERLIB='I2ADA.NULLPDS.LOAD'
VIO=VIO
HLI=I2ADA
VERSION='WORK'
SIZE='6000K'
AMSLIST='DUMMY'
ERRDCB='(RECFM=FB,LRECL=120,BLKSIZE=120)'
LISTDCB='(RECFM=VB,LRECL=136,BLKSIZE=140)'
SYSOUT='*'  (WHICH DEFAULTS TO MSGCLASS)
```

For more information please see the linker options listing that follows.

LINKER OPTIONS

```
//*
//* Ada Link and Go
//*
//ACVCLG  PROC LUNIT=,              MAIN PROGRAM
//             PTN=,                PARTITION NAME
//             MEM,                 EXECUTABLE NAME
//             LOAD=ACVC1,          MIDDLE QUALIFIER FOR LOADLIB
//             TSOID=I2ADA,         HIGH LEVEL QUALIFIER FOR LOADLIB
//             USERLIB='I2ADA.NULLPDS.LOAD', RTS TEST LIBRARY
//             VIO=VIO,             SCRATCH UNIT
//             HLI=I2ADA,           HIGH LEVEL INDEX,
//             VERSION='WORK',      LEVEL OF THE COMPILER
//             SIZE='6000K',        REGION SIZE
//             AMSLIST='DUMMY',     AMS LISTING FILE
//              ERRDCB='(RECFM=FB,LRECL=120,BLKSIZE=120)',
//              LISTDCB='(RECFM=VB,LRECL=136,BLKSIZE=140)',
//              SYSOUT='*'         OUTPUT CLASS (DEFAULTS TO MSGCLASS)
//*
```

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 FLOAT is digits 15
     range -16#0.ffffffffffffff#e63 .. 16#0.ffffffffffffff#e63;
   type SHORT_FLOAT is digits 6 range -16#0.ffffff#e63 .. 16#0.ffffff#e63;

   type DURATION is delta 2.0 ** (-14) range -86400.0 .. 86400.0;
   ...
end STANDARD;
```

Appendix F. IMPLEMENTATION DEPENDENCIES

This section constitutes Appendix F of the Ada LRM for this implementation. Appendix F from the LRM states:

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

The reference manual of each Ada implementation must include an appendix (called Appendix F) that describes all implementation-dependent characteristics. The Appendix F for a given implementation must list in particular:

1. *The form, allowed places, and effect of every implementation-dependent pragma.*
2. *The name and the type of every implementation-dependent attribute.*
3. *The specification of the package SYSTEM (see 13.7).*
4. *The list of all restrictions on representation clauses (see 13.1).*
5. *The conventions used for any implementation-generated name denoting implementation-dependent components (see 13.4).*
6. *The interpretation of expressions that appear in address clauses, including those for interrupts (see 13.5).*
7. *Any restriction on unchecked conversions (see 13.10.2).*
8. *Any implementation-dependent characteristics of the input-output packages (see 14).*

In addition, the present section will describe the following topics:

9. Any implementation-dependent rules for termination of tasks dependent on library packages (see 9.4:13).
10. Other implementation dependencies.
11. Compiler capacity limitations.

F.1 Pragmas

This section describes the form, allowed places, and effect of every implementation-dependent pragma.

F.1.1 Pragmas LIST, PAGE, PRIORITY, ELABORATE

Pragmas LIST, PAGE, PRIORITY and ELABORATE are supported exactly in the form, in the allowed places, and with the effect as described in the LRM.

F.1.2 Pragma SUPPRESS

Form: Pragma SUPPRESS (identifier)
where the identifier is that of the check that can be omitted. This is as specified in LRM B(14), except that suppression of checks for a particular name is not supported. The name clause (ON=>name), if given, causes the entire pragma to be ignored.

The suppression of the following run-time checks, which correspond to situations in which the exceptions CONSTRAINT_ERROR, STORAGE_ERROR, or PROGRAM_ERROR may be raised, are supported:

ACCESS_CHECK
DISCRIMINANT_CHECK
INDEX_CHECK
LENGTH_CHECK
RANGE_CHECK
STORAGE_CHECK
ELABORATION_CHECK

The checks which correspond to situations in which the exception NUMERIC_ERROR may be raised occur in the hardware and therefore pragma SUPPRESS of DIVISION_CHECK and OVERFLOW_CHECK are not supported.

Allowed Places: As specified in LRM B(14) : SUPPRESS.

Effect: Permits the compiler not to emit code in the unit being compiled to perform various checking operations during program execution. The supported checks have the effect of suppressing the specified check as described in the LRM. A pragma SUPPRESS specifying an unsupported check is ignored.

F.1.3 Pragma SUPPRESS_ALL

Form: Pragma SUPPRESS_ALL

Allowed Places: As specified in LRM B(14) for pragma SUPPRESS.

Effect: The implementation-defined pragma SUPPRESS_ALL has the same effect as the specification of a pragma SUPPRESS for each of the supported checks.

F.1.4 Pragma INLINE

Form: Pragma INLINE (SubprogramNameCommaList)

Allowed Places: As specified in LRM B(4) : INLINE

Effect:

Effect: If the subprogram body is available, and the subprogram contains no nested subprograms, packages, or tasks, the code is expanded in-line at every non-recursive call site and is subject to all optimizations.

The stack-frame needed for the elaboration of the inline subprogram will be allocated as a temporary in the frame of the containing code.

Parameters will be passed properly, by value or by reference, as for non-inline subprograms. Register-saving and the like will be suppressed. Parameters may be stored in the local stack-frame or held in registers, as global code generation allows.

Exception-handlers for the INLINE subprogram will be handled as for block-statements.

Use: This pragma is used either when it is believed that the time required for a call to the specified routine will in general be excessive (this for frequently called subprograms) or when the average expected size of expanded code is thought to be comparable to that of a call.

F.1.5 Pragma INTERFACE

Form: Pragma INTERFACE (language_name, subprogram_name)
where the language_name must be an enumeration value of the type SYSTEM.Supported_Language_Name (see Package SYSTEM below).

Allowed Place: As specified in LRM B(5) : INTERFACE.
Unit must include "with" of package SYSTEM.

Effect: Specifies that a subprogram will be provided outside the Ada program library and will be callable with a specified calling interface. Neither an Ada body nor an Ada body_stub may be provided for a subprogram for which INTERFACE has been specified. In the absence of a Pragma

LINK_NAME, the first eight characters of the Ada name are assured to match the entry point name.

Use: Use with a subprogram being provided via another programming language and for which no body will be given in any Ada program. See also the LINK_NAME pragma.

F.1.6 Pragma LINK_NAME

Form: Pragma LINK_NAME (subprogram_name, link_name)

Allowed Places: As specified in LRM B(5) for pragma INTERFACE.

Effect: Associates with subprogram subprogram_name the name link_name

Syntax: The value of link_name must be a character string literal. as its entry point name.

Use: To allow Ada programs, with help from INTERFACE pragma, to reference non-Ada subprograms. Also allows non-Ada programs to call specified Ada subprograms.

F.1.7 Pragma CONTROLLED

Form: Pragma CONTROLLED (AccessTypeName)

Allowed Places: As specified in LRM B(2) : CONTROLLED.

Effect: Means that heap objects are not automatically reclaimed but are explicitly reclaimable by use of unchecked_deallocation.

F.1.8 Pragma PACK

Form: Pragma PACK (type_simple_name)

Allowed Places: As specified in LRM 13.1(12)

Effect: Components are allowed their minimal number of storage units as provided for by their own representation and/or packing.

Floating-point components are aligned on storage-unit boundaries, either 4 bytes or 8 bytes, depending on digits.

Use: Pragma PACK is used to reduce storage size. This can allow records and arrays, in some cases, to be passed by value instead of by reference.

Size reduction usually implies an increased cost of accessing components. The decrease in storage size may be offset by increase in size of accessing code and by slowing of accessing operations.

F.1.9 Pragmas SYSTEM_NAME, STORAGE_UNIT, MEMORY_SIZE, SHARED

These pragmas are not supported and are ignored.

F.1.10 Pragma OPTIMIZE

Pragma OPTIMIZE is ignored; optimization is always enabled.

F.2 Implementation-dependent Attributes

This section describes the name and the type of every implementation-dependent attribute.

There are no implementation defined attributes. These are the values for certain language-defined, implementation-dependent attributes:

Type INTEGER.

INTEGER'SIZE	= 32 -- bits.
INTEGER'FIRST	= - (2**31)
INTEGER'LAST	= (2**31-1)

Type SHORT_FLOAT.

SHORT_FLOAT'SIZE	= 32 -- bits.
SHORT_FLOAT'DIGITS	= 6
SHORT_FLOAT'MANTISSA	= 21
SHORT_FLOAT'EMAX	= 84
SHORT_FLOAT'EPSILON	= 2.0**(-20)
SHORT_FLOAT'SMALL	= 2.0**(-85)
SHORT_FLOAT'LARGE	= 2.0**84
SHORT_FLOAT'MACHINE_ROUNDS	= false
SHORT_FLOAT'MACHINE_RADIX	= 16
SHORT_FLOAT'MACHINE_MANTISSA	= 6
SHORT_FLOAT'MACHINE_EMAX	= 63
SHORT_FLOAT'MACHINE_EMIN	=-64
SHORT_FLOAT'MACHINE_OVERFLOWS	= false
SHORT_FLOAT'SAFE_EMAX	= 252
SHORT_FLOAT'SAFE_SMALL	= 16#0.800000#E-63
SHORT_FLOAT'SAFE_LARGE	= 16#0.FFFFF8#E63

Type FLOAT.

FLOAT'SIZE	= 64 -- bits.
FLOAT'DIGITS	= 15
FLOAT'MANTISSA	= 51
FLOAT'EMAX	= 204
FLOAT'EPSILON	= 2.0**(-50)
FLOAT'SMALL	= 2.0**(-205)
FLOAT'LARGE	= (1.0-2**(-51))*2.0**204
FLOAT'MACHINE_ROUNDS	= false
FLOAT'MACHINE_RADIX	= 16
FLOAT'MACHINE_MANTISSA	= 14
FLOAT'MACHINE_EMAX	= 63
FLOAT'MACHINE_EMIN	= -64
FLOAT'MACHINE_OVERFLOWS	= false

FLOAT'SAFE_EMAX	= 252
FLOAT'SAFE_SMALL	= 16#0.80000000000000#E-63
FLOAT'SAFE_LARGE	= 16#0.FFFFFFFFFFFFE0#E63

Type DURATION.

DURATION'DELTA	= 2.0**(-14) -- seconds
DURATION'FIRST	= - 86,400
DURATION'LAST	= 86,400
DURATION'SMALL	= 2.0**(-14)

Type PRIORITY.

PRIORITY'FIRST	= -127
PRIORITY'LAST	= 127

F.3 Package SYSTEM

```
package SYSTEM is

  --| OVERVIEW
  --|
  --| This is the predefined library package "System", which contains
  --| the definitions of certain configuration-dependent
  --| characteristics.

 type ADDRESS is private;   -- "=", "/=" defined implicitly;
 type NAME is (UTS, MVS, CMS, Prime50, Sperry1100, MIL_STD_1750A);

 SYSTEM_NAME : constant NAME := UTS;

 STORAGE_UNIT : constant := 8;
 MEMORY_SIZE : constant := 2**31;
   -- In storage units

 -- System-Dependent Named Numbers:

 MIN_INT : constant := INTEGER'POS(INTEGER'FIRST);
 MAX_INT : constant := INTEGER'POS(INTEGER'LAST);
 MAX_DIGITS : constant := 15;
 MAX_MANTISSA : constant := 31;
 FINE_DELTA : constant := 2.0**(-31);
 TICK : constant := 1.0E-3;  -- CLOCK function has msec resolution

 -- Other System-Dependent Declarations

 subtype PRIORITY is INTEGER range -127..127;

-------------------------------------------------------------
-- Implementation-dependent additions to package SYSTEM --
-------------------------------------------------------------

 NULL_ADDRESS : constant ADDRESS;
   -- Same bit pattern as "null" access value
   -- This is the value of 'ADDRESS for named numbers.
   -- The 'ADDRESS of any object which occupies storage
   --  is NOT equal to this value.

 ADDRESS_SIZE : constant := 32;
   -- Number of bits in ADDRESS objects, = ADDRESS'SIZE, but static.

 ADDRESS_SEGMENT_SIZE : constant := 2**24;
   -- Number of storage units in address segment

 type ADDRESS_OFFSET is new INTEGER;  -- Used for address arithmetic
 type ADDRESS_SEGMENT is new INTEGER;  -- Always zero on targets with
                                       --    unsegmented address space.

 subtype NORMALIZED_ADDRESS_OFFSET is
   ADDRESS_OFFSET range 0 .. ADDRESS_SEGMENT_SIZE - 1;
```

```
    -- Range of address offsets returned by OFFSET_OF

function "+"(addr : ADDRESS; offset : ADDRESS_OFFSET) return ADDRESS;
function "+"(offset : ADDRESS_OFFSET; addr : ADDRESS) return ADDRESS;

--| EFFECTS
--|
--| Add an offset to an address. May cross segment boundaries on
--| targets where objects may span segments. On other targets,
--| CONSTRAINT_ERROR will be raised when
--| OFFSET_OF(addr) + offset not in NORMALIZED_ADDRESS_OFFSET.

function "-"(left, right : ADDRESS) return ADDRESS_OFFSET;

--| EFFECTS
--|
--| Subtract two addresses, returning an offset. This
--| offset may exceed the segment size on targets where
--| objects may span segments. On other targets,
--| CONSTRAINT_ERROR will be raised if SEGMENT_OF(left) /=
--| SEGMENT_OF(right).

function "-"(addr : ADDRESS; offset : ADDRESS_OFFSET) return
  ADDRESS;

--| EFFECTS
--|
--| Subtract an offset from an address, returning an address.
--| May cross segment boundaries on targets where
--|  objects may span segments.
--| On other targets, CONSTRAINT_ERROR will be raised when
--|  OFFSET_OF(addr) - offset not in NORMALIZED_ADDRESS_OFFSET.

function OFFSET_OF (addr : ADDRESS) return NORMALIZED_ADDRESS_OFFSET;

--| EFFECTS
--|
--| Returns offset part of ADDRESS
--| Always in range 0..seg_size - 1

function SEGMENT_OF (addr : ADDRESS) return ADDRESS_SEGMENT;


--|
--| Returns segment part of address (zero on targets with
--| unsegmented address space).

function MAKE_ADDRESS (offset : ADDRESS_OFFSET;
                       segment : ADDRESS_SEGMENT := 0) return ADDRESS;

--| EFFECTS
--|
--| Builds an address given an offset and a segment.
--| Offsets may be > segment size on targets where objects may
--| span segments, in which case it is equivalent to
```

```
   --| "MAKE_ADDRESS (0, segment) + offset".
   --| On other targets, CONSTRAINT_ERROR will be raised when
   --| offset not in NORMALIZED_ADDRESS_OFFSET.

   type Supported_Language_Name is (   -- Target dependent
      -- The following are "foreign" languages:
      ASSEMBLER,
      FORTRAN_MAIN,
      FORTRAN,
      COBOL_MAIN,
      COBOL,
      JOVIAL_MAIN,
      PL1_MAIN,

      AIE_ASSEMBLER, -- NOT a "foreign" language - uses AIE RTS
      UNSPECIFIED_LANGUAGE_MAIN,
      UNSPECIFIED_LANGUAGE
   );

   -- Most/least accurate built-in integer and float types

   subtype LONGEST_INTEGER is STANDARD.INTEGER;
   subtype SHORTEST_INTEGER is STANDARD.INTEGER;

   subtype LONGEST_FLOAT is STANDARD.FLOAT;
   subtype SHORTEST_FLOAT is STANDARD.SHORT_FLOAT;

private

   type ADDRESS is access INTEGER;
     -- Note: The designated type here (INTEGER) is irrelevant.
     --       ADDRESS is made an access type simply to guarantee it has
     --       the same size as access values, which are single addresses.
     --       Allocators of type ADDRESS are NOT meaningful.

   NULL_ADDRESS : constant ADDRESS := null;

end SYSTEM ;
```

F.4 Representation Clauses

This section describes the list of all restrictions on representation clauses.

"NOTE: An implementation may limit its acceptance of representation clauses to those that can be handled simply by the underlying hardware.... If a program contains a representation clause that is not accepted [by the compiler], then the program is illegal." (LRM 13.1(10)).

There are no restrictions except as follows:

a. Length clauses:

   Size specification must be a multiple of 8 bits. Note that this represents only an upper bound. The compiler may allocate fewer bits for components of packed arrays or records, or when fewer bits are specified by a record representation clause. Size specifications are most useful for record types, in which case the specified size is always used.

   Collection Size specifications--not supported. All collections draw from a single heap.

   Task storage size specifications--not supported. All task stacks automatically extended as needed.

   Specification of SMALL: Specified value must be a power of 2, less than or equal to the DELTA for the type.

b. Enumeration Representation Clauses:
   Fully supported for non-derived types. Not supported for derived types. Note that enumeration types with a representation clause require more code for looping, 'SUCC, 'PRED, 'POS, 'VAL, 'IMAGE and 'VALUE even if the specified representation is contiguous.

c. Record-representation-clause:

   Alignment clause is not supported.

   Only integer, fixed, and enumeration types may have a non-zero first bit number. All other types must start at bit zero of the specified storage unit, and end on a storage unit boundary.

   If the first bit is non-zero, the last bit number must be less than or equal to 31.

   Record components, including those generated implicitly by the compiler, whose locations are not given by the representation-clause, are layed out by the compiler following all the components whose locations are given by the representation-clause. Such components of the invariant part of the record are allocated to follow the user-specified components of the invariant part, and such components in any given variant part are allocated to follow the user-specified components of that variant part.

F.5 Implementation-dependent Components

This section describes the conventions used for any implementation-generated name denoting implementation-dependent components.

There are no implementation-generated names denoting implementation-dependent (record) components, although there are, indeed, such components. Hence, there is no convention (or possibility) of naming them and, therefore, no way to offer a representation clause for a record containing such components.

NOTE: Records containing dynamic-sized components will contain (generally) unnamed offset components which will "point" to the dynamic-sized components stored later in the record. There is no way to specify the representation of such components.

F.6 Address Clauses

This section describes the interpretation of expressions that appear in address clauses, including those for interrupts.

For an object--fully supported; address expression may be any run-time expression of type System.Address. The object address clause is interpreted to mean that the object is "already" at that address. No additional space is allocated for the object.

Typically this means that the address specified is an expression involving the address of some pre-existing object, for instance, a buffer, and the address clause is being used to gain access to some part of this object, perhaps a message embedded within the buffer.

For a subprogram--supported only subprograms with a pragma Interface. The subprogram address clause is interpreted to mean that the subprogram is "already" at that address. This provides a means to call a subprogram given only its address, passed as a parameter or stored in a data structure.

For a package, task, or task entry--not supported.

F.7 Unchecked Conversions

This section describes any restrictions on unchecked conversions.

The source and target subtypes must occupy the same number of bits. Note that access to unconstrained arrays actually point at a "dope vector" followed by the array, so unchecked conversion of an address to or from such an access value will generally produce unpredictable results. Given the address of an array, it is more meaningful to convert it to an access to a constrained array subtype with the appropriate bounds.

F.8 Input-Output

This section describes implementation-dependent characteristics of the input-output packages.

(a) Declaration of type Direct_IO.Count? [14.2.5]
    0..Integer'last;

(b) Effect of input/output for access types?
    Not meaningful if read by different program invocations

(c) Disposition of unclosed IN_FILE files at program termination? [14.1(7)]
    Files are closed.

(d) Disposition of unclosed OUT_FILE files at program termination? [14.1(7)]
    Files are closed.

(e) Disposition of unclosed INOUT_FILE files at program termination? [14.1(7)]
    Files are closed.

(f) Form of, and restrictions on, file names? [14.1(1)]
    MVS filenames.

(g) Possible uses of FORM parameter in I/O subprograms? [14.1(1)]
    See Appendix A.

(h) Where are I/O exceptions raised beyond what is described in Chapter 14? [14.1(11)]
    None raised.

(i) Are alternate specifications (such as abbreviations) allowed for file names? If so, what is the form of these alternatives? [14.2.1(21)]
    No.

(j) When is DATA_ERROR *not* raised for sequential or direct input of an inappropriate ELEMENT_TYPE? [14.2.2(4), 14.2.4(4)]
    Unresolved issue.

(k) What are the standard input and standard output files? [14.3(5)]
    MVS standard input (SYSIN) and output (SYSPRINT).

(l) What are the forms of line terminators and page terminators? [14.3(7)]
    Line terminator is ASCII.LF (line feed).
    Page terminator is ASCII.FF (form feed).

(m) Value of Text_IO.Count'last? [14.3(8)]
    integer'last.

(n) Value of Text_IO.Field'last? [14.3.7(2)]
    integer'last.

(o) Effect of instantiating ENUMERATION_IO for an integer type? [14.3.9(15)]
    The instantiated Put will work properly, but the instantiated Get will raise Data_Error.

(p) Restrictions on types that can be instantiated for input/output?

No direct I/O on unconstrained types.

(q) Specification of package Low_Level_IO? [14.6]

Low_Level_IO is not provided.

F.9 Tasking

This section describes implementation-dependent characteristics of the tasking run-time packages.

The scheduler of the MVS/Ada run-time tasking system runs tasks of equal priority in the order that they became eligible to run and allows them to run until blocked or until interrupted by the eligibility of a task of higher priority.

A task whose priority is higher than the task currently running may be made eligible to run by an interrupt or timer runout. Such an event will cause the currently running task to be immediately blocked so that the higher priority task may run.

Even though a main program completes and terminates (its dependent tasks, if any, having terminated), the execution of the program as a whole continues until each task dependent upon a library unit package has either terminated or reached an open terminate alternative. See LRM 9.4(13).

F.10 Other Matters

This section describes other implementation-dependent characteristics of the system.

a. Package Machine_Code
   This package is not provided.

b. Order of compilation of generic bodies and subunits (LRM 10.3:9):
   Body and subunits of generic must be in the same compilation as the specification if instantiations precede them (see AI-00257/02).

F.11 Compiler Limitations

(a) Maximum length of variable of type STRING?
  2**31-1 characters (Address space size).

(b) Maximum length of a record?
  2**31-1 bytes (Address space size).

(c) Maximum length of an array?
  2**31-1 bytes. (Address space size).

(d) Maximum size of Ada Program Library?
  Limited by available disk storage.

(e) Maximum length of source line?
  255 characters.

(f) Maximum number of "use" scopes?
  Limit is 50, set arbitrarily by SEMANTICS as maximum number of distinct packages actively "used."

(g) Maximum number of co-existing tasks?
  Limited by available memory space.

(h) Maximum number of tasks initially allocated?
  One if any tasking constructs exist; zero if no tasking constructs exist.

(i) Maximum length of identifier?
  255 characters.

(j) Maximum number of nested loops?
  24 nested loops.

(k) Maximum number of identifiers in symbol table?
  Limited only by address space.

(l) Package nesting limit?
  Limited by parse-stack nesting limit, 200.

(m) Subprogram nesting limit?
  Limited by parse-stack nesting limit, 200.

(n) Logical expression nesting limit?
  Limited by parse-stack nesting limit, 200.

(o) Minimum storage requirement for compiler's operation?
  5 megabytes.

(p) Can compiler provide listing files in variable block format?
  Yes.

(q) Can compiler provide object in fixed block 80 format?
  Yes.

(r) What storage is required, beyond the storage-units for an object, for allocators in heaps; in collections?

Approximately 8 bytes per element plus sixteen bytes for collection as a whole.