REPORT DOCUMENTATION PAGE

Form Approved
OPM No. 0704-0188

2

AD-A240 784

PORT DATE	3 REPORT TYPE AND DATES COVERED
	Final: 30 Jul 1991 to 01 Jun 1993

4 TITLE AND SUBTITLE
Ada Compiler Validation Summary Report:U.S. NAVY, Ada/L, Version 4.0 (/OPTIMIZE), VAX 8550(Host) to AN/UYK-43 (Single CPU)(Bare Board)(Target), 910626S1.11172

5. FUNDING NUMBERS

6. AUTHOR(S)
National Institute of Standards and Technology
Gaithersburg, MD
USA

7 PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
National Institute of Standards and Technology
National Computer Systems Laboratory
Bldg. 255, Rm A266
Gaithersburg, MD 20899 USA

8. PERFORMING ORGANIZATION REPORT NUMBER
NIST90USN510_5_1.11

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)
U.S. NAVY, Ada/L, Version 4.0 (OPTIMIZE) Gaithersburg, MD, VAX 8550, running VAX/VMS Version 5.3 (Host) to AN/UYK-43 (Single CPU)(Bare Board)(Target), ACVC 1.11.

91-11054

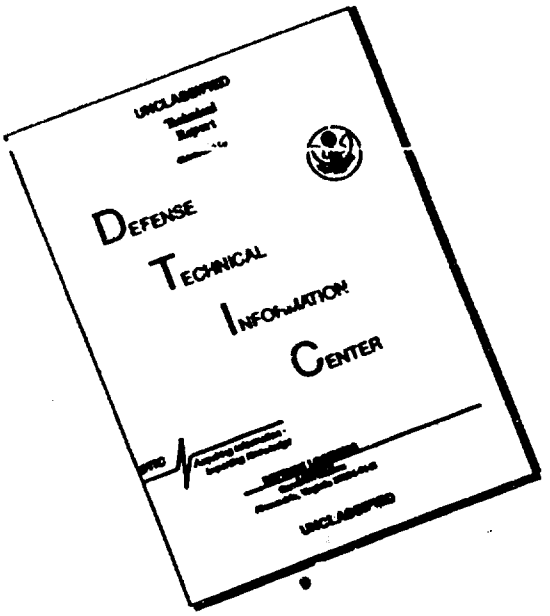
14 SUBJECT TERMS
Ada programming language, Ada Compiler Val. Summary Report, Ada Compiler Val. Capability, Val. Testing, Ada Val. Office, Ada Val. Facility, ANSI/MIL-STD-1815A, AJPO.

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	

DISCLAIMER NOTICE

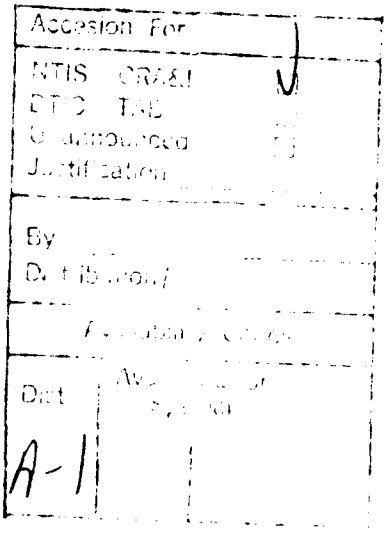


THIS DOCUMENT IS BEST QUALITY AVAILABLE. THE COPY FURNISHED TO DTIC CONTAINED A SIGNIFICANT NUMBER OF PAGES WHICH DO NOT REPRODUCE LEGIBLY.

AVF Control Number: NIST90USN510_5_1.11
DATE COMPLETED
BEFORE ON-SITE: 1991-04-05
AFTER ON-SITE: 1991-06-26
REVISIONS: 1991-07-30

Ada COMPILER
VALIDATION SUMMARY REPORT:
Certificate Number: 910626S1.11172
U.S. NAVY
Ada/L, Version 4.0 (/OPTIMIZE)
VAX 8550 => AN/UYK-43 (Single CPU) (Bare Board)

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



AVF Control Number: NIST90USN510_5_1.11

Certificate Information

The following Ada implementation was tested and determined to pass ACVC 1.11. Testing was completed on 1991-06-26.

Compiler Name and Version: Ada/L, Version 4.0 (/OPTIMIZE)

Host Computer System: VAX 8550, running VAX/VMS Version 5.3

Target Computer System: AN/UYK-43 (Single CPU) (Bare Board)

A more detailed description of this Ada implementation is found in section 3.1 of this report.

As a result of this validation effort, Validation Certificate 910626S1.11172 is awarded to U.S. NAVY. This certificate expires on 01 March 1993.

This report has been reviewed and is approved.

Ada Validation Facility
Dr. David K. Jefferson
Chief, Information Systems
Engineering Division (ISED)

Ada Validation Facility
Mr. L. Arnold Johnson
Manager, Software Standards
Validation Group

Computer Systems Laboratory (CLS)
National Institute of Standards and Technology
Building 225, Room A266
Gaithersburg, MD 20899

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

The following declaration of conformance was supplied by the customer.

DECLARATION OF CONFORMANCE

Customer: U.S. NAVY

Certificate Awardee: U.S. NAVY

Ada Validation Facility: National Institute of Standards and Technology
Computer Systems Laboratory (CSL)
Software Validation Group
Building 225, Room A266
Gaithersburg, Maryland 20899

ACVC Version: 1.11

Ada Implementation:

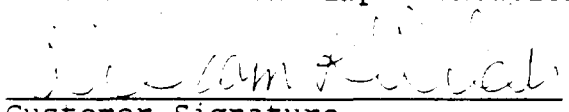
Compiler Name and Version: Ada/L, Version 4.0 (/OPTIMIZE)

Host Computer System: VAX 8550, running VAX/VMS Version 5.3

Target Computer System: AN/UYK-43 (Single CPU) (Bare Board)

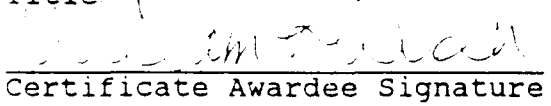
Declaration:

I the undersigned, declare that I have no knowledge of deliberate deviations from the Ada Language Standard ANSI/MIL-STD-1815A ISO 8652-1987 in the implementation listed above.



Customer Signature
Company U.S. Navy
Title

6/21/91
Date



Certificate Awardee Signature
Company U.S. Navy
Title

6/21/91
Date

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 some inapplicable tests (see Section 3.2 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, Validation consisting of the test suite, the support programs, the ACVC Capability user's guide and the template for the validation summary (ACVC) report.
Ada Implementation	An Ada compiler with its host computer system and its target computer 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.
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

Some tests are withdrawn by the AVO from the ACVC because they do not conform to the Ada Standard. The following 94 tests had been withdrawn by the Ada Validation Organization (AVO) at the time of validation testing. 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 91-05-03.

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	B83026B	C83026A	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. The inapplicability criteria for some tests are explained in documents issued by ISO and the AJPO known as Ada Issues and commonly referenced in the format AI-dddd. For this implementation, the following tests were inapplicable for the reasons indicated; references to Ada Issues 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)

CD4113H..K (4 tests) use a line length greater than MAX_IN_LEN.

C35713B, C45423B, B86001T, and C86006H check for the predefined type SHORT_FLOAT; for this implementation, there is no such type.

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				

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.

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.

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, there is no such type.

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

CD1009C checks whether a length clause can specify a non-default size for a floating-point type; this implementation does not support such sizes.

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.

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 are not applicable because the given file operations are supported for the given combination of mode and file access method.

Test	File Operation	Mode	File Access Method
CE2102E	CREATE	OUT_FILE	SEQUENTIAL_IO
CE2102F	CREATE	INOUT_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
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

The tests listed in the following table are not applicable because the given file operations are not supported for the given combination of mode and file access method.

Test	File Operation	Mode	File Access Method
CE2105A	CREATE	IN_FILE	SEQUENTIAL_IO
CE2105B	CREATE	IN_FILE	DIRECT_IO
CE3109A	CREATE	IN_FILE	TEXT_IO

The following 19 tests check operations on sequential, direct, and text files when multiple internal files are associated with the same external file; USE_ERROR is raised when this association is attempted.

CE2107A..H	CE2107L	CE2110B	CE2110D	CE2111D
CE2111H	CE3111A..B	CE3111D..E	CE3114B	CE3115A

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.

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 44 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.

B22003A	B22004A	B23004A	B24005A	B24005B	B28003A
B33201C	B33202C	B33203C	B33301B	B37106A	B37301I
B38003A	B38003B	B38009A	B38009B	B44001A	B44004A
B54A01L	B55A01A	B61005A	B85008G	B85008H	B95063A
B97103E	BB1006B	BC1102A	BC1109A	BC1109B	BC1109C
BC1109D	BC1201F	BC1201G	BC1201H	BC1201I	BC1201J
BC1201L	BC3013A	BE2210A	BE2413A		

C83030C and C86007A were graded passed by Test Modification as directed by the AVO. These tests were modified by inserting "PRAGMA ELABORATE (REPORT);" before the package declarations at lines 13 and 11, respectively. Without the pragma, the packages may be elaborated prior to package Report's body, and thus the packages' calls to function REPORT.IDENT_INT at lines 14 and 13, respectively, will raise PROGRAM_ERROR.

C34005P and C34005S were graded passed by Test Modification as directed by the AVO. These tests contain expressions of the form "I - X'FIRST + Y'FIRST", where X and Y are of an array type with a lower bound of INTEGER'FIRST; this implementation recognizes that "X'FIRST + Y'FIRST" is a loop invariant and so evaluates this part of the expression separately, which raises NUMERIC_ERROR. These tests were modified by inserting parens to force a different order of evaluation (i.p., to force the subtraction to be evaluated first) at lines 187 and 262/263, respectively; those modified lines are:

[C34005P, line 187]

```
IF NOT EQUAL (X (I), Y ((I - X'FIRST) + Y'FIRST)) THEN
```

[C34005S, lines 261..4 (only 262 & 263 were modified)]

```
IF NOT EQUAL (X (I, J),
              Y ((I - X'FIRST) + Y'FIRST,
                 (J - X'FIRST(2)) +
                     Y'FIRST(2))) THEN
```

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:

Mr. Christopher T. Geyer
Fleet Combat Directions Systems Support Activity
Code 81, Room 301D
200 Catalina Blvd.
San Diego, California 92147
619-553-9447

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

NOT APPLICABLE FOR THIS IMPLEMENTATION

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.

a) Total Number of Applicable Tests	3772
b) Total Number of Withdrawn Tests	94
c) Processed Inapplicable Tests	304
d) Non-Processed I/O Tests	0
e) Non-Processed Floating-Point Precision Tests	0

f) Total Number of Inapplicable Tests 304 (c+d+e)
g) Total Number of Tests for ACVC 1.11 4170 (a+b+f)

When this implementation 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 304 tests were inapplicable to this implementation. All inapplicable tests were processed during validation testing. 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 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.

The tests were compiled and linked on the host computer system and executed on the target computer system.

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:

FOR /OPTIMIZE the options were:

/SUMMARY /OPTIMIZE /SOURCE /OUT=<filename>

The options invoked by default for validation testing during this test were:

FOR /OPTIMIZE the options were:

NO_MACHINE_CODE NO_ATTRIBUTE NO_CROSS_REFERENCE
NO_DIAGNOSTICS NO_NOTES PRIVATE LIST CONTAINER_GENERATION

CODE_ON_WARNING NO_MEASURE DEBUG CHECKS NO_EXECUTIVE
NO_RTE_ONLY TRACE_BACK /NO_EMR

Test output, compiler and linker listings, and job logs were captured on magnetic tape and archived at the AVF. Selected 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	120
$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 contains the values for the remaining macro parameters.

Macro Parameter	Macro Value
$ACC_SIZE	32
$ALIGNMENT	4
$COUNT_LAST	2_147_483_647
$DEFAULT_MEM_SIZE	1_048_576
$DEFAULT_STOR_UNIT	32
$DEFAULT_SYS_NAME	ANUYK43
$DELTA_DOC	2#0.0000_0000_0000_0000_0000_0000_0000_001#
$ENTRY_ADDRESS	SYSTEM.CLASS_I_UNHANDLED_ADDRESS
$ENTRY_ADDRESS1	SYSTEM.CLASS_II_UNHANDLED_ADDRESS
$ENTRY_ADDRESS2	SYSTEM.CLASS_III_UNHANDLED_ADDRESS
$FIELD_LAST	2_147_483_647
$FILE_TERMINATOR	' '
$FIXED_NAME	NO_SUCH_FIXED_TYPE
$FLOAT_NAME	NO_SUCH_FLOAT_TYPE
$FORM_STRING	""
$FORM_STRING2	"CANNOT_RESTRICT_FILE_CAPACITY"
$GREATER_THAN_DURATION	131071.5
$GREATER_THAN_DURATION_BASE_LAST	131_073.0
$GREATER_THAN_FLOAT_BASE_LAST	7.5E+75
$GREATER_THAN_FLOAT_SAFE_LARGE	7.5E+75

```
$GREATER_THAN_SHORT_FLOAT_SAFE_LARGE     0.0E0

$HIGH_PRIORITY                  15

$ILLEGAL_EXTERNAL_FILE_NAME1    \NODIRECTORY\FILENAME\

$ILLEGAL_EXTERNAL_FILE_NAME2    THIS-FILE-NAME-IS-TOO-LONG-FOR-
                                MY-SYSTEM

$INAPPROPRIATE_LINE_LENGTH      -1

$INAPPROPRIATE_PAGE_LENGTH      -1

$INCLUDE_PRAGMA1                PRAGMA INCLUDE ("A28006D1.TST")

$INCLUDE_PRAGMA2                PRAGMA INCLUDE ("B28006F1.TST")

$INTEGER_FIRST                  -2147483647

$INTEGER_LAST                    2147483647

$INTEGER_LAST_PLUS_1             2147483648

$INTERFACE_LANGUAGE             MACRO_NORMAL

$LESS_THAN_DURATION             -131071.5

$LESS_THAN_DURATION_BASE_FIRST      -131_073.0

$LINE_TERMINATOR                ASCII.LF

$LOW_PRIORITY                   0

$MACHINE_CODE_STATEMENT         formati'(f_lb,0,0,0,0,0,0)

$MACHINE_CODE_TYPE              formati

$MANTISSA_DOC                   31

$MAX_DIGITS                     15

$MAX_INT                        9223372036854775807

$MAX_INT_PLUS_1                 9223372036854775808

$MIN_INT                        -9223372036854775807

$NAME                           NO_SUCH_TYPE_AVAILABLE

$NAME_LIST                      ANUYK43
```

$NAME_SPECIFICATION1	X2120A
$NAME_SPECIFICATION2	X2120B
$NAME_SPECIFICATION3	X3119A
$NEG_BASED_INT	16#FFFFFFFFFFFFFFFD#
$NEW_MEM_SIZE	1_048_576
$NEW_STOR_UNIT	32
$NEW_SYS_NAME	ANUYK43
$PAGE_TERMINATOR	ASCII.FF
$RECORD_DEFINITION	record f:i6_bit; a:i3_bit; k:i3_bit; b:i3_bit; i:i1_bit; s:i3_bit; y:i13_bit; end record;
$RECORD_NAME	formatii
$TASK_SIZE	32
$TASK_STORAGE_SIZE	1024
$TICK	0.000048828125
$VARIABLE_ADDRESS	16#0020#
$VARIABLE_ADDRESS1	16#0021#
$VARIABLE_ADDRESS2	16#0023#
$YOUR_PRAGMA	EXECUTIVE

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.

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.

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 -2_147_483_647 .. 2_147_483_647;
 type LONG_INTEGER is range
          -9_223_372_036_854_775_807 .. 9_223_372_036_854_775_807;
 type FLOAT is digits 6 range
     -(16#0.FF_FFF8#E63) .. (16#0.FF_FFF8#E63);
 type LONG_FLOAT is digits 15 range
     -(16#0.FF_FFFF_FFFF_FFE0#E63) .. (16#0.FF_FFFF_FFFF_FFE0#E63);
 type DURATION is delta 2.0 ** (-14) range
       -131_071.0 .. 131_071.0;

end STANDARD;
```

Appendix F

The Ada Language for the AN/UYK-43 Target

The source language accepted by the compiler is Ada, as described in the Military Standard, Ada Programming Language, ANSI/MIL-STD-1815A-1983, 17 February 1983 ("Ada Language Reference Manual").

The Ada definition permits certain implementation dependencies. Each Ada implementation is required to supply a complete description of its dependencies, to be thought of as Appendix F to the Ada Language Reference Manual. This section is that description for the AN/UYK-43 target.

F.1 Options

There are several compiler options provided by all ALS/N Compilers that directly affect the pragmas defined in the Ada Language Reference Manual. These compiler options currently include the CHECKS and OPTIMIZE options which affect the SUPPRESS and OPTIMIZE pragmas, respectively. A complete list of ALS/N Compiler options can be found in Section 9.

The CHECKS option enables all run-time error checking for the source file being compiled, which can contain one or more compilation units. This allows the SUPPRESS pragma to be used in suppressing the run-time checks discussed in the Ada Language Reference Manual, but note that the SUPPRESS pragma(s) must be applied to each compilation unit. The NO_CHECKS option disables all run-time error checking for all compilation units within the source file and is equivalent to SUPPRESSing all run-time checks within every compilation unit.

The OPTIMIZE option enables all compile-time optimizations for the source file being compiled, which can contain one or more compilation units. This allows the OPTIMIZE pragma to request either TIME-oriented or SPACE-oriented optimizations be performed, but note that the OPTIMIZE pragma must be applied to each compilation unit. If the OPTIMIZE pragma is not present, the ALS/N Compiler's Global Optimizer tends to optimize for TIME over SPACE. The NO_OPTIMIZE option disables all compile-time optimizations for all compilation units within the source file regardless of whether or not the OPTIMIZE pragma is present.

In addition to those compiler options normally provided by the ALS/N Common Ada Baseline compilers, the Ada/L compiler also implements the EXECUTIVE, DEBUG, and MEASURE options.

The EXECUTIVE compiler option enables processing of PRAGMA EXECUTIVE and allows WITH of units compiled with the RTE_ONLY option. If NO_EXECUTIVE is specified on the command line, the pragma will be ignored and will have no effect on the generated code.

The DEBUG compiler option enables processing of PRAGMA DEBUG to provide debugging support. If NO_DEBUG is specified, the DEBUG pragmas shall have no effect. Program units containing DEBUG pragmas and compiled with the DEBUG compiler option may be linked with program units containing DEBUG pragmas and compiled with the NO_DEBUG option; only those program units compiled with the DEBUG option shall have additional DEBUG support.

The MEASURE compiler option enables run-time calls to Run-Time Performance Measurement Aids (RTAids) to record the entrance into all subprograms whose bodies are in the compilation. Program units compiled with the /MEASURE option may be linked with program units not compiled with the /MEASURE option; at run-time, only those subprograms in program units compiled with the /MEASURE option shall have this additional MEASURE support.

F.2 Pragmas

Both implementation-defined and Ada language-defined pragmas are provided by all ALS/N compilers. These paragraphs describe the pragmas recognized and processed by the Ada/L compiler. The syntax defined in Section 2.8 of the Ada Language Reference Manual allows a pragma as the only element in a compilation unit, before a compilation unit, at defined places within a compilation unit, or following a compilation unit. Ada/L associates pragmas with compilation units as follows:

a. If a pragma appears before any compilation unit in a compilation, it will affect all following compilation units, as specified below and in section 10.1 of the Ada Language Reference Manual.

b. If a pragma appears inside a compilation unit, it will be associated with that compilation unit, and with the listings associated with that compilation unit, as described in the Ada Language Reference Manual, or below.

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

The pragmas MEMORY_SIZE, STORAGE_UNIT, and SYSTEM_NAME are described in Section 13.7 of the Ada Language Reference Manual. They may appear only at the start of the first compilation when creating a program library. In the ALS/N, however, since program libraries are created by the Program Library Manager and not by the compiler, the use of these pragmas is obviated. If they appear anywhere, a diagnostic of severity level WARNING is generated.

F.2.1 Language-Defined Pragmas

The following notes specify implementation-specific changes to those pragmas described in Appendix B of the Ada Language Reference Manual. Unmentioned pragmas are implemented as defined in the Ada Language Reference Manual.

pragma INLINE (arg {,arg});

The arguments designate subprograms. There are three instances in which the INLINE pragma is ignored. Each of these cases produces a warning message which states that the INLINE did not occur.

a. If the compilation unit containing the INLINEd subprogram depends on the compilation unit of its caller, a routine call is made instead.

b. If the INLINEd subprogram's compilation unit depends on the compilation unit of its caller (a routine call is made instead).

c. If an immediately recursive subprogram call is made within the body of the INLINEd subprogram (the pragma INLINE is ignored entirely).

pragma INTERFACE (language_name, subprogram_name);

The language_name specifies the language and type of interface to be used in calls used to the externally supplied subprogram specified by subprogram_name. The allowed values for language_name are MACRO_NORMAL and MACRO_QUICK. MACRO_NORMAL indicates that parameters will be passed on the stack and the calling conventions used for normal Ada subprogram calls will apply. MACRO_QUICK is used in RTLIB routines to indicate that parameters are passed in registers. See Section 7 "Parameter Passing" for details on the space required to pass various types of parameters.

You must ensure that an assembly-language body container will exist in the program library before linking.

pragma OPTIMIZE (arg);

The argument is either TIME or SPACE. If TIME is specified, the optimizer concentrates on optimizing code execution time. If SPACE is specified, the optimizer concentrates on optimizing code size. The default is If the OPTIMIZE option is enabled and pragma OPTIMIZE is not present, global optimization is still performed with the default argument, SPACE. Program units containing OPTIMIZE pragmas and compiled with the OPTIMIZE option may be linked with program units containing OPTIMIZE pragmas and compiled with the NO_OPTIMIZE option; but only those program units compiled with the OPTIMIZE option will have global optimization support.

pragma PRIORITY (arg);

The argument is an integer static expression in the range 0..15, where 0 is the lowest use-specifiable task priority and 15 is the highest. If the value of the argument is out of range, the pragma will have no effect other than to generate a WARNING diagnostic. A value of zero will be used if priority is not defined. The pragma will have no effect when not specified in a task (type) specification or the outermost declarative part of a subprogram. If the pragma appears in the declarative part of a subprogram, it will have no effect unless that subprogram is designated as the main subprogram at link time.

pragma SUPPRESS (arg {,arg});

This pragma is unchanged with the following exceptions:

Suppression of OVERFLOW_CHECK applies only to integer operations; and PRAGMA SUPPRESS has effect only within the compilation unit in which it appears, except that suppression of ELABORATION_CHECK applied at the declaration of a subprogram or task unit applies to all calls or activations.

F.2.2 Implementation-Defined Pragmas

This paragraph describes the use and meaning of those pragmas recognized by Ada/L which are not specified in Appendix B of the Ada Language Reference Manual.

pragma DEBUG;

> This pragma enables the inclusion of full symbolic information and support for the Embedded Target Debugger. The DEBUG PRAGMA is enabled by the /DEBUG command line option and has no effect if this option is not provided. This pragma must appear within a compilation unit, before the first declaration or statement.

pragma EXECUTIVE [(arg)];

> This pragma allows you to specify that a compilation unit is to run in the executive state of the machine and/or utilize privileged instructions. The pragma has no effect if the Compiler option NO_EXECUTIVE is enabled, either explicitly or by default.
>
> If PRAGMA EXECUTIVE is specified without an argument, executive state is in effect for the compilation unit and the code generator does not generate privileged instructions for the compilation unit. If PRAGMA EXECUTIVE (INHERIT) is specified, a subprogram in the compilation unit inherits the state of its caller and the code generator does not generate privileged instructions for the compilation unit. If PRAGMA EXECUTIVE (PRIVILEGED) is specified, the executive state is in effect and the code generator may generate privileged instructions for the compilation unit. Currently, the Ada/L compiler does not generate such instructions. In the absence of PRAGMA EXECUTIVE, the compilation unit executes in task state and the code generator does not generate privileged instructions. If PRAGMA EXECUTIVE (INTERRUPT_CMR) is specified, the Ada/L compiler generates code which uses executive state registers instead of task state registers (i.e. SCI instead of SCT).
>
> PRAGMA EXECUTIVE is applied once per compilation unit, so its scope is the entire compilation unit. PRAGMA EXECUTIVE may appear between the context clause and the outermost unit. If there is no context clause, PRAGMA EXECUTIVE must appear within that unit before the first declaration or statement. The placement of the pragma before the context clause has no effect on any or all following compilation units. If PRAGMA EXECUTIVE appears in the specification of a compilation unit, it

must also appear in the body of that unit, and vice versa. If the pragma appears in a specification but is absent from the body, you are warned and the pragma is effective. If the pragma appears in the body of a compilation unit, but is absent from the corresponding specification, you are warned and the pragma has no effect. PRAGMA EXECUTIVE does not propagate to subunits. If a subunit is compiled without PRAGMA EXECUTIVE and the parent of the subunit is compiled with PRAGMA EXECUTIVE, you are warned and PRAGMA EXECUTIVE has no effect on the subunit.

pragma FAST_INTERRUPT_ENTRY (entry_name, IMMEDIATE);

This pragma provides for situations of high interrupt rates with simple processing per interrupt, (such as adding data to a buffer), and where complex processing occurs only after large numbers of these interrupts (such as when the buffer is full). This allows for lower overhead and faster response capability by restricting you to disciplines that are commensurate with limitations normally found in machine level interrupt service routine processing.

pragma MEASURE (extraction_set, [arg {,arg}]);

This pragma enables one or more performance measurement features. Pragma MEASURE specifies a user-defined extraction set for the Run-Time Performance Measurement Aids and Embedded Target Profiler. The user-defined extraction set consists of all occurences pragma MEASURE throughout the program. Extraction_set is a numeric literal, which is an index into a user-supplied table. Arg is a variable or a list of variables whose values are reported at this point in the execution. These values describe the nature (TYPE) of the values collected to an independent data reduction program. Pragma MEASURE is enabled by the /MEASURE command line option and has no effect if this option is not provided. This pragma should be applied to a package body rather than a package specification.

pragma STATIC (INTERRUPT_HANDLER_TASK);

The pragma STATIC is only allowed immediately after the declaration of a task body containing an immediate interrupt entry. The argument is INTERRUPT_HANDLER_TASK. The effect of this pragma will be to allow generation of nonreentrant and nonrecursive code in a compilation unit, and to allow static allocation of all data in a compilation unit. This pragma shall be used to allow for procedures within immediate (fast) interrupt entries. The effect will be

for the compiler to generate nonreentrant code for the affected procedure bodies. If a STATIC procedure is called recursively, the program is erroneous.

pragma TITLE (arg);

This is a listing control pragma. It takes a single argument of type string. The string specified will appear on the second line of each page of the source listing produced for the compilation unit within which it appears. The pragma should be the first lexical unit to appear within a compilation unit (excluding comments). If it is not, a warning message is issued.

pragma TRIVIAL_ENTRY (NAME: entry_simple_name);

This pragma is only allowed within a task specification after an entry declaration and identifies a Trivial_Entry to the system. A trivial entry represents a synchronization point, contained in a normal Ada task, for rendezvous with a fast interrupt entry body. The body of a trivial entry must be null.

pragma UNMAPPED (arg {,arg});

The effect of this pragma is for unmapped (i.e., not consistently mapped within the virtual space) allocation of data in a compilation unit. The arguments of this pragma are access types to be unmapped. If a program tries to allocate more UNMAPPED space than is available in the physical configuration, STORAGE_ERROR will be raised at run-time. PRAGMA UNMAPPED must appear in the same declarative region as the type and after the type declaration.

F.2.3 Scope of Pragmas

The scope for each pragma previously described as differing from the Ada Language Reference Manual is given below:

DEBUG — Applies to the compilation unit in which the pragma appears.

EXECUTIVE — Applies to the compilation unit in which the pragma appears, i.e., to all subprograms and tasks within the unit. Elaboration code is not affected. The pragma is not propagated from specifications to bodies, or from bodies to subunits. The pragma must appear consistently in the specification, body, and subunits associated with a library unit.

FAST_INTERRUPT_ENTRY — Applies to the compilation unit in which the pragma appears.

INLINE — Applies only to subprogram names in its arguments. If the argument is an overloaded subprogram name, the INLINE pragma applies to all definitions of that subprogram name which appear in the same declarative part as the INLINE pragma.

INTERFACE — Applies to all invocations of the named imported subprogram.

MEASURE — No scope, but a WARNING diagnostic is generated.

MEMORY_SIZE — No scope, but a WARNING diagnostic is generated.

OPTIMIZE — Applies to the entire compilation unit in which the pragma appears.

PRIORITY — Applies to the task specification in which it appears, or to the environment task if it appears in the main subprogram.

STATIC — Applies to the compilation unit in which the pragma appears.

STORAGE_UNIT — No scope, but a WARNING diagnostic is generated.

SUPPRESS — Applies to the block or body that contains the declarative part in which the pragma appears.

SYSTEM_NAME	No scope, but a WARNING diagnostic is generated.
TITLE	The compilation unit within which the pragma occurs.
TRIVIAL_ENTRY	Applies to the compilation unit in which the pragma appears.
UNMAPPED	Applies to all objects of the access type named as arguments.

F.3 Attributes

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

T'MACHINE_EMAX	is 63.
T'MACHINE_EMIN	is -64.
T'MACHINE_MANTISSA	is 6.
T'MACHINE_OVERFLOWS	is TRUE.
T'MACHINE_RADIX	is 16.
T'MACHINE_ROUNDS	is FALSE.

F.4 Predefined Language Environment

The predefined Ada language environment consists of the packages STANDARD and SYSTEM, which are described below.

F.4.1 Package STANDARD

The package STANDARD contains the following definitions in addition to those specified in Appendix C of the Ada Language Reference Manual.

```
TYPE boolean IS (false, true);
FOR  boolean'SIZE USE 1;

TYPE integer IS RANGE -2_147_483_647 .. 2_147_483_647;
TYPE long_integer IS RANGE
   -9_223_372_036_854_775_807 .. 9_223_372_036_854_775_807;

TYPE float IS DIGITS 6 RANGE
               -(16#0.FF_FFF8#E63) .. (16#0.FF_FFF8#E63);
TYPE long_float IS DIGITS 15 RANGE
               -(16#0.FF_FFFF_FFFF_FFE0#E63) ..
                (16#0.FF_FFFF_FFFF_FFE0#E63);

SUBTYPE natural  IS integer RANGE 0 .. integer'LAST;
SUBTYPE positive IS integer RANGE 1 .. integer'LAST;
SUBTYPE long_natural  IS long_integer
                      RANGE 0 .. long_integer'LAST;
SUBTYPE long_positive IS long_integer
                      RANGE 1 .. long_integer'LAST;

FOR character'SIZE USE 8;
TYPE string IS ARRAY (positive RANGE <>) OF character;
PRAGMA PACK(string);

TYPE duration IS DELTA 2.0 ** (-14)
                 RANGE -131_071.0 .. 131_071.0;

-- The predefined exceptions:

constraint_error : exception;
numeric_error    : exception;
program_error    : exception;
storage_error    : exception;
tasking_error    : exception;
```

F.4.2 Package SYSTEM

The package SYSTEM for Ada/L is as follows:

```
TYPE name     IS (anuyk43);
system_name   : CONSTANT system.name := system.anuyk43;
storage_unit  : CONSTANT := 32;
memory_size   : CONSTANT := 1_048_576;
TYPE address IS RANGE 0..system.memory_size - 1;

-- System Dependent Named Numbers

min_int       : CONSTANT := -((2**63)-1);
max_int       : CONSTANT :=   (2**63)-1;
max_digits    : CONSTANT :=    15;
max_mantissa  : CONSTANT :=    31;
fine_delta    : CONSTANT
   := 2#0.0000_0000_0000_0000_0000_0000_0000_001#;
tick          : CONSTANT :=    4.8828125e-05;
   -- 1/20480 seconds is the basic clock period.
null_addr     : CONSTANT address := 0;

-- Other System Dependent Declarations

SUBTYPE smaller_integer IS
   integer RANGE (integer'FIRST/64)..(integer'LAST/64);

SUBTYPE priority IS integer RANGE 0..15;
TYPE entry_kind is (normal, immediate);

physical_memory_size : CONSTANT := 2**31;
TYPE physical_address IS
   RANGE 0..system.physical_memory_size - 1 ;

null_phys_addr : CONSTANT physical_address := 0;
TYPE word IS NEW INTEGER;

--
-- Address clause (interrupt) addresses
--
Class_I_Unhandled_address              : CONSTANT
                                       address := 16#0800#;
Class_II_Unhandled_address             : CONSTANT
                                       address := 16#1800#;
CP_Operand_Memory_Resume_address       : CONSTANT
                                       address := 16#1000#;
CP_IOC_Command_Resume_address          : CONSTANT
                                       address := 16#1100#;
CP_Instruction_Memory_Resume_address   : CONSTANT
                                       address := 16#1200#;
CP_IOC_Interrupt_Code_Resume_address   : CONSTANT
                                       address := 16#1300#;
CP_Operand_Memory_Error_address        : CONSTANT
```

```
                                                address := 16#1400#;
   CP_Instruction_Memory_Error_address          : CONSTANT
                                                address := 16#1500#;
   CP_IOC_Command_Operand_Error_address         : CONSTANT
                                                address := 16#1600#;
   IOC_Memory_Error_address                     : CONSTANT
                                                address := 16#1700#;
   IPI_Fault_address                            : CONSTANT
                                                address := 16#1900#;
   IOC_Memory_Resume_address                    : CONSTANT
                                                address := 16#1A00#;
   Intercomputer_Timeout_address                : CONSTANT
                                                address := 16#1B00#;
   Confidence_Test_Fault_address                : CONSTANT
                                                address := 16#1C00#;
   CPU_IOC_Microprocessor_Stop_address          : CONSTANT
                                                address := 16#1D00#;
   Module_Interrupt_address                     : CONSTANT
                                                address := 16#1E00#;
   Power_Tolerance_Interrupt_address            : CONSTANT
                                                address := 16#1F00#;
   Class_III_Unhandled_address                  : CONSTANT
                                                address := 16#2800#;
   CP_Illegal_Instruction_Error_address         : CONSTANT
                                                address := 16#2200#;
   Privileged_Instruction_Error_address         : CONSTANT
                                                address := 16#2300#;
   Data_Pattern_Breakpoint_address              : CONSTANT
                                                address := 16#2400#;
   Operand_Breakpoint_Match_address             : CONSTANT
                                                address := 16#2500#;
   Operand_Read_address                         : CONSTANT
                                                address := 16#2600#;
   DCU_Status_Interrupt_address                 : CONSTANT
                                                address :=  16#2700#;
   Operand_Write_Protection_address             : CONSTANT
                                                address := 16#2900#;
   Operand_Limit_Protection_address             : CONSTANT
                                                address := 16#2A00#;
   Instruction_Breakpoint_Match_address         : CONSTANT
                                                address := 16#2B00#;
```

```
   -- RTAS interrupt addresses (16#2B01# .. 16#2B1F#)
   rtdebug_pseudo_interrupt : CONSTANT address := 16#2B01#;
   PMAids_pseudo_address    : CONSTANT address := 16#2B10#;

   RPD_Underflow_address                        : CONSTANT
                                                address := 16#2C00#;
   Instruction_Execute_Protection_address       : CONSTANT
                                                address := 16#2D00#;
   Instruction_Limit_Protection_address         : CONSTANT
                                                address := 16#2E00#;
   Precisely_Timed_Interrupts_address           : CONSTANT
                                                address := 16#2F00#;

   once_only_pti : CONSTANT duration := 0.0;
      -- Used to indicate that a PTI is not to be periodic.
   SUBTYPE pti_address IS address RANGE 16#2F01#..16#2F1F#;
   TYPE pti_state IS (active,inactive,unregistered);

   IOC_Illegal_CAR_Instruction    : CONSTANT address := 16#3000#;
   IOC_Memory_Protection          : CONSTANT address := 16#3100#;
   IOC_Channel_Function_Error     : CONSTANT address := 16#3300#;
   IOC_Illegal_Chain_Instruction  : CONSTANT address := 16#3400#;
   IOC_Confidence_Test_Fault      : CONSTANT address := 16#3800#;
   IOC_Breakpoint_Match           : CONSTANT address := 16#3900#;
   IOC_CP_Interrupt               : CONSTANT address := 16#3B00#;
   IOC_External_Interrupt_Monitor : CONSTANT address := 16#3C00#;
   IOC_External_Function_Monitor  : CONSTANT address := 16#3D00#;
   IOC_Output_Data_Monitor        : CONSTANT address := 16#3E00#;
   IOC_Input_Data_Monitor         : CONSTANT address := 16#3F00#;

   SUBTYPE IO_interrupts IS address RANGE
      IOC_Illegal_CAR_Instruction..IOC_Input_Data_Monitor;
   SUBTYPE channel_numbers IS INTEGER RANGE 0..63;

   --
   -- The following exceptions are provided as a "convention"
   -- whereby the Ada program can be compiled with all implicit
   -- checks suppressed (i.e. pragma SUPPRESS or equivalent),
   -- explicit checks included as necessary, the appropriate
   -- exception raised when required, and then the exception is
   -- either handled or the Ada program terminates.
   --
   ACCESS_CHECK          : EXCEPTION;
   DISCRIMINANT_CHECK    : EXCEPTION;
   INDEX_CHECK           : EXCEPTION;
   LENGTH_CHECK          : EXCEPTION;
   RANGE_CHECK           : EXCEPTION;
   DIVISION_CHECK        : EXCEPTION;
   OVERFLOW_CHECK        : EXCEPTION;
   ELABORATION_CHECK     : EXCEPTION;
   STORAGE_CHECK         : EXCEPTION;
```

```
   -- implementation-defined exceptions.
   UNRESOLVED_REFERENCE : EXCEPTION;
   SYSTEM_ERROR         : EXCEPTION;
   CAPACITY_ERROR       : EXCEPTION;
   -- The exception CAPACITY_ERROR is raised by the RTExec when
   -- Pre-RunTime specified resource limits are exceeded.


   FUNCTION ADDRESS_OF
   -- returns the system.address of the given Class III
   -- interrupt for the specified channel
      (interrupt   : IN IO_interrupts;
       for_channel : IN channel_numbers
      ) RETURN address;
      -- The address to be used in the
      -- representation (address) clause.

   PRAGMA INTERFACE (MACRO_NORMAL,ADDRESS_OF);


   FUNCTION "AND"
   -- returns the logical 32 bit 'AND' between two integers.
      (operand_a : IN integer;
       operand_b : IN integer
      ) RETURN integer;

   PRAGMA INTERFACE (MACRO_NORMAL, "AND");

   FUNCTION "NOT"
   -- returns the logical 32 bit 'NOT' of an integer.
      (operand_a : IN integer
      ) RETURN integer;

   PRAGMA INTERFACE (MACRO_NORMAL, "NOT");

   FUNCTION "OR"
   -- returns the logical 32 bit 'OR' between two integers.
      (operand_a : IN integer;
       operand_b : IN integer
      ) RETURN integer;

   PRAGMA INTERFACE (MACRO_NORMAL, "OR");
```

F.5 Character Set

Ada compilations may be expressed using the following characters in addition to the basic character set:

lower case letters:

a b c d e f g h i j k l m n o p q r s t u v w x y z

special characters:

! $ % ? @ [ ] ^ ` { } ~

The following transliterations are permitted:

a. Exclamation point for vertical bar,

b. Colon for sharp, and

c. Percent for double-quote.

F.6 Declaration and Representation Restrictions

Declarations are described in Section 3 of the Ada Language Reference Manual, and representation specifications are described in Section 13 of the Ada Language Reference Manual and discussed here.

In the following specifications, the capitalized word SIZE indicates the number of bits used to represent an object of the type under discussion. The upper case symbols D, L, R, correspond to those discussed in Section 3.5.9 of the Ada Language Reference Manual.

F.6.1 Integer Types

Integer types are specified with constraints of the form:

```
RANGE L..R
```

where:

```
R <= SYSTEM.MAX_INT & L >= SYSTEM.MIN_INT
```

For a prefix "t" denoting an integer type, length specifications of the form:

```
FOR t'SIZE USE n ;
```

may specify integer values n such that n in 2..64,

```
R <= 2**(n-1)-1 & L >= -(2**(n-1)-1)
```

or else such that

```
R <= (2**n)-1 & L >= 0
```

and 1 < n <= 31.

For a stand-alone object of integer type, a default SIZE of 32 is used when:

```
R <= 2**31 - 1 & L >= -(2**31 - 1)
```

Otherwise, a SIZE of 64 is used.

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

F.6.2 Floating Types

Floating types are specified with constraints of the form:

```
DIGITS D
```

where D is an integer in the range 1 through 15.

For a prefix "t" denoting a floating point type, length specifications of the form:

```
FOR t'SIZE USE n;
```

are permitted only when the integer value n = 32 for D <= 6 or N = 64 for 7 <= D <= 15.

F.6.3 Fixed Types

Fixed types are specified with constraints of the form:

```
DELTA D RANGE L..R
```

where:

$$\frac{\text{MAX (ABS(R), ABS(L))}}{\text{actual delta}} <= 2^{31} - 1.$$

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

For fixed point types, length specifications of the form:

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

are permitted only when N in 1 .. 32, if:

```
R - actual_delta <= 2**(N-1)-1 * actual_delta, and
L + actual_delta >= -2**(n-1) * actual)delta
```

or

```
R - actual_delta <= 2**(N)-1 * actual_delta, and
L >= 0
```

For stand-alone objects of fixed point type, a default size of 32 is used. For components of fixed point types within packed composite objects, the size from the length specification will be used.

For specifications of the form:

```
FOR t'SMALL USE n;
```

are permitted for any value of X, such that X <= D. X must be specified either as a base 2 value or as a base 10 value. Note that when X is specified as other than a power of 2, actual delta will still be the largest integreal power of two less than X.

F.6.4 Enumeration Types

In the absence of a representation specification for an enumeration type "t," the internal representation of t'FIRST is 0. The default size for a stand-alone object of enumeration type "t" is 32, so the internal representations of t'FIRST and t'LAST both fall within the range:

```
-(2**31 - 1) .. 2**31 - 1.
```

For enumeration types, length specifications of the form:

```
FOR t'SIZE USE n;
```

and/or enumeration representations of the form:

```
FOR t USE <aggregate>;
```

are permitted for n in 2..32, provided the representations and the SIZE conform to the relationship specified above.

Or else for n in 1..32, is supported for enumeration types and provides an internal representation of:

```
t'FIRST >= 0 .. t'LAST <= 2**(t'SIZE) - 1.
```

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

Enumeration representations for types derived from the predefined type STANDARD.BOOLEAN will not be accepted, but length specifications will be accepted.

F.6.5 Access Types

For access type, "t," length specifications of the form:

```
FOR t'SIZE USE n;
```

will not affect the runtime implementation of "t," therefore n = 32 is the only value permitted for SIZE, which is the value returned by the attribute.

For collection size specifications of the form:

```
FOR t'STORAGE_SIZE USE n;
```

for any value of "n" is permitted for STORAGE_SIZE (and that value will be returned by the attribute call). The collection size specification will affect the implementation of "t" and its collection at runtime by limiting the number of objects for type "t" that can be allocated.

The value of t'STORAGE_SIZE for an access type "t" specifies the maximum number of storage_units used for all objects in the collection for type "t." This includes all space used by the allocated objects, plus any additional storage required to maintain the collection.

F.6.6 Arrays and Records

For arrays and records, a length specification of the form:

```
FOR t'size USE n;
```

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

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

A record type representation specification may be used to describe the allocation of components in a record. Bits are numbered 0..31 from the right. Bit 32 starts at the right of the next higher numbered word. Each location specification must allow at least n bits of range, where n is large enough to hold any value of the subtype of the component being allocated. Otherwise, a diagnostic message of severity ERROR is generated. Components that are arrays, records, tasks, or access variables may not be allocated to specified locations. If a specification

of this form is entered, a diagnostic message of severity ERROR is generated.

For records, an alignment clause of the form:

```
AT MOD n
```

specify alignments of 1 word (word alignment) or 2 words (doubleword alignment).

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

F.6.7 Other Length Specifications

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

A length specification for a task type "t," of the form:

```
FOR t'STORAGE_SIZE USE n;
```

specifies the number of SYSTEM.STORAGE_UNITS that are allocated for the execution of each task object of type "t." This includes the runtime stack for the task object but does not include objects allocated at runtime by the task object. If a t'STORAGE_SIZE is not specified for a task type "t," the default value is 2K (words).

A length specification for a task type "t" of the form:

```
FOR t'SIZE USE n;
```

is allowable only for n = 32.

F.7 System Generated Names

Refer to Section 13.7 of the Ada Language Reference Manual and the section above on the Predefined Language Environment for a discussion of package SYSTEM.

The system name is chosen based on the target(s) supported, but it cannot be changed. In the case of Ada/L, the system name is ANUYK43.

F.8 Address Clauses

Refer to Section 13.5 of the Ada Language Reference Manual for a description of address clauses. All rules and restrictions described there apply. In addition, the following restrictions apply.

An address clause may designate a single task entry. Such an address clause is allowed only within a task specification compiled with the EXECUTIVE compiler option. The meaningful values of the simple_expression are the allowable interrupt entry addresses as defined in Table F-1. The use of other values will result in the raising of a PROGRAM_ERROR exception upon creation of the task.

If more than one task entry is equated to the same interrupt entry address, the most recently executed interrupt entry registration permanently overrides any previous registrations.

At most one address clause is allowed for a single task entry. Specification of more than one interrupt address for a task entry is erroneous.

Address clauses for objects and code other than task entries are allowed by the Ada/L target, but they have no effect beyond changing the value returned by the 'ADDRESS attribute call.

AN/UYK-43(V) Interrupt Summary			
Target-Computer Interrupt	ISC CODE	Interrupt Entry Address	Registration
CLASS 0			
Class I Unhandled Interrupt	None	16#0800#	
CLASS I			
Class II Unhandled Interrupt	None	16#1800#	
CP-Operand Memory Resume	16#0#	16#1000#	
CP-IOC Command Resume	16#1#	16#1100#	
CP-Instruction Memory Resume	16#2#	16#1200#	
CP-IOC Interrupt Code Resume	16#3#	16#1300#	
CP-Operand Memory Error	16#4#	16#1400#	
CP-Instruction Memory Error	16#5#	16#1500#	
CP-IOC Command/Operand Error	16#6#	16#1600#	
IOC Memory Error	16#7#	16#1700#	
IPI Fault	16#9#	16#1900#	
IOC Memory Resume	16#A#	16#1A00#	
Intercomputer Timeout	16#B#	16#1B00#	
CP Confidence Test Fault	16#C#	16#1C00#	
CPU/IOC Microprocessor Stop	16#D#	16#1D00#	
Module Interrupt	16#E#	16#1E00#	
Power Tolerance	16#F#	16#1F00#	
CLASS II			
Class III Unhandled Interrupt	None	16#2800#	
Interprocessor Interrupt	16#0#	16#2000#	UNDEFINABLE
Floating Point Error	16#1#	16#2100#	UNDEFINABLE
Illegal Instruction	16#2#	16#2200#	
Privileged Instruction Error	16#3#	16#2300#	
Data Pattern Breakpoint	16#4#	16#2400#	
Operand Address Breakpoint	16#5#	16#2500#	
Operand Read or Indirect Addressing	16#6#	16#2600#	
DCU Status Interrupt	16#7#	16#2700#	
Operand Write	16#9#	16#2900#	
Operand Limit	16#A#	16#2A00#	
Instruction Address Breakpoint	16#B#	16#2B00#	
RPD Underflow	16#C#	16#2C00#	
Instruction Execute	16#D#	16#2D00#	
Instruction Limit	16#E#	16#2E00#	
Monitor Clock	16#F#	16#2F00#	UNDEFINABLE
PTI	None	16#2F01# .. 16#2F1F#	

Table F-1a - Interrupt Entry Addresses

AN/UYK-43(V) Interrupt Summary

Target-Computer Interrupt	ISC CODE	Interrupt Entry Address	Registration
CLASS III			
IOC Illegal CAR Instruction	16#0#	16#30I0#	
IOC Memory Protection	16#1#	16#31IC#	
If the above interrupt is generated during CAR execution, no channel number is available. The interrupt will be translated to Class II Unhandled.			
UNDEFINED	16#2#	16#3200#	UNDEFINABLE
Channel Function Error	16#3#	16#33IC#	
IOC Illegal Chain Instruction	16#4#..16#7#	16#34IC#	
IOC Confidence Test Fault	16#8#	16#38IC#	
If the above interrupt is generated during CAR execution, no channel number is available. The interrupt will be translated to Class II Unhandled.			
IOC Breakpoint Match	16#9#	16#39IC#	
If the above interrupt is generated during CAR execution, no channel number is available. The interrupt will be translated to Class II Unhandled.			
IOC Monitor Clock	16#A#	16#3AI0#	UNDEFINABLE
IOC Processor Interrupt	16#B#	16#3BIC#	
External Interrupt Monitor	16#C#	16#3CIC#	
External Function Monitor	16#D#	16#3DIC#	
Output Data Monitor	16#E#	16#3EIC#	
Input Data Monitor	16#F#	16#3FIC#	
For class III interrupts, the following interpretations apply:			
IC => IOC, channel number where 16#00#..16#1F# indicates IOC 0, channel 16#00..16#1F#, 16#20#..16#3F# indicates IOC 1, channel 16#00..16#1F#			

Table F-1b - Interrupt Entry Addresses (Continued)

F.9 Unchecked Conversions

Refer to Section 13.10.2 of the Ada Language Reference Manual for a description of UNCHECKED_CONVERSION. It is erroneous if your Ada program performs UNCHECKED_CONVERSION when the source and target objects have different sizes.

F.10 Restrictions on the Main Subprogram

Refer to Section 10.1 (8) of the Ada Language Reference Manual for a description of the main subprogram. The subprogram designated as the main subprogram cannot have parameters. The designation as the main subprogram of a subprogram whose specification contains a formal_part results in a diagnostic of severity ERROR at link time.

The main subprogram can be a function, but the return value will not be available upon completion of the main subprogram's execution. The main subprogram may not be an import unit.

F.11 Input/Output

Refer to Section 14 of the Ada Language Reference Manual for a discussion of Ada Input/Output and to Section 12 of the Ada/L Run Time Environment Handbook for more specifics on the Ada/L Input/Output subsystem.

The Ada/L Input/Output subsystem provides the following packages: TEXT_IO, SEQUENTIAL_IO, DIRECT_IO, and LOW_LEVEL_IO. These packages execute in the context of the user-written Ada program task making the I/O request. Consequently, all of the code that processes an I/O request on behalf of the user-written Ada program executes sequentially. The package IO_EXCEPTIONS defines all of the exceptions needed by the packages SEQUENTIAL_IO, DIRECT_IO, and TEXT_IO. The specification of this package is given in Section 14.5 of the Ada Language Reference Manual. This package is visible to all of the constituent packages of the Ada/L I/O subsystem so that appropriate exception handlers can be inserted.

I/O in Ada/L is performed solely on external files. No allowance is provided in the I/O subsystem for memory resident files (i.e., files which do not reside on a peripheral device). This is true even in the case of temporary files. With the external files residing on the peripheral devices, Ada/L makes the further restriction on the number of files that may be open on an individual peripheral device.

Section 14.1 of the Ada Language Reference Manual states that all I/O operations are expressed as operations on objects of some file type, rather than in terms of an external file. File objects are implemented in Ada/L as access objects which point to a data structure called the File Control Block. This File Control Block is defined internally to each of the high-level I/O packages; its purpose is to represent an external file. The File Control Block contains all of the I/O-specific information about an external file needed by the high-level I/O packages to accomplish requested I/O operations.

F.11.1 Naming External Files

The naming conventions for external files in Ada/L are of particular importance to you. All of the system-dependent information needed by the I/O subsystem about an external file is contained in the file name. External files may be named using one of three file naming conventions: standard, temporary, and user-derived.

F.11.1.1 Standard File Names

The standard external file naming convention used in Ada/L identifies the specific location of the external file in terms of the physical device on which it is stored. For this reason, you should be aware of the configuration of the peripheral devices on the AN/UYK-43 at your particular site.

Standard file names consist of a six character prefix and a file name of up to twenty characters. The six character prefix has a predefined format. The first and second characters must be either "DK," "MT," or "TT," designating an AN/UYH-3(V) Recorder/Reproducer Set Magnetic Disk, the RD-358 Magnetic Tape Subsystem, or the AN/USQ-69 Data Terminal Set, respectively.

The third and fourth characters specify the channel on which the peripheral device is connected. Since there are sixty-four channels on the AN/UYK-43, the values for the third and fourth positions must lie in the range "00" to "63."

The range of values for the fifth position in the prefix (the unit number) depends upon the device specified by the characters in the first and second positions of the external file name. If the specified peripheral device is the AN/UYH-3 magnetic disk drive, the character in the fifth position must be one of the characters "0," "1," "2," or "3." This value determines which of the four disk units available on the AN/UYH-3 is to be accessed. If the specified peripheral device is the RD-358 magnetic tape drive, the character in the fifth position must be one of the characters "0," "1," "2," or "3." This value determines which of the four tape units available on the RD-358 is to be accessed. If the specified peripheral device is the AN/USQ-69 militarized display terminal, the character in the fifth position depends on the channel type. If the channel type is parallel then this character must be a "0." This is the only allowable value for the unit number when the AN/USQ-69 is connected to a parallel I/O channel. This is because the AN/USQ-69 may have only one unit on a parallel channel. If the channel type is serial then the character in the fifth position must be one of the characters "0," "1," "2," "3," "4," "5," "6," "7," or "8" (the character "8" will be used to specify a broadcast mode transmission). The AN/USQ-69 allows up to eight terminals to be daisy chained together when running on a serial channel.

The colon (":") is the only character allowed in the sixth position. If any character other than the colon is in this position, the file name will be considered non-standard and the file will reside on the default device defined during the elaboration of CONFIGURE_IO.

Positions seven through twenty-six are optional to your Ada program and may be used as desired. These positions may contain any printable character you choose in order to make the file name

more intelligible. Embedded blanks, however, are not allowed.

The location of an external file on a peripheral device is thus a function of the first six characters of the file name regardless of the characters that might follow. For example, if the external file "MT000:Old_Data" has been created and not subsequently closed, an attempt to create the external file "MT000:New_Data" will cause the exception DEVICE_ERROR (rather than NAME_ERROR or USE_ERROR) to be raised because the peripheral device on channel "00" and cartridge "0" is already in use.

You are advised that any file name beginning with "xxxxx:" (where x denotes any printable character) is assumed to be a standard external file name. If this external file name does not conform to the Ada/L standard file naming conventions, the exception NAME_ERROR will be raised.

F.11.1.2 Temporary File Names

Section 14.2.1 of the Ada Language Reference Manual defines a temporary file to be an external file that is not accessible after completion of the main subprogram. If the null string is supplied for the external file name, the external file is considered temporary. In this case, the high level I/O packages internally create an external file name to be used by the lower level I/O packages. The internal naming scheme used by the I/O subsystem is a function of the type of file to be created (text, direct or sequential) and the current date and time. This scheme is consistent with the requirement specified in the Ada Language Reference Manual that all external file names be unique.

The first two characters of the file name are "TX," "D_," or "S_." The next eight characters are the date (four characters for the year, two characters for the month, and two characters for the day). The remaining ten characters are the time (five for seconds and five for the fraction part of a second). For instance, the temporary external file name "D_19880331234598765" would be a DIRECT_IO file created March 31, 1988 at 12,345.98765 seconds.

F.11.1.3 User-Derived File Names

A random string containing a sequence of characters of length one to twenty may also be used to name an external file. External files with names of this nature are considered to be permanent external files. You are cautioned from using names which conform to the scheme used by the I/O subsystem to name temporary external files (see list item "b").

It is not possible to associate two or more internal files with the same external file. The exception USE_ERROR will be

raised if this restriction is violated.

F.11.2 The FORM Specification for External Files

Section 14.2.1 of the Ada Language Reference Manual defines a string argument called the FORM, which supplies system-dependent information that is sometimes required to correctly process a request to create or open a file. In Ada/L, the string argument supplied to the FORM parameter on calls to CREATE and OPEN is retained while the file is open, so that calls to the function FORM can return the string to your Ada program. FORM options specified on calls to CREATE have the effects stated below. FORM options specified on calls to OPEN have no effect.

Ada/L only allows a FORM parameter when a file is open or created on the RD-358 tape drive. A USE_ERROR will be raised when a FORM parameter is associated with any other Ada/L system device. The FORM parameter specifically controls the positioning and formatting of the tape prior to tape I/O operations. This section identifies the arguments of the FORM parameter. Refer to Section 14.2.1 of the Ada Language Reference Manual and to Section 12 of the Ada/L Run-Time Environment Handbook for more detail on the use of the FORM parameter.

The FORM parameter is a string literal of which a maximum of twenty characters is processed. If the supplied FORM string is longer than the maximum allowed (20 characters), the exception USE_ERROR will be raised. The string literal is interpreted as a sequence of arguments. If you wish to utilize the default arguments, a FORM parameter need not be supplied.

Only the first two arguments within the string are processed. All following characters or arguments will cause the USE_ERROR to be raised. The arguments are not case sensitive. The arguments must be separated by at least one delimiter. A legal delimiter consists of a comma or blank. Extra delimiters are ignored. Of the recognized arguments, at most one formatting and one positioning argument are allowed. If conflicting arguments are used, the exception USE_ERROR will be raised.

Positioning arguments allow control of tape before its use. The following positioning arguments are available:

a. REWIND - specifies that a rewind will be performed prior to the requested operation.

b. NOREWIND - specifies that the tape remains positioned as is.

c. APPEND - specifies that the tape be positioned at the logical end of tape (LEOT) prior to the requested operation. The LEOT is denoted by two consecutive tape_marks.

The formatting argument specifies information about tape mat. If a formatting argument is not supplied, the file is assumed to contain a format header record determined by the ALS/N I/O system. The following formatting arguments are available:

a. NOHEAD - specifies that the designated file has no header record. This argument allows the reading and writing of tapes used on computer systems using different header formats.

b. DENSITY_800 - specifies that the tape is written/read with a density of 800 BPI. This is the default density. Attempting to write/read files of different density on the same tape will cause unpredictable results.

c. DENSITY_1600 - specifies that the tape is written/read with a density of 1600 BPI. Attempting to write/read files of different density on the same tape will cause unpredictable results.

F.11.3 File Processing

Processing allowed on Ada/L files is influenced by the characteristics of the underlying device. The following restrictions apply:

a. Only one file may be open on an individual RD-358 tape drive at a time.

b. The attempt to CREATE a file with the mode IN_FILE is not supported since there will be no data in the file to read.

F.11.4 Text Input/Output

TEXT_IO is invoked by your Ada program to perform sequential access I/O operations on text files (i.e., files whose content is in human-readable form). TEXT_IO is not a generic package and, thus, its subprograms may be invoked directly from your program, using objects with base type or parent type in the language-defined type character. TEXT_IO also provides the generic packages INTEGER_IO, FLOAT_IO, FIXED_IO, and ENUMERATION_IO for the reading and writing of numeric values and enumeration values. The generic packages within TEXT_IO require an instantiation for a given element type before any of their subprograms are invoked. The specification of this package is given in Section 14.3.10 of the Ada Language Reference Manual.

The implementation-defined type COUNT that appears in Section 14.3.10 of the Ada Language Reference Manual is defined as follows:

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

The implementation-defined subtype FIELD that appears in Section 14.3.10 of the Ada Language Reference Manual is defined as follows:

```
subtype FIELD is INTEGER range 0...INTEGER'LAST;
```

At the beginning of program execution, the STANDARD_INPUT file and the STANDARD_OUTPUT file are open, and associated with the files specified by you at export time. Additionally, if a program terminates before an open file is closed (except for STANDARD_INPUT and STANDARD_OUTPUT), the last line you added to the file may be lost; if the file is on magnetic tape, the file structure on the tape may be inconsistent.

A program is erroneous if concurrently executing tasks attempt to perform overlapping GET and/or PUT operations on the same terminal. The semantics of text layout as specified in the Ada Language Reference Manual, Section 14.3.2, (especially the concepts of current column number and current line) cannot be guaranteed when GET operations are interweaved with PUT operations. A program which relies on the semantics of text layout under those circumstances is erroneous.

For TEXT_IO processing, the line length can be no longer than 1022 characters. An attempt to set the line length through SET_LINE_LENGTH to a length greater than 1022 will result in USE_ERROR.

F.11.5 Sequential Input/Output

SEQUENTIAL_IO is invoked by your Ada program to perform I/O on the records of a file in sequential order. The SEQUENTIAL_IO package also requires a generic instantiation for a given element type before any of its subprograms may be invoked. Once the package SEQUENTIAL_IO is made visible, it will perform any service defined by the subprograms declared in its specification. The specification of this package is given in Section 14.2.3 of the Ada Language Reference Manual.

The following restrictions are imposed on the use of the package SEQUENTIAL_IO:

a. SEQUENTIAL_IO cannot be instantiated with an unconstrained array type.

b. SEQUENTIAL_IO cannot be instantiated with a record type with discriminants with no default values.

c. Ada/L does not raise DATA_ERROR on a read operation if the data input from the external file is not of the instantiating type (see the Ada Language Reference Manual, Section 14.2.2).

F.11.6 Direct Input/Output

DIRECT_IO is invoked by your Ada program to perform I/O of the records of a file in an arbitrary order. The package DIRECT_IO requires a generic instantiation for a given element type before any of its subprograms may be invoked. Once the package DIRECT_IO is made visible, it will perform any service defined by the subprograms declared in its specification. The specification of this package is given in Section 14.2.5 of the Ada Language Reference Manual.

The following restrictions are imposed on the use of the package DIRECT_IO:

a. DIRECT_IO cannot be instantiated with an unconstrained array type.

b. DIRECT_IO cannot be instantiated with a record type with discriminants with no default values.

c. Ada/L does not raise DATA_ERROR on a read operation if the data input from the external file is not of the instantiating type (see the Ada Language Reference Manual, Section 14.2.4).

F.11.7 Low Level Input/Output

LOW_LEVEL_IO is invoked by your Ada program to initiate physical operations on peripheral devices, and thus executes as part of a program task. Requests made to LOW_LEVEL_IO from your program are passed through the RTEXEC_GATEWAY to the channel programs in CHANNEL_IO. Any status check or result information is the responsibility of the invoking subprogram and can be obtained from the subprogram RECEIVE_CONTROL within LOW_LEVEL_IO.

The package LOW_LEVEL_IO allows your Ada program to send I/O commands to the I/O devices (using SEND_CONTROL) and to receive status information from the I/O devices (using RECEIVE_CONTROL). A program is erroneous if it uses LOW_LEVEL_IO to access a device that is also accessed by high-level I/O packages such as SEQUENTIAL_IO and TEXT_IO. The following is excerpted from the package LOW_LEVEL_IO.

```
SUBTYPE channel_range IS INTEGER RANGE 0..63;
   -- Range of values allowed for channel number.

SUBTYPE device_str IS STRING;
   -- To be passed to CHANNEL_IO for future implementations
   -- of logical path name.  The string will be ignored until
   -- logical path name support is added.

SUBTYPE btc_int IS INTEGER RANGE 0..16383;
   -- Passes transfer counts to/from IO_MANAGEMENT/RTEXEC.

SUBTYPE io_functions IS INTEGER RANGE 0..20;
   -- Specifies the I/O function to be performed by LOW_LEVEL_IO.
   -- The following table shows the values associated with device
   -- and device functions available.
```

```
------------------------------------------------------------
-- VALUE -- DEVICE -- FUNCTION
------------------------------------------------------------
--    0       RD-358    Normal Read
--    1       RD-358    Read with Search data
--    2       RD-358    Normal Write
--    3       RD-358    Send EF Command
--    4       RD-358    Initialize Channel
--
--    0       UYH-3     Read with 2 word EF
--    1       UYH-3     Read with 1 word EF
--    2       UYH-3     Write
--    3       UYH-3     Send 1 word EF Command
--    4       UYH-3     Send 2 word EF Command
--    5       UYH-3     Send 1 word EF Command (Same as function 3)
--    6       UYH-3     Initialize Channel
--
--    0       USQ-69    Read
--    1       USQ-69    Write
--    2       USQ-69    Write (Same as function 1)
--    3       USQ-69    Send Command
--    4       USQ-69    Initialize Channel
------------------------------------------------------------

TYPE cap_block IS
   -- Information that can be found in IOC control memory on
   -- a per channel/ per function basis.
   RECORD
      cap           : INTEGER;  -- CAP register.
      instruct_base : INTEGER;  -- CAP instruction base.
      index         : INTEGER;  -- CAP index register.
      accumulator   : INTEGER;  -- CAP accumulator register.
      status        : INTEGER;  -- CAP status register.
      buffer_base   : INTEGER;  -- CAP buffer base.
      bcw           : INTEGER;  -- CAP buffer control word.
      operand_base  : INTEGER;  -- CAP operand base.
   END RECORD;

TYPE short_rec_control_block  IS
   -- I/O control block sent to LOW_LEVEL_IO as a parameter
   -- when calling subprogram RECEIVE_REQUEST.
   RECORD
      channel     : low_level_io.channel_range;
         -- Specifies channel of interest.
      ei_word     : INTEGER;
         -- External interrupt returned by the peripheral device.
   END RECORD;
```

```
TYPE receive_control_block  IS
   -- I/O control block sent to LOW_LEVEL_IO as a parameter
   -- when calling subprogram RECEIVE_REQUEST.
   RECORD
      data   : low_level_io.short_rec_control_block;
         -- Channel and ei_word.
      ef     : low_level_io.cap_block;
         -- External Function CAP information.
      output : low_level_io.cap_block;
         -- Output CAP information.
      ei     : low_level_io.cap_block;
         -- External Interrupt CAP information.
      input  : low_level_io.cap_block;
         -- Input CAP information.
   END RECORD;

TYPE send_control_block IS
   -- I/O control block sent to LOW_LEVEL_IO as a parameter
   -- when calling subprogram SEND_REQUEST.
   RECORD
      function_pos   : low_level_io.io_functions;
         -- Indicates which I/O function is to be requested
         -- of LOW_LEVEL_IO.
      channel        : low_level_io.channel_range;
         -- Specifies channel number.
      transfer_count : low_level_io.btc_int;
         -- Buffer transfer count for I/O operation.
      buffer_addr    : system.address;
         -- Address of data buffer.
      command_1      : INTEGER;
         -- Holds the first word of the external
         -- function for the device.
      command_2      : INTEGER;
         -- Holds the second word of the external
         -- function for the device.
      filler_1       : INTEGER;
         -- Passes additional information to
         -- CHANNEL_IO (such as the terminal_address
         -- for the USQ-69 device).
   END RECORD;
```

```
PROCEDURE SEND_CONTROL
-- Passes I/O control information to a procedure in
-- IO_MANAGEMENT/RTEXEC in order to send data to the
-- specified device.

   (device : IN low_level_io.device_str := "";
             -- This string will be ignored until
             -- logical path names are implemented.
    data   : IN low_level_io.send_control_block
             -- I/O control block for send request.
    );


PROCEDURE RECEIVE_CONTROL
-- Passes I/O control information to a procedure in
-- IO_MANAGEMENT/RTEXEC in order to obtain the status of
-- the I/O operation.

   (device : IN     low_level_io.device_str := "";
             -- This string will be ignored until
             -- logical path names are implemented.
    data   : IN OUT low_level_io.receive_control_block
             -- I/O control block for receive request.
   );


PROCEDURE RECEIVE_CONTROL
-- Passes I/O control information to a procedure in
-- IO_MANAGEMENT/RTEXEC in order to obtain the status of
-- the I/O operation.

   (device : IN     low_level_io.device_str := "";
             -- This string will be ignored until
             -- logical path names are implemented.
    data   : IN OUT low_level_io.short_rec_control_block
             -- I/O control block for receive request.
   );
```

F.12 System Defined Exceptions

In addition to the exceptions defined in the Ada Language Reference Manual, this implementation pre-defines the exceptions shown in Table F-2 below.

Name	Significance
CAPACITY_ERROR	Raised by the Run-Time Executive when Pre-Runtime specified resource limits are exceeded.
PAST_PTI_TIME	Raised by the PTI support package if the PTI start time is greater than the current CALENDAR.CLOCK.
SYSTEM_ERROR	Serious error detected in underlying AN/UYK-43 operating system.
UNREGISTERED_PTI	Raised by the PTI support package if the PTI's state is returned as "unregistered".
UNRESOLVED_REFERENCE	Attempted call to a subprogram whose body is not linked into the executable program image.

Table F-2 - System Defined Exceptions

F.13 Machine Code Insertions

The Ada language permits machine code insertions as defined in Section 13.8 of the Ada Language Reference Manual. This section describes the specific details for writing machine code insertions as provided by the predefined package MACHINE_CODE.

You may, if desired, include AN/UYK-43 instructions within an Ada program. This is done by including a procedure in the program which contains only record aggregates defining machine instructions. The package MACHINE_CODE, included in the system program library, contains type, record, and constant declarations which are used to form the instructions. Each field of the aggregate contains a field of the resulting machine instruction. These fields are specified in the order in which they appear in the actual instruction. Since the AN/UYK-43 has several different formats for instructions, package MACHINE_CODE defines different types for each of these formats. For each of the fields which must have a certain value for a given instruction (i.e., part of the opcode), package MACHINE_CODE defines a constant to use for that field.

The following procedure implements a floating point exponential. Note that this actual procedure would not be used, because package MATH_PACK implements the same operation in a more efficient manner.

```
with machine_code; use machine_code;
procedure floating_point_exponential
          (x  : FLOAT;
           ex : OUT FLOAT) is

BEGIN
   formatI'(f_LA,1,3,6,0,0,0);
          -- LA  A1,B6+0

   formatV'(f_FEX,1,f2_FEX,2,0,0,0,f6_FEX);
          -- FEX A1,A2

   formatI'(f=>f_SA,a=>2,k=>3,b=>6,i=>0,s=>0,y=>1);
          -- SA  A2,B6+1
END;
```

Note that either positional or names aggregates may be used. Whenever a field does not appear in the MACRO/L instruction, it must be filled in with 0, since no missing fields are allowed. For formatI instructions, when k=0, the s and y field are collapsed and used together. For your convenience, an additional record type, formatIi, for immediate, can be used to define the s and y fields as a single 16-bit quantity. This quantity is defined as an unsigned integer, so if a negative number x is desired, one should instead put the number x + 65535.

Table F-3 contains a list of MACRO/L instructions and their Ada/L machine code equivalents, sorted by MACRO/L mnemonic.

MACRO/L		Ada/L
AA	a,y,k,b,s	formatI'(f_AA,a,k,b,i,s,y);
AB	a,y,k,b,s	formatI'(f_AB,a,k,b,i,s,y);
AEI	a,sy,b	formatIi'(f_AEI,a,k_AEI,b,i,sy);
ALP	a,y,b,s	formatI'(f_ALP,a,k_ALP,b,i,s,y);
ANA	a,y,k,b,s	formatI'(f_ANA,a,k,b,i,s,y);
ANB	a,y,k,b,s	formatI'(f_ANB,a,k,b,i,s,y);
ATSF	a,b	formatV'(f_ATSF,a,f2_ATSF,b,0,0,0,f6_ATSF);
BC	ak,y,b,s	formatIa'(f_BC,a,k,b,i,s,y);
BS	ak,y,b,s	formatIa'(f_BS,a,k,b,i,s,y);
BZ	ak,y,b,s	formatIa'(f_BZ,a,k,b,i,s,y);
C	a,y,k,b,s	formatI'(f_C,a,k,b,i,s,y);
CB		formatIVA'(f_CB,a_CB,0,i_CB);
CBN	a,n	formatIVC'(f_CBN,a,f4_CBN,n);
CBR	a,b	formatV'(f_CBR,a,f2_CBR,b,0,0,0,f6_CBR);
CCT	a,b	formatIVA'(f_CCT,a,b,i_CCT);
CE		formatIVA'(f_CE,a_CE,0,i_CE);
CG	a,y,k,b,s	formatI'(f_CG,a,k,b,i,s,y);
CHCL	a,y,b,s	formatI'(f_CHCL,a,k_CHCL,b,i_CHCL,s,y);
CL	a,y,k,b,s	formatI'(f_CL,a,k,b,i,s,y);
CM	a,y,k,b,s	formatI'(f_CM,a,k,b,i,s,y);
CMPS	a,b	formatV'(f_CMPS,a,f2_CMPS,b,0,0,0,f6_CMPS);
CNT	a,y,b,s	formatI'(f_CNT,a,k_CNT,b,i,s,y);
CRB	a,b	formatV'(f_CRB,a,f2_CRB,b,0,0,0,f6_CRB);
CXI	a,y,k,b,s	formatI'(f_CXI,a,k,b,i,s,y);
D	a,y,k,b,s	formatI'(f_D,a,k,b,i,s,y);
DA	a,y,b,s	formatI'(f_DA,a,k_DA,b,i,s,y);
DAN	a,y,b,s	formatI'(f_DAN,a,k_DAN,b,i,s,y);
DC	a,y,b,s	formatI'(f_DC,a,k_DC,b,i,s,y);
DJNZ	a,y,k,b,s	formatIII'(f_DJNZ,a,f3_DJNZ,k,b,i,s,y);
DJZ	a,y,k,b,s	formatIII'(f_DJZ,a,f3_DJZ,k,b,i,s,y);
DL	a,y,b,s	formatI'(f_DL,a,k_DL,b,i,s,y);
DS	a,y,b,s	formatI'(f_DS,a,k_DS,b,i,s,y);
DSP	a,b,m	formatV'(f_DSP,a,f2_DSP,b,0,0,m,f6_DSP);
EECM		formatIVA'(f_EECM,a_EECM,0,i_EECM);
ESCM		formatIVA'(f_ESCM,a_ESCM,0,i_ESCM);
ETCM		formatIVA'(f_ETCM,a_ETCM,0,i_ETCM);
FA	a,y,b,s	formatI'(f_FA,a,k_FA,b,i,s,y);
FAC	a,b	formatV'(f_FAC,a,f2_FAC,b,0,0,0,f6_FAC);
FAN	a,y,b,s	formatI'(f_FAN,a,k_FAN,b,i,s,y);
FANR	a,y,b,s	formatI'(f_FANR,a,k_FANR,b,i,s,y);
FAR	a,y,b,s	formatI'(f_FAR,a,k_FAR,b,i,s,y);
FAS	a,b	formatV'(f_FAS,a,f2_FAS,b,0,0,0,f6_FAS);
FAT	a,b	formatV'(f_FAT,a,f2_FAT,b,0,0,0,f6_FAT);
FD	a,y,b,s	formatI'(f_FD,a,k_FD,b,i,s,y);
FDR	a,y,b,s	formatI'(f_FDR,a,k_FDR,b,i,s,y);

Table F-3a - Machine Code Instructions

MACRO/L		Ada/L
FEX	a,b	formatV'(f_FEX,a,f2_FEX,b,0,0,0,f6_FEX);
FLN	a,b	formatV'(f_FLN,a,f2_FLN,b,0,0,0,f6_FLN);
FLTF	a,n	formatV'(f_FLTF,a,f2_FLTF,n,0,0,0,f6_FLTF);
FM	a,y,b,s	formatI'(f_FM,a,k_FM,b,i,s,y);
FMR	a,y,b,s	formatI'(f_FMR,a,k_FMR,b,i,s,y);
FPA	a,b	formatV'(f_FPA,a,f2_FPA,b,0,0,0,f6_FPA);
FPD	a,b	formatV'(f_FPD,a,f2_FPD,b,0,0,0,f6_FPD);
FPM	a,b	formatV'(f_FPM,a,f2_FPM,b,0,0,0,f6_FPM);
FPS	a,b	formatV'(f_FPS,a,f2_FPS,b,0,0,0,f6_FPS);
FSA	a,b	formatV'(f_FSA,a,f2_FSA,b,0,0,0,f6_FSA);
FSC	a,b	formatV'(f_FSC,a,f2_FSC,b,0,0,0,f6_FSC);
FSD	a,b	formatV'(f_FSD,a,f2_FSD,b,0,0,0,f6_FSD);
FSM	a,b	formatV'(f_FSM,a,f2_FSM,b,0,0,0,f6_FSM);
FSS	a,b	formatV'(f_FSS,a,f2_FSS,b,0,0,0,f6_FSS);
FTSL	a,b	formatV'(f_FTSL,a,f2_FTSL,b,0,0,0,f6_FTSL);
HA	a,b	formatIVA'(f_HA,a,b,0);
HAEI	a,b	formatIVA'(f_HAEI,a,b,i_HAEI);
HAI		formatIVA'(f_HAI,0,0,0);
HALT		formatIVA'(f_HALT,0,0,i_HALT);
HAN	a,b	formatIVA'(f_HAN,a,b,0);
HAND	a,b	formatIVA'(f_HAND,a,b,i_HAND);
HC	a,b	formatIVA'(f_HC,a,b,0);
HCB	a,b	formatIVA'(f_HCB,a,b,0);
HCL	a,b	formatIVA'(f_HCL,a,b,0);
HCM	a,b	formatIVA'(f_HCM,a,b,0);
HCP	a	formatIVA'(f_HCP,a,0,0);
HCRC	a,b	formatIVA'(f_HCRC,a,b,i_HCRC);
HD	a,b	formatIVA'(f_HD,a,b,0);
HDCP	a	formatIVA'(f_HDCP,a,0,0);
HDLC	a,m	formatIVB'(f_HDLC,a,m);
HDRS	a,m	formatIVB'(f_HDRS,a,m);
HDRZ	a,m	formatIVB'(f_HDRZ,a,m);
HDSF	a,b	formatIVA'(f_HDSF,a,b,0);
HLB	a,b	formatIVA'(f_HLB,a,b,0);
HLC	a,m	formatIVB'(f_HLC,a,m);
HLCA	a,b	formatIVA'(f_HLCA,a,b,i_HLCA);
HLCI	af4,b	formatIVA_1'(f_HLCI,af4,b,i_HLCI);
HLCT	af4,b	formatIVA_1'(f_HLCT,af4,b,i_HLCT);
HLTC	a,b	formatIVA'(f_HLTC,a,b,i_HLTC);
HM	a,b	formatIVA'(f_HM,a,b,0);
HOR	a,b	formatIVA'(f_HOR,a,b,0);
HPEI	a,b	formatIVA'(f_HPEI,a,b,i_HPEI);
HPI		formatIVA'(f_HPI,0,0,0);
HR	a,b	formatV'(f_HR,a,f2_HR,b,0,0,0,f6_HR);
HRS	a,m	formatIVB'(f_HRS,a,m);

Table F-3b - Machine Code Instructions (Continued)

MACRO/L		Ada/L
HRT	a,b	formatIVA'(f_HRT,a,b,0);
HRZ	a,m	formatIVB'(f_HRZ,a,m);
HSCA	a,b	formatIVA'(f_HSCA,a,b,i_HSCA);
HSCI	af4,b	formatIVA_1'(f_HSCI,af4,b,i_HSCI);
HSCT	af4,b	formatIVA_1'(f_HSCT,af4,b,i_HSCT);
HSF	a,b	formatIVA'(f_HSF,a,b,0);
HSIM	a,b	formatIVA'(f_HSIM,a,b,i_HSIM);
HSTC	a,b	formatIVA'(f_HSTC,a,b,i_HSTC);
HST1		formatIVA'(f_HST1,a_HST1,b_HST1,i_HST1);
HST2		formatIVA'(f_HST2,a_HST2,b_HST2,i_HST2);
HST3		formatIVA'(f_HST3,a_HST3,b_HST3,i_HST3);
HST4		formatIVA'(f_HST4,a_HST4,b_HST4,i_HST4);
HSTD	a,b	formatIVA'(f_HSTD,a,b,i_HSTD);
HSTV	a,b	formatIVA'(f_HSTV,a,b,i_HSTV);
HV	a,b	formatV'(f_HV,a,f2_HV,b,0,0,0,f6_HV);
HWFI		formatIVA'(f_HWFI,0,0,i_HWFI);
HXOR	a,b	formatIVA'(f_HXOR,a,b,0);
IBSC	a	formatIVA'(f_IBSC,a,0,i_IBSC);
IILM	a	formatIVA'(f_IILM,a,0,i_IILM);
IO	a,y,b,s	formatI'(f_IO,a,k_IO,b,i,s,y);
IOCL	a	formatIVA'(f_IOCL,a,0,i_IOCL);
IOCR	a	formatIVA'(f_IOCR,a,0,i_IOCR);
IOCS	a	formatIVA'(f_IOCS,a,0,i_IOCS);
IOT	a,b,m	formatV'(f_IOT,a,f2_IOT,b,0,0,m,f6_IOT);
IPI	y,b,s	formatI'(f_IPI,a_IPI,k_IPI,b,i,s,y);
IR		formatI'(f_IR,0,k_IR,0,0,0,0);
IRMMS	a,b	formatIVA'(f_IRMMS,a,b,i_IRMMS);
IRMSR	a,b	formatIVA'(f_IRMSR,a,b,i_IRMSR);
ISMSR	a,b	formatIVA'(f_ISMSR,a,b,i_ISMSR);
ISP	a,b,m	formatV'(f_ISP,a,f2_ISP,b,0,0,m,f6_ISP);
J	y,k,b,s	formatIII'(f_J,a_J,f3_J,k,b,i,s,y);
JBNZ	a,y,k,b,s	formatIII'(f_JBNZ,a,f3_JBNZ,k,b,i,s,y);
JC	a,y,k,b,s	formatIII'(f_JC,a,f3_JC,k,b,i,s,y);
JE	y,k,b,s	formatIII'(f_JE,a_JE,f3_JE,k,b,i,s,y);
JEP	a,y,k,b,s	formatIII'(f_JEP,a,f3_JEP,k,b,i,s,y);
JG	y,k,b,s	formatIII'(f_JG,a_JG,f3_JG,k,b,i,s,y);
JGE	y,k,b,s	formatIII'(f_JGE,a_JGE,f3_JGE,k,b,i,s,y);
JL	y,k,b,s	formatIII'(f_JL,a,f3_JL,k,b,i,s,y);
JLE	y,k,b,s	formatIII'(f_JLE,a_JLE,f3_JLE,k,b,i,s,y);
JLT	y,k,b,s	formatIII'(f_JLT,a_JLT,f3_JLT,k,b,i,s,y);
JN	a,y,k,b,s	formatIII'(f_JN,a,f3_JN,k,b,i,s,y);
JNE	y,k,b,s	formatIII'(f_JNE,a_JNE,f3_JNE,k,b,i,s,y);
JNF	y,k,b,s	formatIII'(f_JNF,a_JNF,f3_JNF,k,b,i,s,y);
JNW	y,k,b,s	formatIII'(f_JNW,a_JNW,f3_JNW,k,b,i,s,y);
JNZ	a,y,k,b,s	formatIII'(f_JNZ,a,f3_JNZ,k,b,i,s,y);

Table F-3c - Machine Code Instructions (Continued)

MACRO/L		Ada/L
JOF	y,k,b,s	formatIII'(f_JOF,a_JOF,f3_JOF,k,b,i,s,y);
JOP	a,y,k,b,s	formatIII'(f_JOP,a,f3_JOP,k,b,i,s,y);
JP	a,y,k,b,s	formatIII'(f_JP,a,f3_JP,k,b,i,s,y);
JS	sy,k,b	formatIII'(f_JS,0,f3_JS,k,b,i,s,y);
JSC	a,y,k,b,s	formatIII'(f_JSC,a,f3_JSC,k,b,i,s,y);
JW	y,k,b,s	formatIII'(f_JW,a_JW,f3_JW,k,b,i,s,y);
JZ	a,y,k,b,s	formatIII'(f_JZ,a,f3_JZ,k,b,i,s,y);
LA	a,y,k,b,s	formatI'(f_LA,a,k,b,i,s,y);
LB	a,y,k,b,s	formatI'(f_LB,a,k,b,i,s,y);
LBJ	a,y,k,b,s	formatIII'(f_LBJ,a,f3_LBJ,k,b,i,s,y);
LBMP	a,y,b,s	formatI'(f_LBMP,a,k_LBMP,b,i,s,y);
LCI	ak,y,b,s	formatIa'(f_LCI,ak,b,i,s,y);
LCM1	y,b,s	formatI'(f_LCM1,a_LCM1,k_LCM1,b,i_LCM1,s,y);
LCM2	y,b,s	formatI'(f_LCM2,a_LCM2,k_LCM2,b,i_LCM2,s,y);
LCM3	y,b,s	formatI'(f_LCM3,a_LCM3,k_LCM3,b,i_LCM3,s,y);
LCM4	y,b,s	formatI'(f_LCM4,a_LCM4,k_LCM4,b,i_LCM4,s,y);
LCMA	y,b,s	formatI'(f_LCMA,a_LCMA,k_LCMA,b,i_LCMA,s,y);
LCMP	y,b,s	formatI'(f_LCMP,a_LCMP,k_LCMP,b,i,s,y);
LCMT	y,b,s	formatI'(f_LCMT,a_LCMT,k_LCMT,b,i_LCMT,s,y);
LCPA	a,y,b,s	formatI'(f_LCPA,a,k_LCPA,b,i,s,y);
LCRA	a,y,b,s	formatI'(f_LCRA,a,k_LCRA,b,i,s,y);
LCT	ak,y,b,s	formatIa'(f_LCT,ak,b,i,s,y);
LDIF	a,y,k,b,s	formatI'(f_LDIF,a,k,b,i,s,y);
LECM		formatIVA'(f_LECM,a_LECM,0,i_LECM);
LIBP	a,y,b,s	formatI'(f_LIBP,a,k_LIBP,b,i,s,y);
LIM	a,sy,b	formatIi'(f_LIM,a,k_LIM,b,i,sy);
LIMP	a,y,b,s	formatI'(f_LIMP,a,k_LIMP,b,i,s,y);
LISR	a,b	formatIVA'(f_LISR,a,b,i_LISR);
LLP	a,y,b,s	formatI'(f_LLP,a,k_LLP,b,i,s,y);
LLPN	a,y,b,s	formatI'(f_LLPN,a,k_LLPN,b,i,s,y);
LM	a,y,k,b,s	formatI'(f_LM,a,k,b,i,s,y);
LNA	a,y,k,b,s	formatI'(f_LNA,a,k,b,i,s,y);
LRR	a,m	formatV'(f_LRR,a,f2_LRR,0,0,0,m,f6_LRR);
LRRA	a,b,i	formatIVA'(f_LRRA,a,b,i);
LSCM		formatIVA'(f_LSCM,a_LSCM,0,i_LSCM);
LSUM	a,y,k,b,s	formatI'(f_LSUM,a,k,b,i,s,y);
LTCM		formatIVA'(f_LTCM,a_LTCM,0,i_LTCM);
LXB	a,y,k,b,s	formatI'(f_LXB,a,k,b,i,s,y);
M	a,y,k,b,s	formatI'(f_M,a,k,b,i,s,y);
MS	a,y,b,s	formatI'(f_MS,a,k_MS,b,i,s,y);
NLP	a,y,b,s	formatI'(f_NLP,a,k_NLP,b,i,s,y);
OR	a,y,b,s	formatI'(f_OR,a,k_OR,b,i,s,y);
PEI	a,sy,b	formatIi'(f_PEI,a,k_PEI,b,i,sy);
PFCD		formatIVA'(f_PFCD,0,0,i_PFCD);
PFCE		formatIVA'(f_PFCE,0,0,i_PFCE);

Table F-3d - Machine Code Instructions (Continued)

MACRO/L		Ada/L
PFR	a,y,b,s	formatI'(f_PFR,a,k_PFR,b,i_PFR,s,y);
PIE		formatIVA'(f_PIE,0,0,i_PIE);
PMM	y,b,s	formatI'(f_PMM,a_PMM,k_PMM,b,i,s,y);
PMR	y,b,s	formatI'(f_PMR,a_PMR,k_PMR,b,i,s,y);
POP	a,b	formatV'(f_POP,a,f2_POP,b,0,0,0,f6_POP);
PUSH	a,b	formatV'(f_PUSH,a,f2_PUSH,b,0,0,0,f6_PUSH);
RA	a,y,k,b,s	formatI'(f_RA,a,k,b,i,s,y);
RALP	a,y,b,s	formatI'(f_RALP,a,k_RALP,b,i,s,y);
RAN	a,y,k,b,s	formatI'(f_RAN,a,k,b,i,s,y);
RCCR	y,b,s	formatI'(f_RCCR,a_RCCR,k_RCCR,b,i,s,y);
RD	a,y,k,b,s	formatI'(f_RD,a,k,b,i,s,y);
RI	a,y,k,b,s	formatI'(f_RI,a,k,b,i,s,y);
RIOAS	a,b	formatIVA'(f_RIOAS,a,b,i_RIOAS);
RISR	a,b	formatIVA'(f_RISR,a,b,i_RISR);
RJ	y,k,b,s	formatIII'(f_RJ,a_RJ,f3_RJ,k,b,i,s,y);
RJC	a,y,k,b,s	formatIII'(f_RJC,a,f3_RJC,k,b,i,s,y);
RJSC	a,y,k,b,s	formatIII'(f_RJSC,a,f3_RJSC,k,b,i,s,y);
RLP	a,y,b,s	formatI'(f_RLP,a,k_RLP,b,i,s,y);
RMMS	a,bi	formatIVA'(f_RMMS,a,b,i);
RMS	a,y,b,s	formatI'(f_RMS,a,k_RMS,b,i,s,y);
RMSR	y,b,s	formatI'(f_RMSR,a_RMSR,k_RMSR,b,i,s,y);
RNLP	a,y,b,s	formatI'(f_RNLP,a,k_RNLP,b,i,s,y);
ROR	a,y,b,s	formatI'(f_ROR,a,k_ROR,b,i,s,y);
RP	a,sy,b	formatIi'(f_RP,a,k_RP,b,i,sy);
RPD	y,k,b,s	formatI'(f_RPD,a_RPD,k,b,i,s,y);
RRR	a,m	formatV'(f_RRR,a,f2_RRR,0,0,0,m,f6_RRR);
RSC	a,y,b,s	formatI'(f_RSC,a,k_RSC,b,i,s,y);
RSD	a	formatIVA'(f_RSD,a,0,i_RSD);
RXOR	a,y,b,s	formatI'(f_RXOR,a,k_RXOR,b,i,s,y);
SA	a,y,k,b,s	formatI'(f_SA,a,k,b,i,s,y);
SB	a,y,k,b,s	formatI'(f_SB,a,k,b,i,s,y);
SBN	a,n	formatIVC'(f_SBN,a,f4_SBN,n);
SBPC	a,y,k,b,s	formatI'(f_SBPC,a,k,b,i,s,y);
SC	a,y,b.s	formatI'(f_SC,a,k_SC,b,i,s,y);
SCI	ak,y,b,s	formatIa'(f_SCI,ak,b,i,s,y);
SCMA	y,b,s	formatI'(f_SCMA,a_SCMA,k_SCMA,b,i_SCMA,s,y);
SCMP	y,b,s	formatI'(f_SCMP,a_SCMP,k_SCMP,b,i,s,y);
SCMT	y,b,s	formatI'(f_SCMT,a_SCMT,k_SCMT,b,i_SCMT,s,y);
SCM1	y,b,s	formatI'(f_SCM1,a_SCM1,k_SCM1,b,i_SCM1,s,y);
SCM2	y,b,s	formatI'(f_SCM2,a_SCM2,k_SCM2,b,i_SCM2,s,y);
SCM3	y,b,s	formatI'(f_SCM3,a_SCM3,k_SCM3,b,i_SCM3,s,y);
SCM4	y,b,s	formatI'(f_SCM4,a_SCM4,k_SCM4,b,i_SCM4,s,y);
SCPA	a,y,b,s	formatI'(f_SCPA,a,k_SCPA,b,i,s,y);
SCRA	a,y,b,s	formatI'(f_SCRA,a,k_SCRA,b,i,s,y);
SCSR	y,b,s	formatI'(f_SCSR,a_SCSR,k_SCSR,b,i,s,y);

Table F-3e - Machine Code Instructions (Continued)

MACRO/L		Ada/L
SCT	ak,y,b,s	formatIa'(f_SCT,ak,b,i,s,y);
SDIF	a,y,b,s	formatI'(f_SDIF,a,k_SDIF,b,i,s,y);
SDMC	a	formatIVA'(f_SDMC,a,0,i_SDMC);
SIBP	a,y,b,s	formatI'(f_SIBP,a,k_SIBP,b,i,s,y);
SIMC	a,b	formatIVA'(f_SIMC,a,b,i_SIMC);
SIMP	a,y,b,s	formatI'(f_SIMP,a,k_SIMP,b,i,s,y);
SIRC	a,b	formatIVA'(f_SIRC,a,b,i_SIRC);
SITC	a,b	formatIVA'(f_SITC,a,b,i_SITC);
SLP	a,y,b,s	formatI'(f_SLP,a,k_SLP,b,i,s,y);
SM	a,y,k,b,s	formatI'(f_SM,a,k,b,i,s,y);
SMCC	a	formatIVA'(f_SMCC,a,0,i_SMCC);
SMSR	y,b,s	formatI'(f_SMSR,a_SMSR,k_SMSR,b,i,s,y);
SNA	a,y,k,b,s	formatI'(f_SNA,a,k,b,i,s,y);
SRRA	a,b,i	formatIVA'(f_SRRA,a,b,i);
SSUM	a,y,b,s	formatI'(f_SSUM,a,k_SSUM,b,i,s,y);
STAF	a,b	formatV'(f_STAF,a,f2_STAF,b,0,0,0,f6_STAF);
STSB	ak,y,b,s	formatIa'(f_STSB,ak,b,i,s,y);
SXB	a,y,k,b,s	formatI'(f_SXB,a,k,b,i,s,y);
TBN	a,n	formatIVC'(f_TBN,a,f4_TBN,n);
TR	a,b	formatV'(f_TR,a,f2_TR,b,0,0,0,f6_TR);
TSBN	a,n	formatIVC'(f_TSBN,a,f4_TSBN,n);
TSF	y,b,s	formatI'(f_TSF,0,k_TSF,b,i,s,y);
TSM	bi	formatIVA'(f_TSM ,a_TSM,b,i);
TV	a,b	formatV'(f_TV,a,f2_TV,b,0,0,0,f6_TV);
WFBP	a,y,b,s	formatI'(f_WFBP,a,k_WFBP,b,i_WFBP,s,y);
WFM	a,y,b,s	formatI'(f_WFM,a,k_WFM,b,i_WFM,s,y);
XOR	a,y,b,s	formatI'(f_XOR,a,k_XOR,b,i,s,y);
XR	y,b,s	formatI'(f_XR,0,k_XR,b,i,s,y);
XRL	y,b,s	formatI'(f_XRL,0,k_XRL,b,i,s,y);
XS	sy,b	formatI'(f_XS,a_XS,k_XS,b,i,sy);

Table F-3f - Machine Code Instructions (Continued)

Option	Function
EXECUTIVE	Enables pragma EXECUTIVE and allows visibility to units which have been compiled with the RTE_ONLY option. Default: NO_EXECUTIVE
MEASURE	Generates code to monitor execution frequency at the subprogram level for the current unit. Default: NO_MEASURE
NO_CHECKS	NO_CHECKS suppresses all run-time error checking. CHECKS provides run-time error checking. Default: CHECKS
NO_CODE_ON_WARNING	NO_CODE_ON_WARNING means no code is generated when there is a diagnostic of severity WARNING or higher. CODE_ON_WARNING generates code only if there are no diagnostics of a severity higher than WARNING. Default: CODE_ON_WARNING
NO_CONTAINER_GENERATION	NO_CONTAINER_GENERATION means that no container is produced even if there are nc diagnostics. CONTAINER_GENERATION produces a container if diagnostic serverity permits. Default: CONTAINER_GENERATION

Table F-4a - Special Processing Options

Option	Function
NO_DEBUG	If NO_DEBUG is specified, only that information needed to link, export and execute the current unit is included in the compiler output. With the DEBUG option in effect, internal representations and additional symbolic information are stored in the container. Default: DEBUG
NO_TRACE_BACK	Disables the location of source exceptions that are not handled by built-in exception handlers. Default: TRACE_BACK
OPTIMIZE	Enables global optimizations in accordance with the optimization pragmas specified in the source program. If the pragma OPTIMIZE is not included, the optimizations emphasize TIME over SPACE. When NO_OPTIMIZE is in effect, no global optimizations are performed, regardless of the pragmas specified. Default: NO_OPTIMIZE
RTE_ONLY	Restricts visibility of this unit only to those units compiled with the EXECUTIVE option. Default: NO_RTE_ONLY

Table F-5b - Special Processing Options (Continued)

Option	Function
ATTRIBUTE	Produces a Symbol Attribute Listing. (Produces an attribute cross-reference listing when both ATTRIBUTE and CROSS_REFERENCE are specified.) Default: NO_ATTRIBUTE.
CROSS_REFERENCE	Produces a Cross-Reference Listing. (Produces an attribute cross-reference listing when both ATTRIBUTE and CROSS_REFERENCE are specified.) Default: NO_CROSS_REFERENCE.
DIAGNOSTICS	Produces a Diagnostic Summary Listing. Default: NO_DIAGNOSTICS.
MACHINE_CODE	Produces a Machine Code Listing if code is generated. Code is generated when CONTAINER_GENERATION option is in effect and (1) there are no diagnostics of severity ERROR, SYSTEM, or FATAL, and/or (2) NO_CODE_ON_WARNING option is in effect and there are no diagnostics of severity higher than NOTE. A diagnostic of severity NOTE is reported when a Machine Code Listing is requested and no code is generated. OCTAL is an additional option that may be used with MACHINE_CODE to output ocatal values on the listing instead of hex values. Default: NO_MACHINE_CODE.
NOTES	Includes diagnostics of NOTE severity level in the Source Listing. Default: NO_NOTES.
SOURCE	Produces listing of Ada source statements. Default: NO_SOURCE.
SUMMARY	Produces a Summary Listing; always produced when there are errors in the compilation. Default: NO_SUMMARY.

Table F-6 - Ada/L Listing Control Options

Option	Function
MSG	Sends error messages and the Diagnostic Summary Listing to the file specified. The default is to send error messages and the Diagnostic Summary Listing to Message Output (usually the terminal).
OUT	Sends all selected listings to a single file specified. The default is to send listings to Standard Output (ususally the terminal).

Table F-7 - Control_Part (Redirection) Options

BLANK PAGE

PAGES DELIBRATELY LEFT BLANK NOT
ESSENTIAL TO REPORT PER TELECON
DAN LEHMANN ADA OFC IDA
DC 10/29/91

BLANK PAGE

PAGES DELIBRATELY LEFT BLANK NOT
ESSENTIAL TO REPORT PER TELECON
DAN LEHMANN ADA OFC IDA
DC 10/29/91

F.16 Linker Options

Option	Function
DEBUG	Produces a linked container to be debugged. Default: NO_DEBUG.
MEASURE	Produces a linked container to be analyzed. Default: NO_MEASURE.
PARTIAL	Produces an incomplete linked container with unresolved references. Default: NO_PARTIAL.
RTL_SELECTIVE	Similar to the SELECTIVE option except that is only refers to RTLIB units. This option is not supported during phase links. Default: NO_RTL_SELECTIVE.
SEARCH	Explicitly searches for the units to be included in the linked container. Default: SEARCH for final links; NO_SEARCH for phase links.
SELECTIVE	Maps into the program only the subprograms called by the main subprogram. Default: SELECTIVE for final links; NO_SELECTIVE for phase links.

Table F-10 - Ada/L Linker Special Processing Options

Option	Function
No option	Linker summary listing always produced.
DEBUGMAP	Generates a debugmap listing. Default: NO_DEBUGMAP.
ELAB_LIST	Generates an elaboration order listing. Default: NO_ELAB_LIST.
LOADMAP	Generates a loadmap listing. Default:NO_LOADMAP.
LOCAL_SYMBOLS	Generates a symbols listing with all internal as well as external definitions in the program. LOCAL_SYMBOLS is to be used in conjunction with the SYMBOLS option. If LOCAL_SYMBOLS is specified with NO_SYMBOLS, a WARNING is produced and the SYMBOLS option is activated. Default: NO_LOCAL_SYMBOLS
SYMBOLS	Produces a Linker symbols listing. Default: NO_SYMBOLS.
UNITS	Produces a Linker units listing. Default: NO_UNITS.

Table F-11 - Linker Listings Options

Option	Function
MSG	Sends error messages to the file specified. The default is to send error messages to Message Output (usually the terminal).
OUT	Sends all selected listings to the single file specified. The default is to send listings to Standard Output (usually the terminal).

Table F-12 - Control_Part (Redirection) Options

F.17 Exporter Options

Option	Function
DEBUG	Permits the generation of a load module with all debugging facilities available. When NO_DEBUG is specified or is in effect by default, no debugging facilities are made available. Export the program for debugging with either the Run-Time Debugger (RTD) or the Embedded Target Debugger (ETD). Default: NO_DEBUG.
DYNAMIC	Deferred.
LOAD	Deferred.
MEASURE	Permits the generation of a load module with all performance measurement facilities available. When NO_MEASURE is specified or is in effect by default, no performance measurement facilities are made available. Default: NO_MEASURE
REV0	In conjunction with the SIM_IMAGE argument to the IMAGE named parameter, this option specifies production of a Target System File suitable for input to Revision 0 of SIM/L and PORTAL/43.

Table F-13 - Ada/L Special Processing Options

Option	Function
MSG	Sends error messages to the file specified. The default is to send error messages to Message Output (usually the terminal).
OUT	Sends all selected listings to the single file specified. The default is to send listings to Standard Output (usually the terminal).

Table F-14 - Control_Part (Redirection) Options

Option	Function
DEBUGMAP	Generates a segment-by-segment listing that describes how the units are mapped onto hardware. Default: NO_DEBUGMAP.
LOADMAP	Generates a listings that describes how the units are mapped onto the hardware. Default: NO_LOADMAP.
LOCAL_SYMBOLS	As an option in addition to SYMBOLS listing, causes the symbols listing to include all internal as well as external definitions in the program. Default: NO_LOCAL_SYMBOLS
NO_DETAILED	Suppresses the listing of subprograms contained within each EXEC psect in the DEBUGMAP and LOADMAP listings. Default: DETAILED.
RTEXEC	Produces executive listings instead of application listings. It can only be used with the LOADMAP and DEBUGMAP options (e.g., /LOADMAP/RTEXEC). Default: NO_RTEXEC
SYMBOLS	Generates a symbols listing of all external definitions in the program. Default: NO_SYMBOLS.
UNITS	Generates a listing of all units. Default: NO_UNITS.

Table F-15 - Ada/L Exporter Listing Options